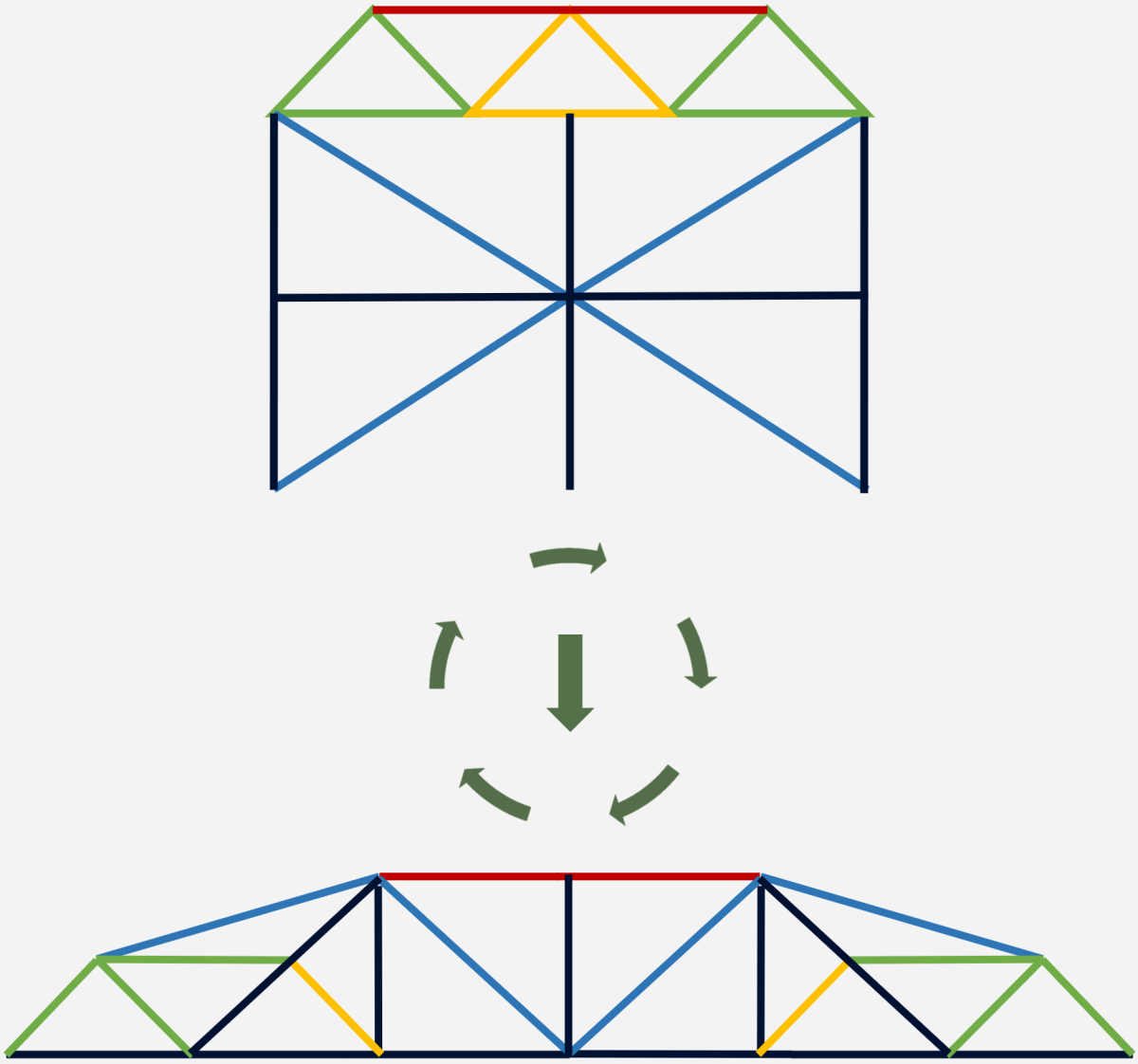
Technische Universiteit Delft

Truss Topology Optimization with Reused Steel Elements



T. P. van Gelderen

TU Delft Delft University of Technology

Aronsohn

Truss Topology Optimization with Reused Steel Elements

An Optimization Tool for Designing Steel Trusses with a Set of Reclaimed Elements

by

T. P. van Gelderen

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on June 17 , 2021.

Student number:	4389689	
Project duration:	November 9, 2020 – June 17, 2021	
Thesis committee:	Dr. Ir. M. A. N. Hendriks,	TU Delft, chair
	Ir. C. A. Slui,	Aronsohn
	Ir. L. P. L. van der Linden,	TU Delft
	Ir. S. Pasterkamp,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft Delft University of Technology

Aronsohn

Preface

T. P. van Gelderen

This Msc. thesis concludes my masters in Building Engineering with specialisation Structural Design at the faculty of Civil Engineering and Geosciences at the Delft University of Technology. After starting my bachelor in 2014, I will end my student days by this thesis to obtain my diploma. A journey which I look back on with great joy. Parametric engineering piqued my interest during the first year of my master. Combined with interests in climate change and circularity, this research topic exactly matched my preferences. I would like to thank Lennert van der Linden for taking time to exchange ideas and helping me transform these ideas into my final research topic.

I would like to thank Aronsohn for their hospitality during the covid pandemic for offering me a workplace at their office in Rotterdam. A special thanks to Casimir Slui for his close involvement during my research. Weekly meetings at the office gained new insights and increased motivation. I would also like to thank Rutger Jansen, Paul Lagendijk and Nemad Nowrowzi for their help and games of table football during the lunch break.

I would like to thank Max Hendriks, Lennert van der Linden, Sander Pasterkamp and Casimir slui for being part of my committee, attending microsoft teams meetings and progress meetings and providing feedback and help via email. Your curiosity made me do this research with much joy.

Last but not least, I would like to thank my family, friends and room mates for their support and motivation during the entire process of doing my research.

I wish you a pleasant and interesting reading!

Abstract

The building industry is a large contributor to the emission of greenhouse gasses, the production of steel and cement alone is responsible for 10% of all global emissions. Combined with scarcity of raw materials, recycling and reuse is becoming more important. Over 90% of all steel is collected and recycled. However, much energy is required for processing the steel. Reuse of structures and components requires less processing and thus less energy and could be much more beneficial than recycling. Many challenges arise when designing with reuse as a result of which it is still rarely put in practice.
Buildings are not designed for reuse making disassembly and reconstruction difficult. Organized documentation of materials and elements of existing buildings for reuse is not widely put in practice making reuse less accessible. A lack of confidence of quality of reclaimed elements holds clients back. Transport and storage of reused elements is difficult and expensive and designing with reuse involves a complex design process. The latter is simplified in this paper by answering the following main research question:

- "To what extend can a topology optimization method be used to simplify the complex design process with reused elements for the design of steel trusses?"

To answer this question a tool has been developed for the design of steel trusses with reused elements. Designs are generated by performing a topology optimization. Volume of structures is minimized and average unity check and reuse percentage maximized. The member adding scheme of He et al. (2019), based on the ground structure method of Dorn (1964), is used as a basis and adapted and extended with new functions. An existing method for designing with reused elements of Brütting et al. (2019b) is used for inspiration. The member adding scheme is a discrete topology optimization method in which size, connectivity and number of elements is variable.

The developed optimization method consists a number of iterating steps. First, topology is optimized after which elements from a database of reclaimed elements are assigned to the optimized members as efficient as possible. Capacity utilization of reused elements is optimized taking availability of elements into account. Elements not available from stock any more are considered as new elements with optimized cross sections. The third step is to penalize inefficient and new members in the designs. A penalty system is developed which determines penalty's based on assigned and optimized cross sections. Penalty's are determined and applied every iteration and avoid members to be present in the design in the next and or further iterations. Threshold numbers are determined to avoid efficient members being penalized. Every iteration the design is verified for virtual strain, efficiency and reuse percentage. In the fourth step, when no violation of requirements occurs, designs are presented to the user.

The optimization tool contains many parameters for values of penalty's and threshold numbers. Varying these parameters can result in a variance of designs. Preference could be given to designs with least volume, least number of members, highest percentage of reuse or highest average unity check. Architectural appearance could also be of importance.

In conclusion, the design tool is useful for designing with reused elements. The discrete topology optimization method of He et al. (2019) proved to be an effective method to automatize and simplify the design process. Designs may not be global optima but many design alternatives can be presented to the user. The developed optimization method is an improvement of the method for designing with reused elements of Brütting et al. (2019b). With limited complexity and large availability of elements, designs have been generated with an average unity check close to the maximum unity check. With increasing complexity effectiveness of the design tool decreases. Complexity is influenced by grid size, number of possible elements, availability of individual and different elements and number of point loads.

A case study has been performed redesigning the steel trusses of the roof structure of Ahoy with reclaimed elements of the steel structure of Provinciehuis Zuid-Holland. The case study contained a complex design problem but by varying parameters different efficient designs could be obtained. For the design of 1 and 2 trusses reuse percentages of 87% to 100% could be obtained. Respectively 23 of 139 available elements and

23 of 62 available elements were used in the designs. Compared to the existing structure, large volume reductions were achieved. However, these volume reductions are not realistic. Wind load and material factors were not considered. The optimized designs are only verified for compression and tensile stresses. Buckling, nodal instability and self-weight are not taken into account. Designs contain many intersecting and overlapping elements all taking part of the load. Compared to standard optimized designs for the trusses of Ahoy with new elements, volume increase could be limited to 20-30%.

Volume increase or reductions of designs with reused elements compared to designs with new elements depend on the number, suitability and diversity of available elements. This emphasizes the importance of organized documentation of buildings available for reuse. Designing with reuse should be made more accessible. It should be less difficult to obtain data of existing buildings to experiment with designing with reused elements.

Applicability of the design tool should be increased by including constraints for buckling, nodal instability, intersection and overlap of elements. Self-weight should be included either. An analysis of a reduction of embodied carbon, energy and construction costs could give insight in attractiveness of designing with reused elements.

List of symbols

List of symbols

a	[-]	Cross section
Acsp	[-]	Accumulating cross section penalty
Acspf	[-]	Accumulating cross section penalty factor
ANmp	[-]	Accumulating new member penalty
ANmpf	[-]	Accumulating new member penalty factor
amin	[m3∗1e-3]	Set of small cross sections
Cn	[-]	Connected nodes
csp	[-]	Cross section penalty
cspf	[-]	Cross section penalty factor
CU	[-]	Capacity utilization
CUf	[-]	Capacity utilization factor
ϵ	[-]	Virtual strain
f	[KN]	Load
i	[-]	Index number
itr	[-]	Iteration
jc	[-]	Joint cost
L	[m]	Length
$Ł_{cut}$	[m]	Maximum cutting length
$Ł_{short}$	[m]	Maximum length a member may be short
maxArea	[m3∗1e-3]	Maximum available cross section
maxUC	[-]	Maximum allowed unity check
mems	[-]	Members
minCU	[-]	Threshold number for csp
minFinalCU	[-]	Threshold number for Acsp
MM	[-]	Matrix method
n	[-]	Number of availability
Nmp	[-]	New member penalty
Nd	[-]	Node
PML	[-]	Possible member list
PSML	[-]	Possible stock member list
q	[KN]	Internal load
sf	[-]	Scaling factor
st	[N/mm2]	Maximum tensile stress
sc	[N/mm2]	Maximum compression stress
u	[mm]	Virtual displacements
UC	[-]	Unity check
vol	[m3∗1e-3]	Volume

Table 1: List of symbols

Contents

1

Introduction: Reuse and optimization in the building industry

1.1. Carbon footprint

Globally the building industry is responsible for 30% of all carbon emissions. Materials share a large contribution to the total embodied carbon. In the European Union 50% of all material is used for buildings and infrastructure (Service, 2013), the fabrication of building materials alone already produces 10% of the total CO2 emission from which about half for the production of steel and half for the production of cement (Favier et al., 2018).
Much effort is put into reducing operational costs and making buildings carbon neutral by improving insulation for example, however, little attention is given to reducing the total embodied carbon in the load bearing structure. Large volumes of material and intensive manufacturing processes have a major contribution to the embodied energy and carbon (Hoxha et al., 2017).

Reducing carbon emissions urges for the use of green energy to fabricate construction materials and for recycling and reuse of construction elements, a circular economy. Already one third of all produced steel comes from recycled scrap but still two third is made from iron ore. Steel is magnetic and therefore easily to separate from other waste (Allwood, 2016). About 90% of all steel waste is collected and recycled. However, a lot of waste is already created during the production of steel products, a quarter of the finished steel is cut off and recycled again (Allwood, 2016). Recycling reduces the amount of required raw materials but still requires much energy reducing the total carbon footprint only marginally. Recycling could also lead to deterioration of materials, recycling and crushing concrete can only be used for purposes such as road constructions (Brütting et al., 2019a). Therefore, reuse could be much more beneficial since no energy is required for manufacturing and quality can be maintained, for steel about 60% less energy is required for processing reused elements than recycled elements (Ferland, 2006).

1.2. Reuse of structures and components

Complete structures or components can be reclaimed and reused for similar or different purposes, reuse can be divided into three categories (Brütting et al., 2019a).

1. Adaptive reuse.
2. System reuse.
3. Component reuse.

With adaptive reuse complete structures are reused on site. This type of reuse is common practice with heritage structures where the original structure is extended with new structures (Gorgolewski, 2008). System reuse implicates the reuse of structures which are disassembled and remounted at a new location. Examples are modular and demountable structures such as scaffolding and tent structures which are easy to disassemble (Brütting et al., 2019a). With component reuse the original structure is not retained and rebuilt. Individual

components are reused in new structures with different arrangements. This method of reuse is more complex and best applicable for timber and steel elements with homogeneous properties and connections which allow easy disassembly (Brütting et al., 2019a).

1.3. Challenges with reuse

Reuse is not new and has been applied for centuries. The ancient Greek used stones and other materials from ruins to construct new buildings, materials were expensive and labour was cheap (Brütting et al., 2019a). In the United Kingdom however, only 4% of steel is reused compared to 92% which is recycled (Hopkinson et al., 2018). Nowadays labour can be expensive and the economically most attractive methods for renovation, construction and demolition do not involve reuse. However, with natural resources running low and ever increasing populations and economies recycling and reuse is becoming more important. Materials costs are increasing and environmental importance makes especially reuse economically more attractive (Gorgolewski, 2008). Existing buildings are regarded as huge reservoirs of materials and components available for reuse, demolition is considered as a loss of potential resources and profit (Kohler and Hassler, 2002). A market for reuse of steel will develop if selling steel for reuse becomes more profitable than selling crap (Dunant et al., 2017).

A challenge is that buildings are not designed for reuse which makes disassembly and reconstruction difficult (Hopkinson et al., 2018). In (Crowther, 2001) the concept of Design for Deconstruction or Disassembly is described, in which demolition and disassembly of buildings is considered already in the design phase to avoid demolition waste. A perfect illustration of this concept is demountable buildings which can be deconstructed and reconstructed at a new location without any waste of material (Waldmann, 2017). An example is the Triodos Bank in the Netherlands, an energy neutral building constructed completely out of wood. Connections are made with screws which allow all elements to be reused (Wainwright, 2020).

Other challenges and barriers for reuse are the lack of confidence of quality (Dunant et al., 2017), assessment of availability, organization of transport, supply and storage and the complex process of designing with reused components.
Inspecting reclaimed components and ensuring quality is important for the application of reused materials in new structures. However, materials should by fully capable of meeting requirements if they were under working load during their lifetime (Hopkinson et al., 2018). In a survey for the reasons for demolition of 148 buildings in North America, 35% of the buildings were demolished because of area redevelopment and 35% due to the building's physical conditions. From the 10 steel buildings in this survey 50% were demolished because they were not suitable for the anticipated use and 30% due to physical conditions of the building. However, the primary factor for the bad physical conditions of the building was a lack of maintenance. Only 5% of all buildings had a specific problem with the structural system, from the steel buildings none of them (Horst, 2004). All load bearing elements were still in good condition and could be reused in the load bearing structure of new buildings meeting all requirements.

Efficient reuse of construction elements depends on organized documentation of their availability in order to establish a market where contractors can buy materials available for salvage (Gorgolewski, 2008). In 2017 Thomas Rau founded Madaster, an online platform for the documentation of materials in a building. A material library is created with the identity of materials and their location. Complete buildings can be imported as an IFC or excel file on their platform, showing quantities and classification of all materials and elements included in the building. This platform makes the reuse of construction elements much more accessible (Rau, 2017).

With component reuse supply can be an issue. When components from different structures from different locations must be transported to the building site, items are not delivered on time or with the wrong size and length. A potential method to avoid this would be to buy an old building on site and reuse its components. However, storage space on site or nearby is required and with small buildings only a limited set of components is available for reuse (Gorgolewski, 2008).

Designing with reclaimed components adds a new level of complexity to the design process. In the traditional process the required length and size of components are specified based on an architect's proposal. Standard sizes are used and an efficient design is established. However, when a stock of reclaimed items is used for an architect's design, the required length and sizes may not be available resulting in oversized structures (Gorgolewski, 2008). A more efficient method would be to reverse the traditional design process, first the

length and size of all available components is identified after which an efficient design is made. This design process involves a complex optimization problem of which component should be where in the new structure. Reusing components for similar purposes would help making an efficient design (Gorgolewski, 2008).

1.4. Structural optimization

The goal of reusing and recycling is to minimize the production of new steel products and the production of steel from iron ore. Both important methods to reduce the carbon footprint of new buildings. Another method to reduce the required amount of steel in a new structure is reducing volume by optimization layout, size and number of structural elements. In general optimization can be divided into three categories:

1. Shape optimization
2. Size optimization
3. Topology optimization

Distinction between these methods is made in terms of design variables. With shape optimization layout is determined after which quantity of material is optimized (Ahmad and Voruganti, 2020), coordinates of nodes are variable (Liu et al., 2020). Size optimization consist the optimization of geometrical dimensions of a fixed shape and layout (Ahmad and Voruganti, 2020), cross sections of members can be changed (Liu et al., 2020). Topology optimization includes both size and shape optimization, location and connectivity of material is variable (Liu et al., 2020). Illustrations of all three methods are provided in Fig. 1.1.

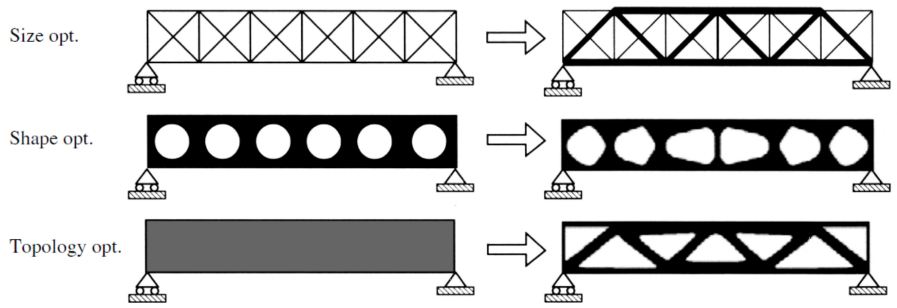


Figure 1.1: Shape, size and topology Optimization (Gebisa and Lemu, 2017)

Topology optimization can be divided into two categories, discrete and continuum optimization (Kentli, 2020). With discrete topology optimization connectivity, number and position of members is optimized. In continuum optimization, it is not placement of members but placement of material that is optimized, material can have a density of 1 or 0 representing material or no material (Ahmad and Voruganti, 2020). An example of both methods is given in Fig. 1.2.

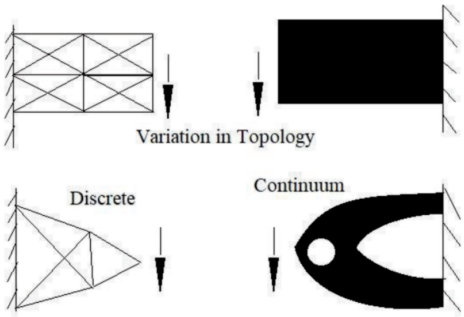


Figure 1.2: Discrete and continuum topology optimization (Kentli, 2020)

Structures optimized with topology optimization, or so called benchmark solutions, can have volume reductions of over 300% (Gilbert and Shepherd, 2019). The first discrete topology optimization method, the ground

structure method, was introduced by Dorn (1964) and since then many research has been done. However, the application of optimized structures is still limited in the building industry. The design of such structures is often too complex to fabricate in practice (Fairclough and Gilbert, 2020). Methods are being developed to perform optimizations with more discrete elements.

1.5. Optimization and reuse

A challenge with reuse is the complex design process which often results in oversized structures. Optimizing the design with the aim to minimize volume could simplify the design process, increase efficiency and compensate for volume increase due to inefficient components present in the design. In this paper an optimization method for the design of steel trusses with a stock of reclaimed components is developed. A tool is programmed in Python, the 'member adding scheme' of He et al. (2019) is used as a basis.

Outline Objectives, research questions and goals are provided in Chapter 2. Promising existing topology optimization methods for optimizing with reused elements and an existing method combining topology optimization with reused elements are described and compared in Chapter 3, concluded is that member adding scheme of He et al. (2019) is most suitable. In Chapter 4 the development of the tool for designing with reused steel elements is described with all adaptations and additions to the member adding scheme of He et al. (2019). Examples and illustrations of the effectiveness and problems of individual and/or combinations of adaptations and additions are provided in Chapter 5. Included in this chapter are guidelines for performing successful optimizations and an extensive example for a truss of 22m subjected to a point load at the centre. A case study is performed in Chapter 6, the trusses of the new roof structure of Ahoy are redesigned with reclaimed elements of the demolished Provinciehuis Zuid-Holland. Conclusions and recommendations are given in Chapter 8.

2

Research content

2.1. Problem statement and objective

The production of new materials and elements for the building industry poses two problems, high emissions of greenhouse gases and scarcity of raw materials. Two methods to reduce the amount of new material in the load bearing structure of new buildings are:

- Recycle and reuse materials and components from old buildings.
- Minimize volume of new structures by optimizing the design.

The objective of this research is to combine both methods by developing a tool designing steel trusses with reused elements by performing a topology optimization. Existing optimization techniques will be adapted and or combined to develop a new optimization tool. Not included in this research is how elements will be dissembled and new connections will be made. It is assumed that physical properties of the reused steel elements have not been downgraded.

2.2. Research goal

The main goal of topology optimization is to minimize volume of structures. With a stock of reused steel elements least volume benchmark solutions will not be achieved. The goal of this research is to design steel trusses with reused steel elements as efficient as possible by performing a topology optimization. Capacity utilization of elements and reuse percentage is maximized and volume of the structure minimized.

2.3. Research questions

The main research question is:

- "To what extend can a topology optimization method be used to simplify the complex design process with reused elements for the design of steel trusses?"

Sub questions are:

1. “Which existing topology optimization method is most suitable to use for the proposed optimization method”
2. “Which adaptations and additions must be made to the existing optimization method to develop the proposed optimization method”

2.4. Structure report

An overview of the structure of the report is given in Fig. 2.1. The report contains 8 chapters which can be divided in 5 parts. Relevance of the research is motivated in Chapter 1 and Chapter 2 in part 1. Part 2 contains theory on topology optimization in Chapter 3 after which subquestion 1 is concluded. The optimization tool for designing with reused elements is developed and tested in Chapter 4 and Chapter 5 in part 3. Testing concept versions of the design tool in preliminary development stages acquired insights for further development: Chapter 4 and Chapter 5 were written simultaneously. Subquestion 2 is concluded in part 3 in the beginning of Chapter 4 together with a concluding example at the end of Chapter 5 .
A case study is performed in part 4 in Chapter 6 to test the optimization tool to its limits. Discussion, conclusions and recommendations are provided in part 5 in Chapter 7 and Chapter 8.

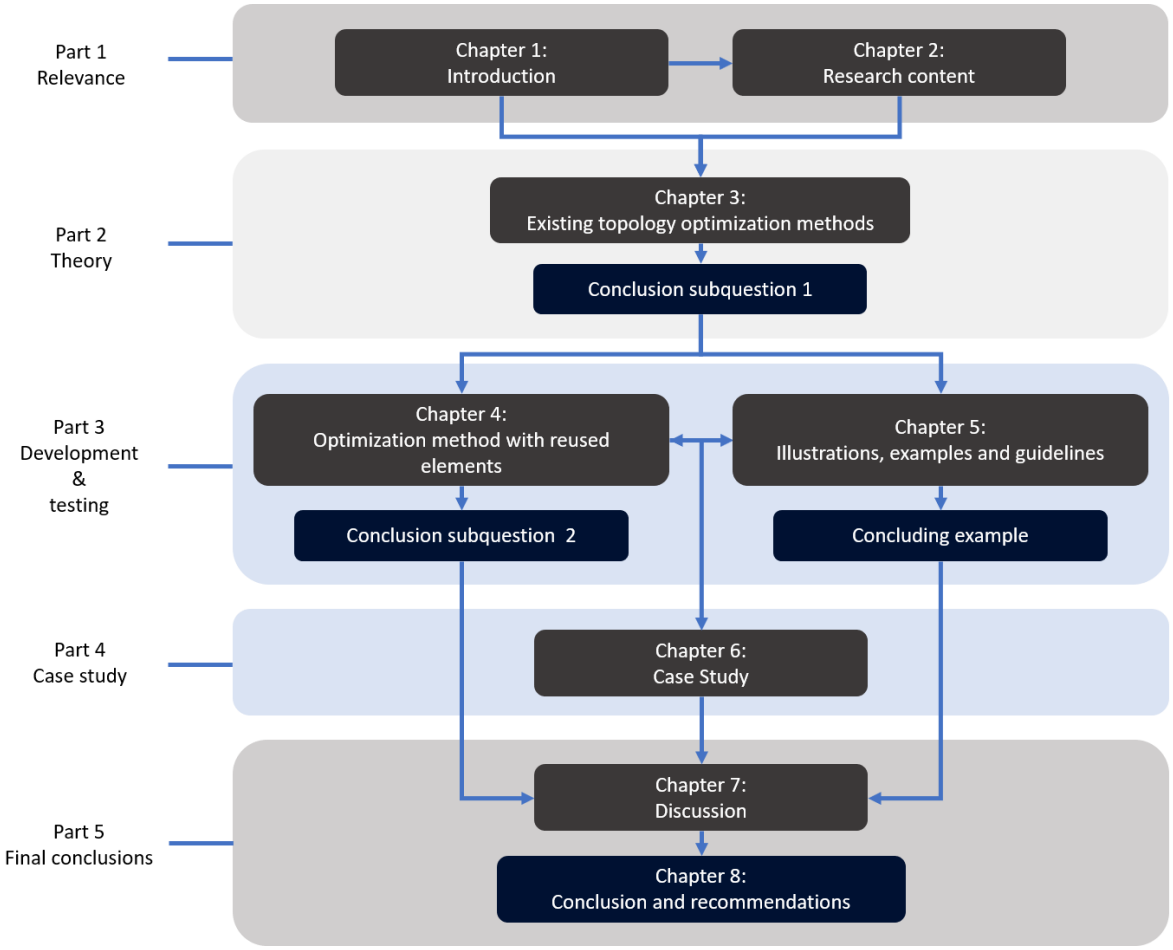


Figure 2.1: Structure of the report

3

Existing topology optimization methods

Topology optimization has been a field of interest for scientist for a few decades. The first topology optimization method was developed by Dorn (1964). Many studies in topology optimization have been carried out and different methods have been developed. Promising methods for performing an optimization with reused elements are the member adding scheme based on the ground structure method of He et al. (2019), the moving morphable component method with level set functions of Guo et al. (2014) and the continuum based topology optimization with discrete elements method with the density approach of Norato et al. (2015). A description of these methods is given in Section 3.1, Section 3.2 and Section 3.3 with advantages, drawbacks and required adaptations and additions to optimize with reused elements. The tool Peregrine from LimitState, a tool to perform topology optimisations in Grasshopper Rhino, is described in Section 3.4. An existing method which combines topology optimization with reused element is described in Section 3.5. A comparison of all methods is made in Section 3.6 after which is concluded that the member adding scheme of He et al. (2019) will be used for the development of the topology optimization method with reused elements in this paper.

3.1. Member adding scheme based on the ground structure method

3.1.1. Ground structure method

The ground structure method was the first topology optimization method developed by Dorn (1964). A design domain is specified and filled with nodes shown in Fig. 3.1(a) and Fig. 3.1(b). Connections are created from every node to every node resulting in a grid in Fig. 3.1(c) with all possible members. A load is applied and cross section of all members is optimized to find a solution with minimal volume in Fig. 3.1(d). Elements with a cross section close to zero are considered as non existing and are not visible in the final design. This method is an attractive method to find global optima since linear programming is used to formulate and solve the optimization problem. However, to acquire adequate results the optimization problem quickly becomes very large since a dense ground structure is required. A ground structure of n x n nodes results in $n^2 * (n^2 - 1)/2$ potential members, with n is 100 this results in 10.000 nodes and 49.995.000 potential members (Sokół, 2011).

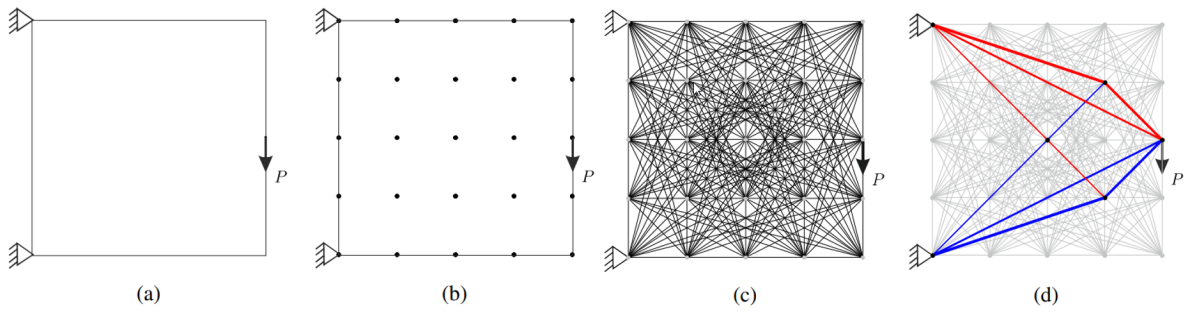


Figure 3.1: Ground Structure Method (He et al., 2019)

3.1.2. Member adding scheme

A more efficient method than the ground structure method was developed by Gilbert and Tyas (2003), the so called adaptive 'member adding scheme'. Reduced computation times can be observed compared to the ground structure method of Dorn (1964). In this method, a reduced ground structure (Fig. 3.3(a)) and 'possible member list' (PML) (Fig. 3.3(b)) is formed. The reduced ground structure is referred to as 'initial connectivity'. The possible member list contains all members not included in the initial connectivity (He et al., 2019). The optimization is an iterative process in which potential members from the PML are gradually added to the reduced set, an overview is provided in Fig. 3.2.

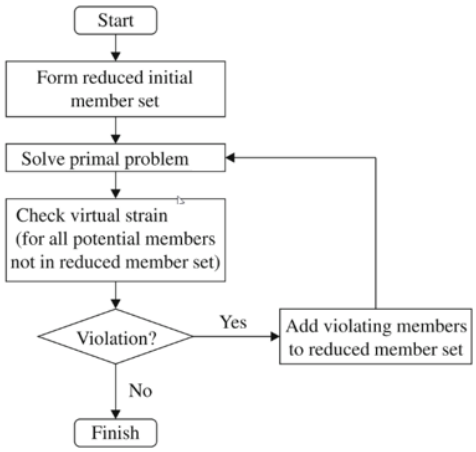


Figure 3.2: Member Adding Scheme Process (He et al., 2019)

In the first iteration, topology is optimized for the initial connectivity by solving Eq. (3.1). Volume is minimized by optimizing cross sections of all elements. Nodal displacements are calculated for all nodes in the design grid. With nodal displacements, virtual strain (ϵ) is calculated for all elements in the PML with Eq. (3.2). Members with the highest violation of virtual strain are added to the initial reduced set of members and Eq. (3.1) is solved again. When the number of violating bars is large, a small percentage of them are added and when the number of violating bars is small, a large percentage of them are added (He et al., 2019). Virtual displacement are calculated assuming that the displacements are infinitesimal (Fairclough and Gilbert, 2020). When no violation of virtual strain occurs any more in the PML, the optimization stops and the final design with minimal volume is presented to the user (He et al., 2019). A simple illustration of the described member adding process is provided in Fig. 3.3.

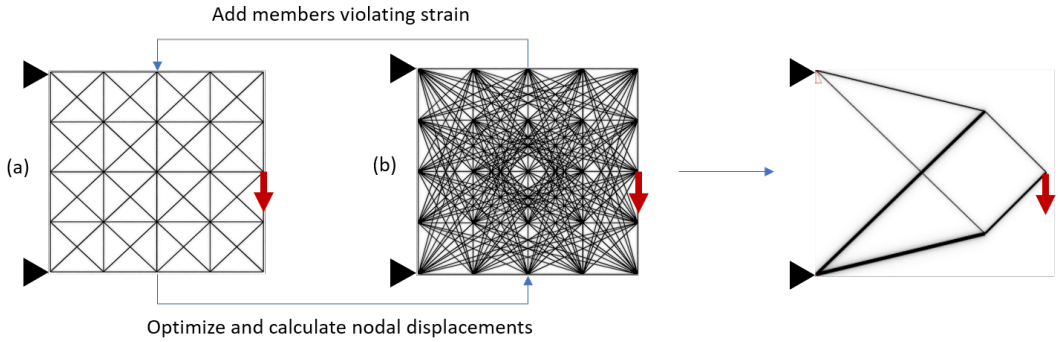


Figure 3.3: Member adding scheme: Initial connectivity (a), possible member list (b) and final design (c)

Every iteration, optimal topology is found by solving Eq. (3.1). Volume $\boldsymbol{V}$ is minimized, calculated by multiplying lengths $\boldsymbol{l}$ with cross sections $\boldsymbol{a}$. The optimization is subjected to three constraints. Equilibrium is established when equilibrium matrix $\boldsymbol{B}$ multiplied with internal loads $\boldsymbol{q}$ is equal to external loads $\boldsymbol{f}$. $\boldsymbol{k}$ Is the load case identifier with $\boldsymbol{p}$ the total number of load cases. Internal loads $\boldsymbol{q}$ must be smaller than the maximum tensile and compression stresses σ multiplied by the optimized cross sections $\boldsymbol{a}$. The number of nodes is denoted with $\boldsymbol{n}$ and the number of members with $\boldsymbol{m}$ (He et al., 2019).

$$\begin{aligned} &\min_{\mathbf{a},\mathbf{q}} V = \tilde{\mathbf{l}}^{\mathrm{T}}\mathbf{a} \\ &\text{s.t.} \\ &\left.\begin{aligned} &\mathbf{B}\mathbf{q}^k = \mathbf{f}^k \\ &\mathbf{q}^k \geq -\sigma^-\mathbf{a} \\ &\mathbf{q}^k \leq \sigma^+\mathbf{a} \\ &\mathbf{a} \geq \mathbf{0} \end{aligned}\right\} \text{for} \quad k = 1,2,\ldots,p \end{aligned} \tag{3.1}$$

Virtual strain ϵ is calculated with nodal displacements $\boldsymbol{u}$ with Eq. (3.2). For each member, maximum stresses σ are multiplied with equilibrium matrix $\boldsymbol{B}$ and virtual displacements $\boldsymbol{u}$ and divided by the length of the member. The sum of virtual strain for member i for all load cases must be smaller than 1.0 (He et al., 2019). According to the duality theory in the paper of He et al. (2019), the obtained solution will be equivalent to the solution obtained with the full problem with the ground structure method of Dorn (1964).

$$\epsilon_i = \sum_{k=1}^{p} \frac{\max\{\sigma^+\mathbf{B}_i^{\mathrm{T}}\mathbf{u}_i^k, -\sigma^-\mathbf{B}_i^{\mathrm{T}}\mathbf{u}_i^k\}}{\tilde{l}_i} \leq 1 \qquad (\text{for } i = 1,\ldots,m) \tag{3.2}$$

Dual Interior Point Method
Eq. (3.1) is solved with the dual interior point method. This is a class of algorithms to solve linear and non-linear convex optimization problems, problems with one global optimum. With this algorithm linear problems can be solved quit fast even for large problems (Boyd et al., 2004).

An illustration of the dual interior point method is provided in Fig. 3.4. A cantilever problem is given in a 1x1 grid with six nodes in the initial connectivity. Multiple solutions are possible, solution 1 as well as solution 2 are in equilibrium and capable of distributing the load towards the supports. With the dual interior point method these solutions are considered and a solution with minimal volume is found. Solution 1 contains 2 members with a larger cross section adding up to a total volume of 3.0, the other 4 members have a very small cross section not visible in the structure any more. Solution 1 has lower volume than solution 2 and is the global optimum.

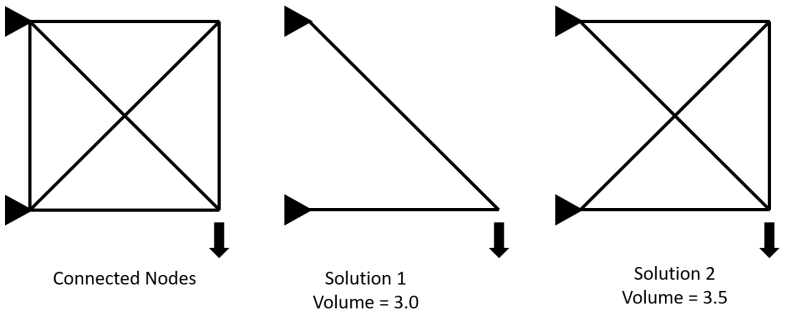


Figure 3.4: Dual interior point method, solutions for cantilever problem

In the python script optimization problem Eq. (3.1) is solved with an in built package cvxpy. This package solves the dual interior point method in just 12 lines of code.

Example
An example is given in Fig. 3.5 for a simple cantilever problem. The reduced ground structure is formed with members with a length smaller than 1.41 after which cross sections are optimized and members added until no violation of strain occurs any more. In Fig. 3.6, a joint cost penalty is added to the optimization. The joint cost penalty adds extra 'virtual length' to each member which penalizes the presence of any members, increasing volume share of each member. The result is that a structure is formed with less members with larger cross sections (He et al., 2019).

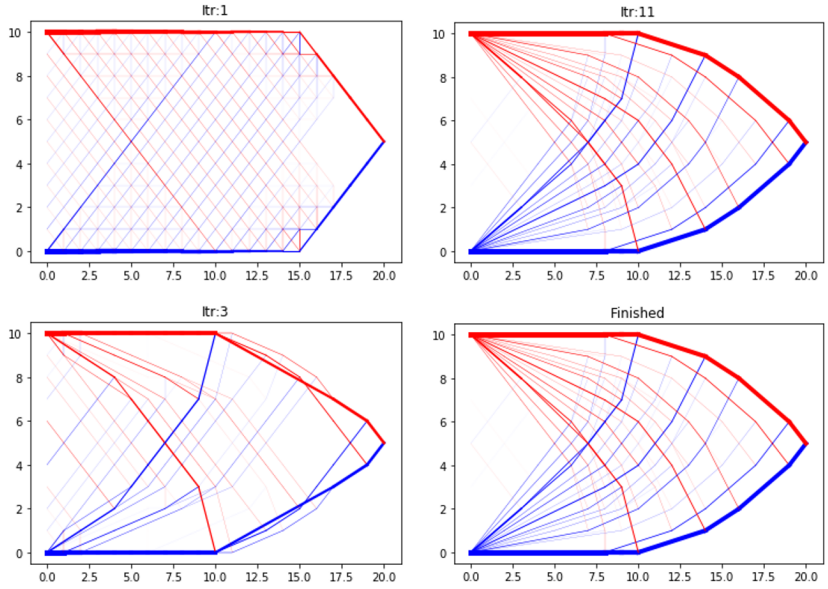


Figure 3.5: Example cantilever beam with member adding scheme (He et al., 2019)

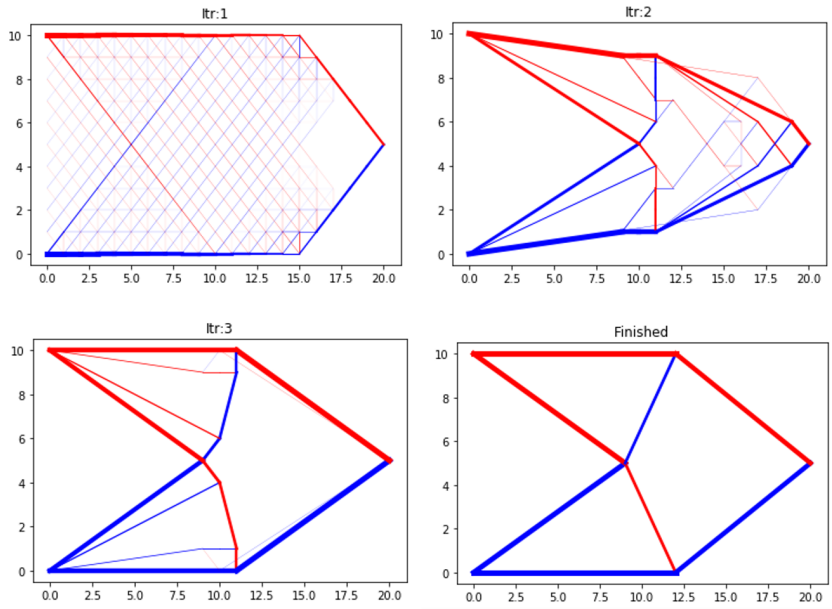


Figure 3.6: Example cantilever beam with member adding scheme and joint cost penalization (He et al., 2019)

3.1.3. Optimizing with reused elements

The member adding scheme based on the ground structure approach is a discrete method with understandable linear programming, increasing simplicity of implementing adaptations and additions. For small design problems only low computation times are required. A drawback is that little geometry information can be incorporated in the optimization, only length of possible members is specified after which cross sectional areas are optimized. Nodes are fixed which require reclaimed elements to have specific lengths to fit in the possible member list of Fig. 3.3(b). Important questions when optimizing with reused elements for this optimization method are:

- How will reused elements with a different length than the distance in between nodes in the ground structure be included in the optimization?
- How will geometry information about cross sectional area of reused elements be included?

- How will intersecting and overlapping elements be included in the structure? Cutting the elements results in multiple sections and requires much manual labour.

- How will the reduced initial ground structure be formed with reused elements? How many members with what length will be included in the initial connectivity?

3.2. Moving morphable component method (MMC Method)

With discrete node point based optimization methods like the ground structure method of Dorn (1964) and density based approach of Bendsøe (1989), geometry and mechanical information is not incorporated. The design domain is filled with material or lines representing members. The optimal solution is found by gradually deleting or removing material of members from the design space. In the density approach material properties are a function of the density. Pixels in the design domain can have a density between 0 and 1 representing no volume or volume. With solid isotropic material with penalization (SIMP), materials with a low density are penalized resulting in a solid-void design (Norato et al., 2015).
A new method, called the moving morphable component method (MMC method), was introduced by Guo et al. (2014). The design domain consists out of a set of components. These components are the building blocks to form the new structure, they are described by an x and y coordinate, a length, thickness and a rotation angle. The components are free to change their shape and position in the design domain by moving, rotating and shrinking or dilating. An important mechanism in this MMC method is that components can overlap with each other. In this way, components can be hidden behind other components making them disappear without removing them from the design domain (Guo et al., 2014).

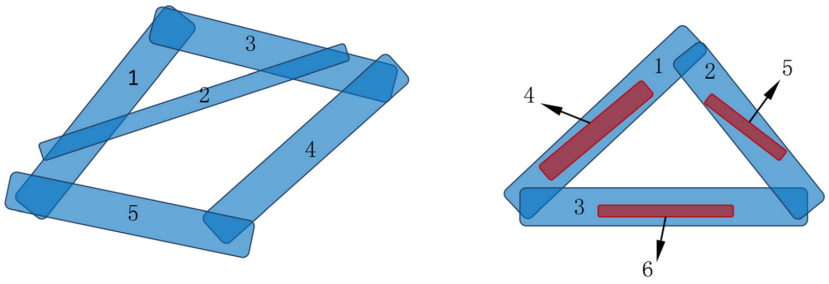


Figure 3.7: Overlapping components MMC method (Guo et al., 2014)

The main problem is formulated in Eq. (3.3) to minimize strain of the components under a volume constraint.

$$
\begin{aligned}
&\text{Find } \boldsymbol{d} = (\boldsymbol{d}_1, \ldots, \boldsymbol{d}_{\mathrm{nc}})^{\top} \\
&\text{Minimize } I = I(\boldsymbol{d}, \boldsymbol{u}) \\
&\text{s.t.} \\
&\int_{\Omega^s = U_{i=1}^{\mathrm{nc}} \Omega_i} \mathbb{E}(\boldsymbol{x}) : \boldsymbol{\varepsilon}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{v}) \mathrm{dV} \\
&\qquad = \int_{\Omega^s = U_{i=1}^{\mathrm{nc}} \Omega_i} f(\boldsymbol{x}) \cdot \boldsymbol{v} \mathrm{dV} + \int_{\Gamma_t} \boldsymbol{t} \cdot \boldsymbol{v} \mathrm{dS}, \quad \forall \boldsymbol{v} \in \mathscr{U}_{\mathrm{ad}} \\
&V(\boldsymbol{d}) \leq \bar{V} \\
&\quad \boldsymbol{d} \subset \mathscr{U}_{\boldsymbol{d}} \\
&\quad \boldsymbol{u} = \overline{\boldsymbol{u}}, \text{ on } \Gamma_u
\end{aligned}
\tag{3.3}
$$

Ω_i, i = 1 .. nc indicate the region occupied by the i-th component, the displacement field is given by $\boldsymbol{u}$ and $\boldsymbol{v}$. The body force density is given by $\boldsymbol{f}$ and $\boldsymbol{t}$, ϵ is the second order linear strain tensor (Guo et al., 2014).
The optimization problem is solved with a fixed finite element mesh on a design domain $\boldsymbol{D}$ and an Eulerian description. The boundaries of each component are described implicitly by an explicit level set function. An advantage of this method is that in the problem formulation only a limited number of members is used reducing the amount of design variables and computation times. Members can change their lengths and positions anywhere in the design domain and can form new elements by connecting several members to

each other, in this way a curved element can be created for example. With only a limited number of members in the design domain many solutions are possible (Guo et al., 2014).

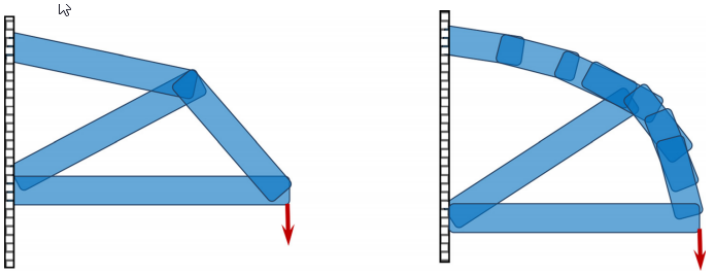


Figure 3.8: MMC Method: Connecting multiple components Guo et al. (2014)

An example of the MMC method is given for the beam and design domain in Fig. 3.9. The problem is modelled and optimized with symmetry, only half of the optimized structure is presented. In Fig. 3.10 intermediate iterations and the final design are presented.

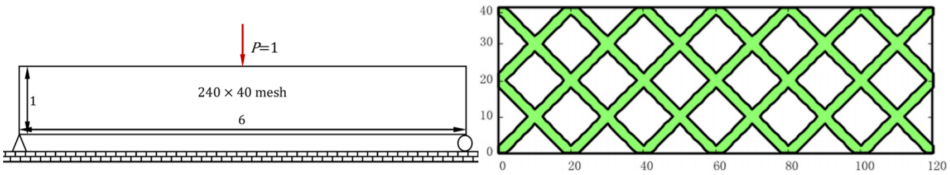


Figure 3.9: MMC Method: Design domain example simple beam (Guo et al., 2014)

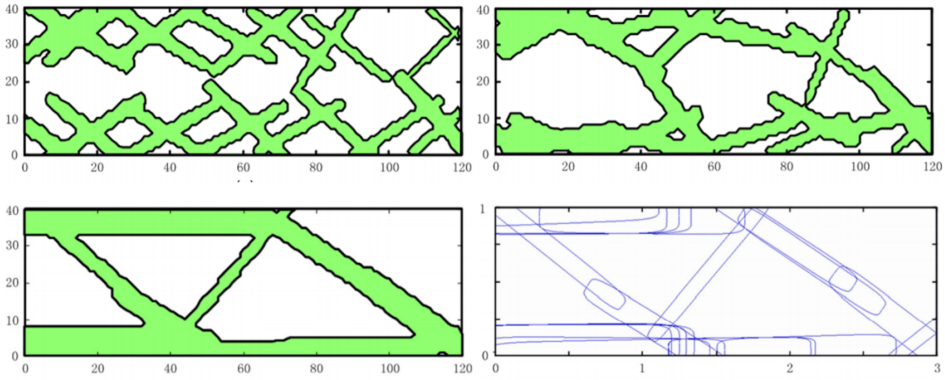


Figure 3.10: MMC Method: Iterations example simple beam (Guo et al., 2014)

3.2.1. Optimizing with reused elements

In this method geometry information about length and thickness can be incorporated in the design space, only few design parameters are required reducing computation times. Optimizing with reused elements requires that overlap and morphing of elements is no longer possible, dimensions of each component should be fixed. This could be a drawback since the strength and efficiency of this method is retrieved by these mechanisms. Important questions when using this method are:

- How will the optimization algorithm be adjusted such that length and thickness is fixed and can not be optimized?
- How will the design domain be created with reused components with different length and thickness?
- How will unnecessary components be removed from the design space if overlap is not allowed?
- Will this method still be powerful if overlapping and morphing of components is not allowed?

3.3. Continuum based topology optimization with discrete elements

Optimized designs with the moving morphable component method are structures with elements with complex shapes and rough boundaries. Additive manufacturing techniques could be used to fabricate these elements but in the building industry standard elements such as beams and pipes are preferred. Norato et al. (2015) introduced a continuum based topology optimization. The design space is formed by bars with semi-circular end which are described by the location of their endpoints and a thickness, bars can have different thickness's. Just as in density based topology optimization methods, the components are projected on the analysis grid to a continuously varying density field using a differentiable mapping from the design space. An adjoint sensitivity analysis is used and volume is minimized (Norato et al., 2015).

In this optimization method bars can overlap and merge, the method can be adjusted to allow only overlapping or only merging of the bars. When bars overlap, their thickness's are added and when bars merge, the effective thickness is the maximum thickness among all bars in the intersection. In both cases joints are stiff and can transfer bending moments. Therefore, the design can be seen as a frame structure (Norato et al., 2015).

Unlike in the MMC Method, bars can also disappear from the design space. Members can vary their length and thickness, when the length or thickness is close to zero, they are removed from the design space. Intermediate values of thickness are penalized with the SIMP method which results in a design with only bars of a thickness of 0 or 1 (Norato et al., 2015).

An example is given in Fig. 3.12 for the optimization problem in Fig. 3.11. Bars with a length or cross section close to zero are illustrated by bars with a grey colour (Norato et al., 2015).

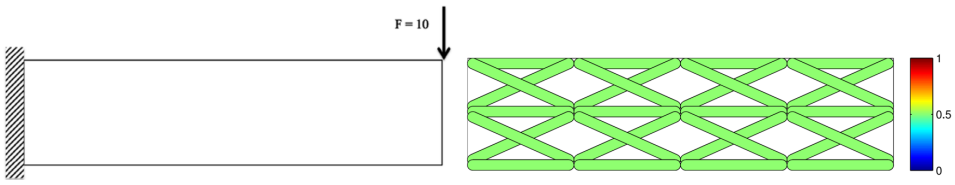


Figure 3.11: Continuum method with discrete elements: Design domain example simple beam (Norato et al., 2015)

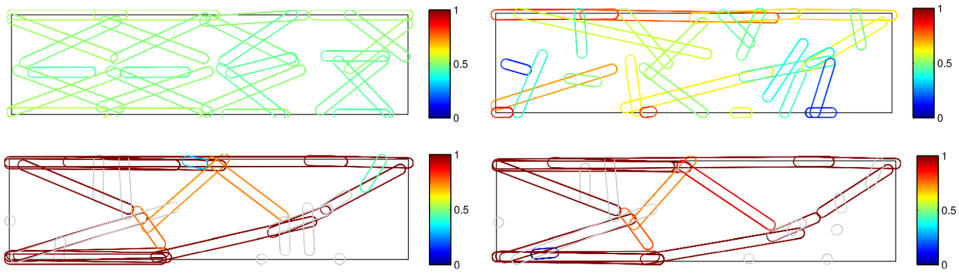


Figure 3.12: Design domain example simple beam: Iterations example simple beam (Norato et al., 2015)

3.3.1. Optimizing with reused elements

Geometry information about length and width can be incorporated in the design space in this optimization method. The optimal design is generated by optimizing thickness, length and layout of all components requiring low computation times. Component are allowed to overlap or merge with other components. Joints are stiff thus frame structures could be optimized. When this method will be used for optimizing with reused elements however, thickness and length should be fixed and overlap and merging of elements would not be allowed. Important questions concerning the required adaptations and additions for optimizing with reused elements are:

- How can the optimization algorithm be adjusted such that thickness and length of components is fixed and not variable?
- How will the design domain be created with reused components with different length and thickness?

- How will unnecessary components be removed from the design space if change of thickness and length is not allowed?

- Will this method still be powerful if overlapping and merging of components is not allowed?

3.4. LimitState Peregrine

Peregrine is an optimization tool in Grasshopper, a graphical algorithm editor. The tool is based on the LimitState FORM Technology. The optimization method used in this tool is the adaptive member adding scheme of He et al. (2019). A ground structure is formed after which a layout and geometry optimization is performed. Optional steps are to check elastic deflections and stresses (Gilbert and Shepherd, 2019).

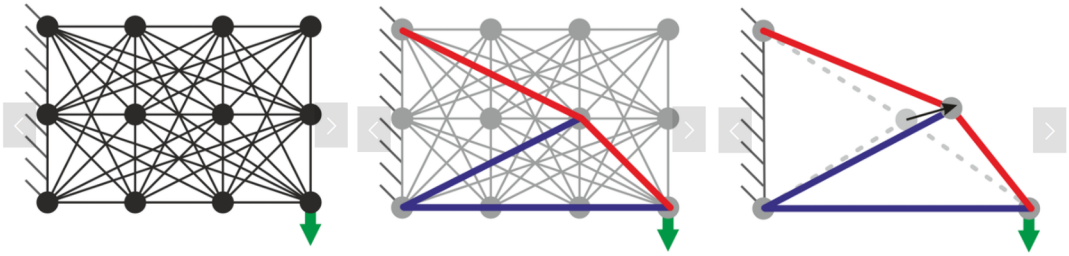
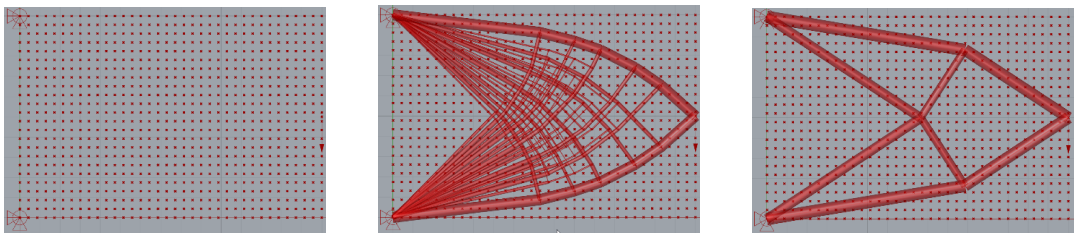


Figure 3.13: Member Adding Scheme LimitState FORM (Gilbert and Shepherd, 2019)

With this tool simple truss optimization problems can be performed with low computation times. First the design space is defined with the number of nodes, support reactions and loads shown in Fig. 3.14(a). Next the topology optimization is performed resulting in the structure in Fig. 3.14(b). The structure consists out of 170 bars with a total volume of 14.46. If this structure had to be manufactured, this would not be a convenient solution. To overcome this problem, the solution can be rationalized and simplified which is shown in Fig. 3.14(c). The structure now only consists out of 8 bars with a slight increase in volume (Gilbert and Shepherd, 2019).



(a) (b) (c)

Figure 3.14: Example cantilever problem in Grasshopper Peregrine

Unfortunately, the optimizations in Peregrine are not flawless. In the example shown in Fig. 3.15, at the bottom right an extra small bar is located. This increases the number of joints and members by one and could not be prevented by rationalizing, simplifying or joint cost penalty. Other methods to make more convenient structures in this tool include joint cost penalty, merging of nodes, creating cross over joints and setting a maximum to the number of members and number of members at a joint. These methods do not always work, their efficiency depends on the design problem. The bars in blue and orange indicate respectively compression and tension in the structure.

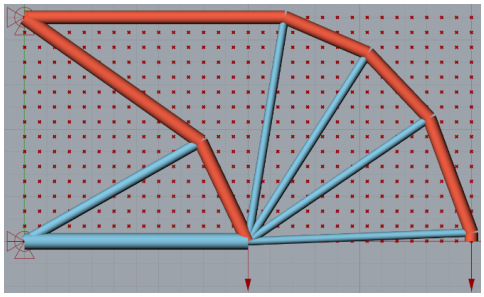


Figure 3.15: Example Cantilever beam with two loads Peregrine

3.5. Topology optimization with assignment of elements and geometry optimization

Brütting et al. (2019b) introduced a 3 step method for the design of truss structures with a stock of reused elements. First a ground structure is defined, after which a topology optimization is performed. Next, all members in the optimized structure are replaced by a set of available elements from stock. Elements assignment is optimized by using a simultaneous analysis and design approach (SAND). Structural mass and element capacity utilization are optimized resulting in optimal topology and assignment. The described problem is given in Eq. (3.4) which can be seen as a mixed-integer linear programming (MILP) problem (Brütting et al., 2019b).

In the third step of this approach a geometry optimization is performed. In this optimization nodes are free to move within a certain radius to match the member length to the assigned elements. This is a non-linear programming (NLP) resulting in local optima. Members and nodes can not be removed during this optimization (Brütting et al., 2019b). Brütting et al. (2020) extended the method of assigning elements from stock by cutting elements into multiple sections available for assignment to multiple members. In this way, elements from stock with a length much longer than all members can be added.

$$
\begin{aligned}
&\min_{T,p,u} M(T) = \bar{l}^T T(\boldsymbol{\alpha} \circ \rho) \\
&\boldsymbol{B}\boldsymbol{p}^{(k)} = \boldsymbol{f}^{(k)} + \boldsymbol{D}\boldsymbol{T}(\boldsymbol{a}\boldsymbol{a}\boldsymbol{y}) \quad \forall k \\
&\boldsymbol{b}_i^T \boldsymbol{u}^{(k)} \sum_{j=1}^{s} \frac{e_j a_j}{\bar{l}_i} t_{i,j} = p_i^{(k)} \quad \forall i,k \\
&-T(\boldsymbol{a} \circ \sigma) \leq \boldsymbol{p}^{(k)} \leq +\boldsymbol{T}(\boldsymbol{a} \circ \sigma) \quad \forall k \\
&-\sum_{j=1}^{s} t_{i,j} p_{i,j}^{buck} \leq p_i^{(k)} \quad \forall i,k \\
&\boldsymbol{u}_{\min}^{(k)} \leq \boldsymbol{u}^{(k)} \leq \boldsymbol{u}_{\max}^{(k)} \forall k \\
&\bar{l}_i \sum_{j=1}^{s} t_{i,j} \leq \sum_{j=1}^{s} t_{i,j} l_j \quad \forall i = 1 \dots m
\end{aligned}
\tag{3.4}
$$

An element buffer is introduced to allow elements from stock, that can not be assigned due to deviating lengths, to be assigned in later iterations if they do fit. Change of nodal positions changes the distance between nodes and length of members which might allow other elements to be assigned (Brütting et al., 2019b).

To compare emissions between structures made from reused and recycled elements, the total amount of embodied carbon and energy is calculated for both cases. An example of a simple cantilever is given in the figures below. In Fig. 3.16 the ground structure and available stocks of elements are given. In Fig. 3.17 the results of the optimization are shown and in Fig. 3.18 a comparison of the total amount of embodied energy and carbon is presented, comparisons are made for the solutions of Fig. 3.17 and two solutions made from new steel elements (Brütting et al., 2019b).

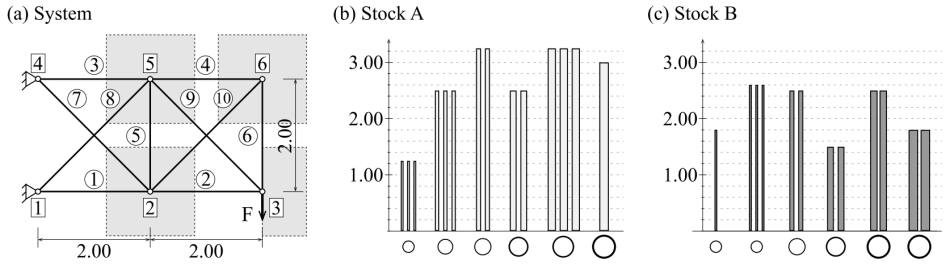


Figure 3.16: System (a), Stock A and B for simple cantilever example (Brütting et al., 2019b)

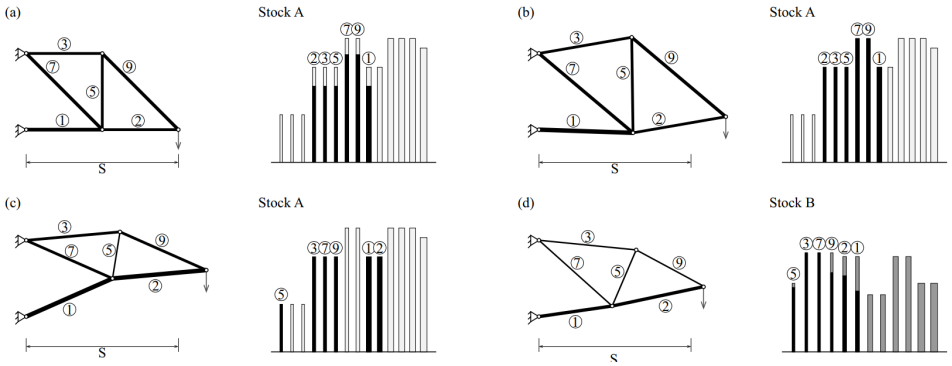


Figure 3.17: Cantilever results: (a) pure assignment and topology optimization, (b) layout optimization without element buffer, (c) and (d) layout optimization with element buffer (Brütting et al., 2019b)

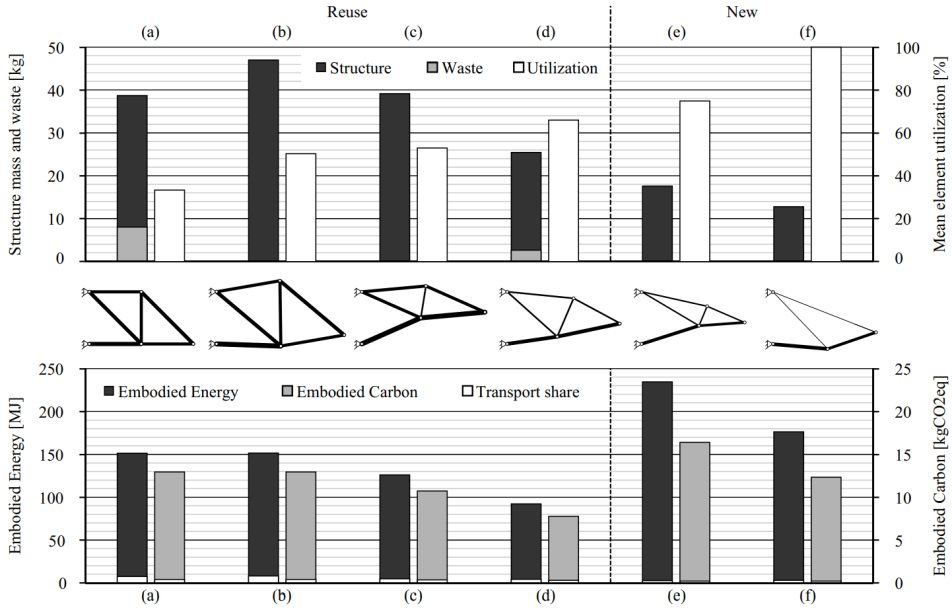


Figure 3.18: Comparison between structures: (a) to (d) made from reused elements, (e) and (f) made of new steel (Brütting et al., 2019b)

Case d results in the highest element utilization with reused elements. Compared to the benchmark solution (f) where the utilization is 100%, case d is still oversized. This will always occur if the stock of elements is limited. However, the embodied carbon and energy in case (d) are much lower than that in case (f) (Brütting et al., 2019b).

3.5.1. Limitations

This optimization approach illustrates a method to implement reused steel elements in the load bearing structure of a new building. However, the optimization could be more efficient. Assignment of elements to members in the optimized ground structure might result in local optima but will not result in global optima with respect to volume optimization. After assignment, members and nodes can not be removed any more, members can only be replacement by other elements during the geometry optimization (Brütting et al., 2019b). With elements available from stock with large cross section compared to the optimized cross section, designs will often be oversized.
An example of this problem is provided in Fig. 3.19. In Fig. 3.19(a), an optimization has been performed with the ground structure method. In Fig. 3.19(b) all members are replaced by stock elements. However, in this situation only stock elements are available with a cross section twice the cross section of the members in the optimized structure in Fig. 3.19(a). Volume of the structure has been doubled, the average capacity utilization is only 50%. A more efficient structure would have been a structure with less members shown in Fig. 3.19(c).

A drawback of this method is that the efficiency of the optimization heavily depends on the established ground structure, available elements and loading conditions. In the examples shown in Fig. 3.16, the nodal distance in the design grid for the ground structure is adapted to the length of available elements from stock to make element assigned more easy. However, with a denser mesh with more nodes, solutions closer to benchmark solutions could have be found. The efficiency and thus capacity utilization of the assigned elements depends on their cross sectional area and loading conditions. When only a small load is applied and the average cross sectional area of available elements is large, solutions arising from the topology optimization will be overdimensioned.

This method could be adapted and improved by simultaneous optimization of topology, element assignment and geometry optimization. The script of this method is not available.

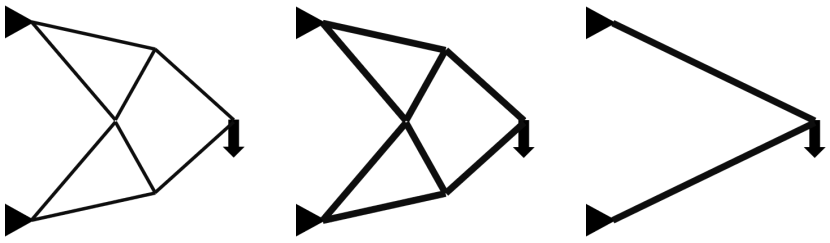


Figure 3.19: Optimization ground structure method (a), oversized reused element assignment (b), alternative more efficient design with reused elements (c)

3.6. Comparison of existing topology optimization methods

The existing topology optimization methods and topology optimization method combined with assignment of elements described in Chapter 3 have promising aspects and drawbacks, an overview is given in Table 3.1. The member adding scheme of He et al. (2019) is a suitable method since this method makes use of discrete elements which is required when optimizing with reused elements, all members in the possible member list (PML) have fixed length. The PML could easily be adjusted to a list with members only with length available from stock. Adjusting the problem formulation to include geometry information about available cross sections is a challenge but could be combined with the method of element assignment of Brütting et al. (2019b). The method of Brütting et al. (2019b) is an interesting method of designing with reused elements, however, with the two step optimization approach efficiency of the design heavily depends on the ratio of optimized cross sections and available cross sections from stock, designs are often oversized. The python script with linear programming of the member adding scheme explained in the paper of He et al. (2019) is readily available for adaptations and additions.

An advantage of the moving morphable component (MMC) method of Guo et al. (2014) and continuum method with discrete elements (CMDE) of Norato et al. (2015) is that geometry information about components can be included. Fixed width of elements can be specified in the method of Norato et al. (2015). Both

methods require simplification of the optimization process. Morphing, overlapping and merging of components must be disabled which are the most important optimization mechanisms of these methods. It is uncertain whether the optimization methods would still be powerful if these features are disabled. Issues arising in both methods are that elements can not be removed from the design space when overlapping, merging and change of thickness and length is not allowed. Solutions are required to make elements in the MMC method and CMDE method more discrete in terms of cross section and/or length. The MMC method and CMDE method have lower computation times than the member adding scheme of He et al. (2019) despite the reduction in computation times relative to the ground structure method.
However, the advantage of discrete elements and linear programming outweigh the advantages of the MMC method and CMDE method of including geometry information and low computation times. The MMC method and CMDE method bring much uncertainty with regard to optimization strategies and possibilities of removing elements from the design domain. Therefore, the member adding scheme of He et al. (2019) is used as a basis for the optimization method with reused elements developed in this paper.

Method	Member Adding Scheme	MMC Method	Continuum with Discrete Elements	Element Assignment
Promising Aspects	• Discrete Elements • Linear Programming • Python script available with explanation. • Reduced computation times	• Geometry information about length and thickness Included • Low computation times	• (Fixed) Geometry information about length, width and thickness Included • Low computation times	• Implementation of reused elements
Required Adjustments	• PML with reused elements • Fixed cross sectional area • Availability constraint • Exclude intersecting and overlapping members	• Reused components • No overlap, morphing, dilatating and shrinking of components • Fixed shape, thickness and length of components	• Reused components • No overlap and merging of components • Fixed thickness and length of components	• Simultaneous topology optimization and element assignment
Problems and Limitations	• Only elements with a specific length fit in the ground structure • Structurally stable ground structure required • High computation times for large problems • No geometry information about size included	• Strength of method depends on overlap and morphing • Components can not be removed from design • No explanation of the script available	• Strength of method depends on overlap and merging • Components can not be removed from the design • No script available	• Local Optima and oversized structures • Members can not be removed from the design • No script available

Table 3.1: Overview existing topology optimization methods

4

Member adding scheme with reused elements

A method combining topology optimization with designing with reused elements has been developed. The member adding scheme of He et al. (2019) has been used as a basis, adaptations and additions have been made to develop a tool able to design and optimize steel trusses with reused elements. The main goal of the optimization is to minimize volume, maximize capacity utilization of reused elements and maximize the percentage of reused elements versus new elements in the optimized designs.
The optimization consists a number of iterating steps, an overview of the process is given in Fig. 4.1. First, topology is optimized with the optimization algorithm of the member adding scheme of He et al. (2019). Next, members in the optimized structure are replaced by elements from stock as efficient as possible, similar to the method described in Section 3.5 of Brütting et al. (2019b). Members which could not be replaced by elements from stock are new elements. In the third step, the structure is recalculated for displacements with the assigned cross sections. In the fourth step, designs are verified for requirements for virtual strain, maximum unity check and minimum efficiency and reuse percentage. Inefficient and new elements receive penalty's in step 5 avoiding them to be present in the next and/or further iterations. The member adding procedure is continued in step 6. Optimized designs are presented in step 7 when all requirements are met.

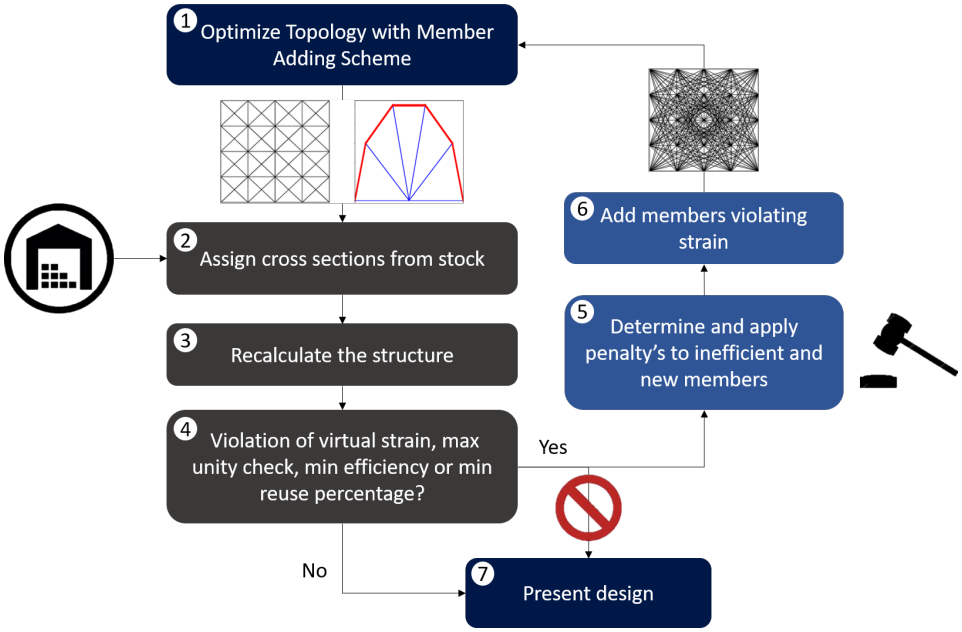


Figure 4.1: Optimization process Member Adding Scheme with reused elements

The final optimization tool contains many parameters which can be varied during the optimization. All parameters have their own influence on the optimization. It seems impossible to determine one single combi-

nation of all input parameters resulting in the best designs. Besides, designating the 'best' design is subjective, designs with less members or a higher percentage of reuse can be more convenient than the design with least volume. Architectural interest can also determine which design will be preferred.
Therefore, multiple designs will be generated with all possible combinations of input parameters, an overview is given in Fig. 4.2. Definition of all parameters will be given in this chapter. Designs with least volume, least number of elements, highest average unity check and highest percentage of reuse will be compared and presented.

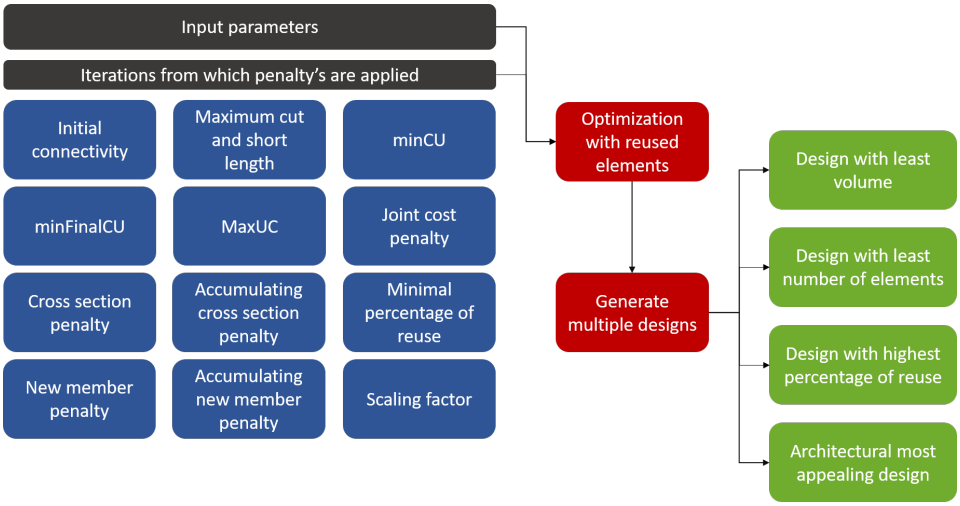


Figure 4.2: Optimization strategy for generating multiple designs with reused elements

A workflow diagram is given in Fig. 4.3 involving all the required adaptations and additions to the member adding scheme of He et al. (2019) to design a steel truss with reused elements. In this report the first three most important tasks are executed. The possible member list (PML) is adapted to a list where only elements with lengths available from stock are included, referred to as possible stock member list (PSML). Additions to the problem formulation have been made, cross sections available for the corresponding available lengths are assigned to the optimized members and finally availability constraints are implemented.

Modifications that have not but should and could be made are excluding intersecting and overlapping members and including constraints on buckling, nodal instability and deflection limits. In literature, research into these modifications has already been done. Constraints on intersecting members have been implemented in the papers of Ohsaki and Katoh (2005) and Zegard and Paulino (2014). Cui et al. (2018) excluded overlapping as well as intersecting members and Fairclough and Gilbert (2020) created constraints on the number of joints and cross over joints, thus the number of intersecting members. Examples of the implementation of buckling can be found in the articles of Zhao (2015) and Cui et al. (2018), Zhao (2015) also included nodal instability just as Ohsaki and Katoh (2005) and Fairclough and Gilbert (2020).

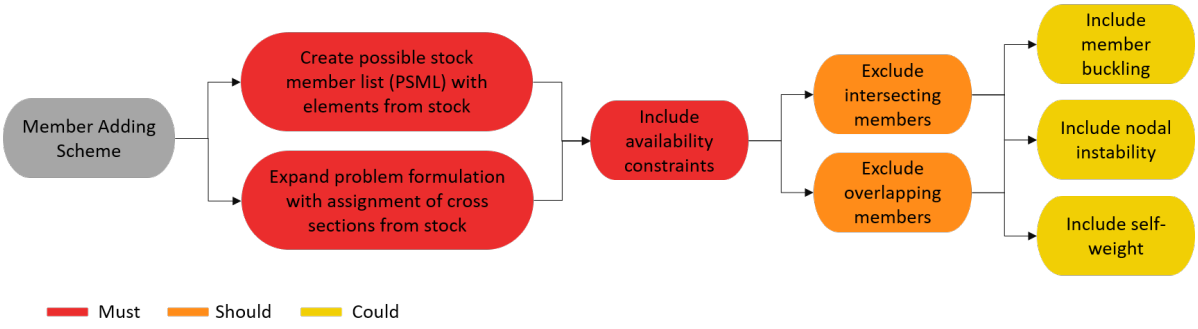


Figure 4.3: Workflow diagram modifications to member adding scheme of He et al. (2019)

4.1. Define the optimization problem

The optimization of a 2D steel truss with reclaimed elements starts with defining the stock of reclaimed elements, grid, and boundary- and loading conditions. Grids are specified by a width and length with a nodal division of 1.0, a grid of 10 x 5 contains 50 nodes. Boundary conditions are defined by determining the degrees of freedom for each node in x and y direction, loading conditions are specified for each node as well. Elements from stock are described by a length ***L***, cross section ***a***, index number ***i*** and number of availability ***n***, elements with similar length but different cross section have different index numbers. Indices are used for programming in python, with index numbers properties of each member can be obtained.

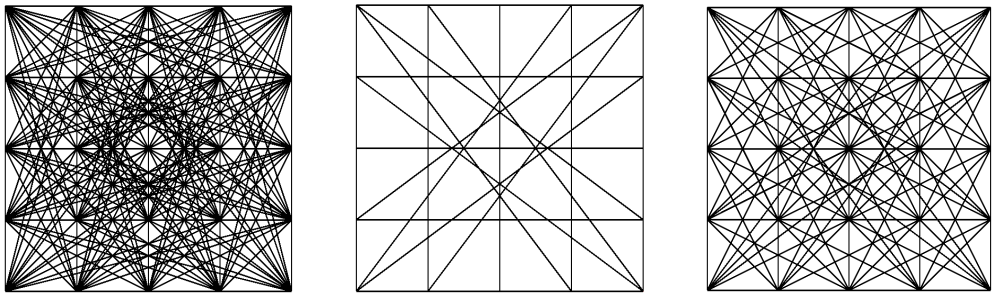
$$Member = [L, a, i, n] \tag{4.1}$$

4.2. Possible stock member list

The possible stock member list (PSML) is created by filtering the possible member list (PML) with available lengths from stock. The length of a member in the PML is determined by the distance in between the nodes that are connected. Members in the PML with a length similar to the length of an element available from stock are included in the PSML. It might occur that the length of an elements available from stock is just shorter or longer than a member in the PML. Therefore, $Ł_{cut}$ and $Ł_{short}$ are introduced which are the maximum length a member from stock can be cut and the maximum length a member from stock may be short to fit in the PML. Members from the PML within the range of $Ł_{Stock}$–L_{short} to $Ł_{Stock} + L_{cut}$ are added to the PSML described by Eq. (4.2).

$$\begin{gathered}\text{Add member from PML to PSML if:} \\ L_{\text{stock}} - L_{\text{short}} \leq L_{\text{PML}} \leq L_{\text{stock}} + L_{\text{cut}}\end{gathered} \tag{4.2}$$

The number of available lengths and the value of $Ł_{cut}$ have a huge influence on the optimization. More available lengths and a higher value of $Ł_{cut}$ increase the number of members in the PSML and thus the number of solutions. In case only few members are in the PSML and no convenient designs can be made, $Ł_{cut}$ should be increased. An example is given in Section 4.2.1, available lengths are 1.50m, 2.30m, 4.0m and 5.0m.



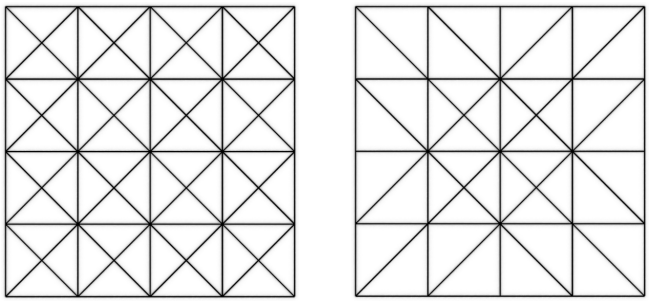
(a) (b) (c)

Figure 4.4: Example PML (a), PSML (b) and PSML with $Ł_{short}$ and $Ł_{cut}$ of 0.10m (c)

4.2.1. Initial connectivity

A challenge with implementing the PSML in python is to determine the initial connectivity. In the original member adding scheme the initial connected nodes are only members with a length of $L < 1.42$ which are all nodes connected to the nodes closest to them shown in Fig. 4.5(a). A requirement for the initial connectivity is that a stable structure must be formed. In the original member adding scheme and with the only adaptation of the possible stock member list the initial connectivity has no influence on the final result, only on computation times.

When lengths of 1m and 1.41m (1x1m) are not available, the initial connectivity of the original member adding scheme can not be used. If elements with a length of 3m and 4.24 (3x3) are available for example, the initial connectivity could be created with members with $L < 4.25m$ shown in Fig. 4.5(b)



(a) (b)

Figure 4.5: Example original (L < 1.41m) (a) and adapted (L < 4.25m) (b) initial connectivity

4.2.2. Scaling

The number of members in the PSML, determined by the number of nodes and available lengths from stock, determine the number of possible solutions and thus the possibilities for an efficient design. When no convenient or efficient results are found with a certain set of available lengths, $Ł_{cut}$ could be increased or a scaling factor could be applied. With scaling, the grid size, available lengths, $Ł_{short}$ and $Ł_{cut}$ are multiplied by a scaling factor increasing the number of nodes and thereby number of members in the possible stock member list. The same effect as increasing the nodal density of the grid is achieved. After the optimization is performed the design can be downscaled again by dividing the grid and length of members by the scaling factor.
Optimization problems with large grids can also be downscaled decreasing the density of the grid, which could be an effective method to reduce complexity and computation times. A scaling factor of 0.5 could be applied for example.

4.3. Adapted problem formulation: Capacity utilization

Designing with reclaimed steel components is complex, each component has an individual length and cross section. The properties of each component should be implemented in the optimization. However, the optimization algorithm of the dual interior point method, described in Section 3.1, can not be extended by constraints for the optimization of cross sections with a list of cross sections available from a database. Constraints must be defined with linear or non-linear formulas (Boyd et al., 2004). Only one constraint for a minimum and maximum cross section can be set, for which the minimum cross section is zero. Therefore, an alternative method is used to implement cross sections available from stock. Topology and cross sections are optimized by solving Eq. (3.1) with the dual interior point method minimizing volume, after which cross sections available from stock are assigned to the optimized cross sections as efficient as possible, capacity utilization of the members is optimized. Every iteration cross sections are assigned again. This approach is based on the method of designing with reused elements of Brütting et al. (2019b) described in Section 3.5, where cross sections were assigned only after the final iteration.

Optimized cross sections are replaced by available cross sections from stock as efficient as possible with Eq. (4.3). For every member the unity check for all possible cross section is calculated. The capacity utilization factor (CUf) is calculated for unity checks lower or higher than 1.0 and the cross section with the lowest capacity utilization factor, thus most efficient, is assigned to the member. If a cross section of $1e-3*maxArea$, with $maxArea$ the largest available cross section, results in the lowest CUf, no cross section from stock will be assigned and the member will retain its optimized cross section, the member is regarded as a new member. Members with a cross section equal or lower than $1e-3*maxArea$ are considered as non-existing, not contributing to the strength of the structure and not visible in the design.

A maximum unity check is defined, if the unity check for assignment of a certain cross section is larger than the maximum unity check, 100 will be added to the capacity utilization factor (CUf) preventing assignment of this cross section. In case no cross sections resulting in a unity check lower than the maximum unity check are available, no cross section will be assigned. The member retains its optimized cross section and is regarded as a new member. A simple example is given in Table 4.1, CUf are calculated for all possible cross sections, the cross section with lowest CUf will be assigned.

In the final design no violation of maximum tension and compression stresses is allowed, unity check of all members should be lower than or equal to 1.0. However, a maximum unity check higher than 1.0 could have a positive influence on the efficiency of the final design. Assignment of cross sections in intermediate iterations could be done more efficiently, a cross section with a unity check of 1.1 versus a cross section with a unit check of 0.1 for example. Besides, a higher maximum unity check could lead to variety of designs due to variety of assignment of cross sections. In the final design however, highest unity check must be lower than 1.0 for all situations.

$$\begin{aligned} &\min_{\mathbf{a},\mathbf{q}} \ \mathbf{C}.\mathbf{U}.\mathbf{f}. \\ &\text{s.t.} \\ &\left.\begin{aligned} &\mathbf{UC} = max(\tfrac{\mathbf{q^k}}{-\sigma^- * a}, \tfrac{\mathbf{q^k}}{\sigma^+ * a}) \\ &\quad if\ \mathbf{UC} > 1.0 \\ &\qquad \mathbf{CUf} = \tfrac{\mathbf{UC}}{1.0} \\ &\quad elif\ \mathbf{UC} > maxUC \\ &\qquad \mathbf{CUf} = 100 + UC \\ &\quad else \\ &\qquad \mathbf{CUf} = \tfrac{1.0}{\mathbf{UC}} \end{aligned}\right\} \text{for} \quad k = 1,2,\ldots,p \end{aligned} \tag{4.3}$$

Internal load (q) = 1.0, maximum stress = 1.0, maximum UC = 1.0		
Cross section (a)	UC	CUf
1.1	0.91	1.1
0.8	1.25	101.25
1.5	0.667	1.5

Table 4.1: Example: Assigning cross section with capacity utilization

4.4. Recalculating the structure: virtual displacements and virtual strain

In the original member adding scheme problem Eq. (3.1) is solved in python with a powerful solver from cvxpy. the output of this solver are the internal forces $\boldsymbol{q}$, cross sections $\boldsymbol{a}$ and virtual displacements $\boldsymbol{u}$. With virtual displacements virtual strain is calculated and if violation of virtual strain occurs in the PML, members with the highest violation are added to the structure. In the new optimization method cross sections from stock are assigned after which virtual displacements are recalculated and the structure reverified for virtual strain, a workflow diagram is given in Fig. 4.6. Methods to calculate virtual displacements are described in Section 4.4.1 and Section 4.4.2.

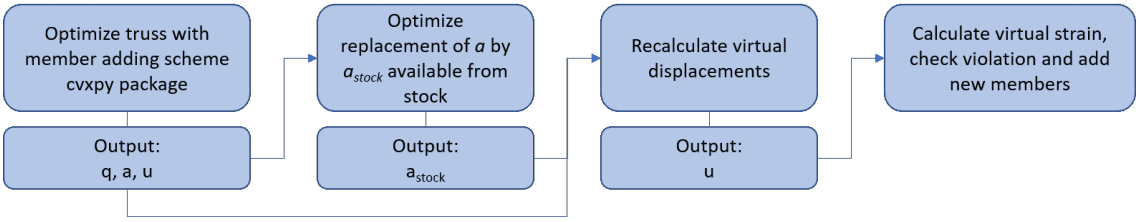


Figure 4.6: Recalculating virtual displacements

4.4.1. Virtual work

Internal forces and cross sections in the trusses are determined based on equilibrium and maximum stresses. Virtual displacements, meaning displacements determined with virtual work are calculated with Eq. (4.4) (Welleman, 2016).

$\boldsymbol{P}$ is the virtual force in the direction of the expected displacement with a value of 1.0. Δ is the displacement, $\boldsymbol{n}$ the internal forces due to $\boldsymbol{P}$, $\boldsymbol{N}$ the internal forces due to the load $\boldsymbol{f}$, $\boldsymbol{L}$ the length of members, $\boldsymbol{A}$ the cross section of members and $\boldsymbol{E}$ the modulus of elasticity.

$$P * \Delta = \sum \frac{n * N * L}{A * E} \tag{4.4}$$

An example is given for the cantilever problem in Fig. 4.7 and Fig. 4.8 with a point load $\boldsymbol{f}$ of 1.0. The vertical and horizontal displacements in B are determined. The modulus of elasticity is 1.0. First, the internal forces due to $\boldsymbol{f}$ are calculated. Next, a virtual load $\boldsymbol{P}$ of 1.0 is applied and internal forces for the virtual load are calculated. The maximum allowed stress in each member is 1.0 resulting in cross sections equal to the absolute value of the internal forces. All the unknowns are known and Δ can be calculated.

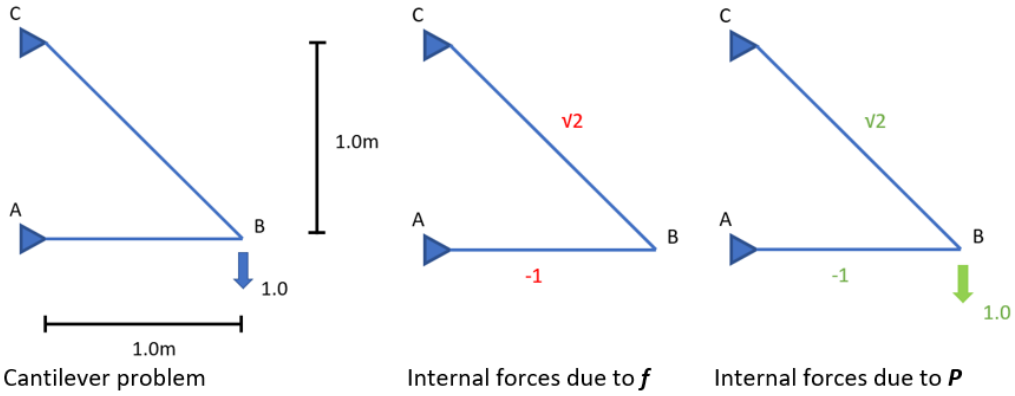


Figure 4.7: Example: load case 1, Virtual work and vertical displacement in B

Internal forces due to $\boldsymbol{f}$ and $\boldsymbol{P}$:

Member	N	n	L	NnL
AB	-1	-1	1m	1
BC	√2	√2	√2m	2√2

$$1.0 * \Delta = \frac{1}{1} + \frac{2 * \sqrt{2}}{\sqrt{2}} = 3.0 \tag{4.5}$$

The horizontal displacement in B is calculated with a horizontal virtual load $\boldsymbol{P}$, the internal forces are:

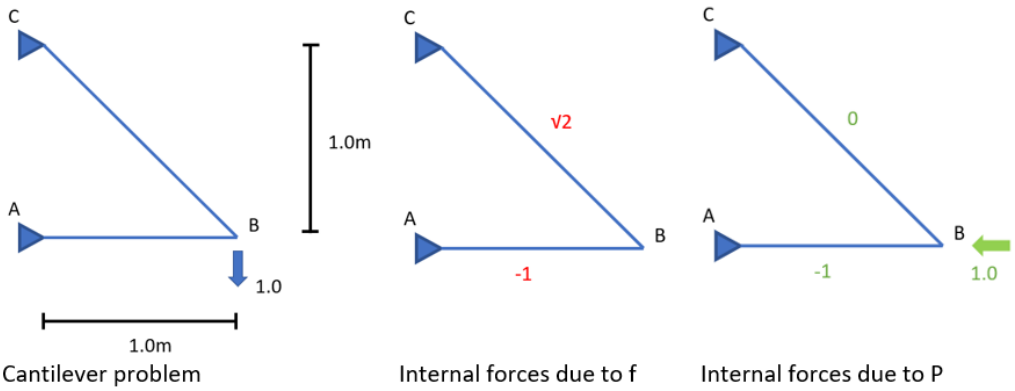


Figure 4.8: Example: load case 1, Virtual work and horizontal displacement in B

Internal forces due to $\boldsymbol{f}$ and $\boldsymbol{P}$:

Member	N	n	L	NnL
AB	-1	-1	1m	1
BC	√2	0	√2m	0

$$1.0 * \Delta = \frac{1}{1} = 1.0 \tag{4.6}$$

Equation (4.5) and (4.6) result in a vertical and horizontal displacement Δ in B of respectively 3.0 and 1.0 which correspond to the values obtained in the python script.

4.4.2. Matrix method

Virtual displacements must be calculated for every possible statically determinate and indeterminate truss. In the adapted python script this is done with the matrix method. In this method displacements $\boldsymbol{u}$ are calculated with stiffness matrix $\boldsymbol{K}$ and force vector $\boldsymbol{f}$ with Eq. (4.7) (Simone, 2007). A demonstration for this method is given for the same cantilever problem used in the example of virtual work in Section 4.4.1.

$$K * u = f \tag{4.7}$$

The cantilever in Fig. 4.9 is considered, all nodes and elements have been numbered, each element has a node $\boldsymbol{i}$ and node $\boldsymbol{j}$ with x and y coordinates. In this two dimensional example all nodes are hinges, therefore all nodes have two degrees of freedom shown in red.

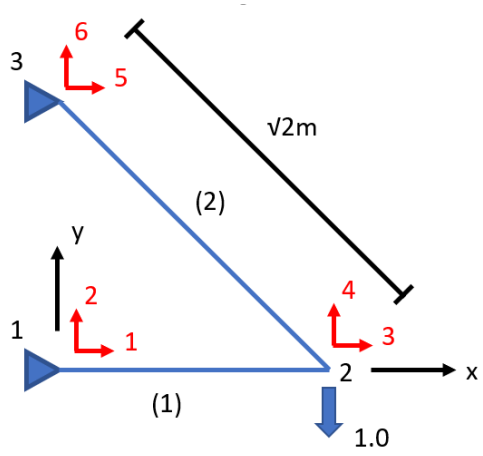


Figure 4.9: Example load case 1: Matrix Method

The stiffness matrix of each element is determined with the coordinates of the connected nodes and length, cross section and modulus of elasticity of the element.

$$K^{(e)} = \frac{AE}{L}\begin{bmatrix} \lambda_x^2 & \lambda_x\lambda_y & -\lambda_x^2 & -\lambda_x\lambda_y \\ \lambda_x\lambda_y & \lambda_y^2 & -\lambda_x\lambda_y & -\lambda_y^2 \\ -\lambda_x^2 & -\lambda_x\lambda_y & \lambda_x^2 & \lambda_x\lambda_y \\ -\lambda_x\lambda_y & -\lambda_y^2 & \lambda_x\lambda_y & \lambda_y^2 \end{bmatrix}$$

For the cantilever problem this result in the following values of λ in tabel Table 4.2:

Member	Node i	Node j	x_j	x_i	y_j	y_i	λ_x	λ_y
1	1	2	1	0	0	0	1	0
2	2	3	0	1	1	0	$-1/\sqrt{2}$	$1/\sqrt{2}$

Table 4.2: Values of λ

With maximum tensile and compression stresses of 1.0, cross sections are equal to the absolute value of the internal forces. The modulus of elasticity is equal to 1.0. This results in stiffness matrix $K^{(1)}$ and $K^{(2)}$ with corresponding degree of freedom.

$$K^{(1)} = \frac{1.0}{1.0}\begin{bmatrix} 1.0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1.0 & 0 & 1.0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} \qquad K^{(2)} = \frac{\sqrt{2}}{\sqrt{2}}\begin{bmatrix} 0.5 & -0.5 & -0.5 & 0.5 \\ -0.5 & 0.5 & 0.5 & -0.5 \\ -0.5 & 0.5 & 0.5 & 0.5 \\ 0.5 & -0.5 & -0.5 & 0.5 \end{bmatrix} \begin{matrix} 3 \\ 4 \\ 5 \\ 6 \end{matrix}$$

With stifness matrix $K^{(1)}$ and $K^{(2)}$ the complete stiffness matrix $\boldsymbol{K}$, displacement vector $\boldsymbol{u}$ and force vecotor $\boldsymbol{f}$ can be assembled. The displacements $\boldsymbol{u}$ correspond to the degree of freedom of each node.

$$K = \begin{matrix} \\ 1.0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \end{matrix} \begin{matrix} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 \end{matrix} \\ \begin{bmatrix} 0 & -1.0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1.5 & -0.5 & -0.5 & 0.5 \\ 0 & -0.5 & 0.5 & 0.5 & -0.5 \\ 0 & -0.5 & 0.5 & 0.5 & 0.5 \\ 0 & 0.5 & -0.5 & -0.5 & 0.5 \end{bmatrix} \end{matrix} \begin{matrix} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{matrix} \qquad u = \begin{bmatrix} u1 \\ u2 \\ u3 \\ u4 \\ u5 \\ u6 \end{bmatrix} \qquad f = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1.0 \\ 0 \\ 0 \end{bmatrix}$$

Taking the boundary conditions into account, only displacement ***u3*** and ***u4*** are non zero. Row reducing matrix ***K***, ***u*** and ***f*** results in:

$$\begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.5 \end{bmatrix} * \begin{bmatrix} u3 \\ u4 \end{bmatrix} = \begin{bmatrix} 0 \\ -1.0 \end{bmatrix}$$

The displacement ***u3*** and ***u4*** can be solved by multiplying the inverse stiffness matrix K^{-1} with force vector ***f***.

$$u = K^{-1} * f$$

This results in:

$$u3 = -1.0 \qquad u4 = -3.0$$

The displacements correspond with the displacements calculated with virtual work.

4.5. Influencing optimization of cross sections

Every iteration, cross sections from stock are assigned to the optimized cross sections as efficient as possible. The structure is recalculated and new members could be added to the structure. However, no influence is exerted on the optimization of topology and cross sections which could result in oversized structures with large deviations between assigned and optimized cross sections. The only constraint for the optimization is that cross sections should be larger than zero. Since it is not possible to write constraints for cross sections to be equal to cross sections from a certain database, three methods are presented in this section to influence the optimization of cross sections.

4.5.1. Joint cost

Joint cost, included in the original member adding scheme, penalizes the presence of every member by increasing the length of all members, increasing volume share, which forces the optimization to create designs with less members but with larger cross sections. However, the number of members required in a structure depends on the load and available cross sections. With a high load and small cross sections many members should be present and with a small load and large cross sections only few members. Whether the joint cost penalty should be used depends on the design problem.

4.5.2. Maximum cross section

In addition to a minimum cross section for all members in the optimization, a maximum cross section can be defined. This can avoid members being present in the design with cross sections larger than available from stock or could force the optimization to use smaller cross sections, resulting in a design with more members.

4.5.3. Cross section penalty

A cross section penalty (csp) is introduced to penalize inefficient members with a large deviation between optimized and assigned cross section. Every iteration penalty's are determined with Eq. (4.8). A threshold number $minCU$ is introduced to avoid that efficient members are penalized as well, only members with a capacity utilization lower than this threshold number are penalized. The value of the cross section penalty is calculated by dividing threshold number $minCU$ by the CU of the particular member multiplied by a cross section penalty factor (cspf), this factor can be varied during the optimization.

$$\begin{aligned} &\text{for } CU < minCU: \\ &csp = \frac{minCU}{CU} * cspf \end{aligned} \tag{4.8}$$

Cross section penalty's are added to the length of a member during the optimization with the dual interior point method. 'Virtual' length increases volume share of this element. The global optimum with least volume will change and a different solution will be presented with different elements, likely without the penalized elements. Joint cost as well as cross section penalty's are added to the length of elements during the optimization with Eq. (4.9). Every iteration cross section penalties are determined again allowing members to be present in the structure in further iterations. When elements are efficient or not visible in the design, the cross section penalty will be zero.

$$L = L + csp + jc \tag{4.9}$$

A cross section penalty can effectively avoid members to be present in the succeeding iteration. However, not enough influence is exerted to the optimization to obtain efficient solutions. In situations with only large or small cross sections available relative to the optimized cross sections, often no convenient designs can be gathered. In a dense grid with nodes close to each other with with many possible elements, penalized elements will simply be replaced by other elements closeby in the design domain. Layout of structures will slightly change every iteration but will not be more efficient. Cross section penalty's keep being added but the optimization will simply switch back and forth between solutions. In 4.5.4 an accumulating cross section penalty is introduced to improve the penalty system.

4.5.4. Accumulating cross section penalty

Cross section penalty's are redetermined every iteration. To avoid switching back and forth and to penalize very inefficient members more severely, an accumulating cross section penalty (Acsp) is introduced. This penalty is applied to very inefficient members with a capacity utilization below the threshold number $minFinalCU$. Every iteration accumulating penalty's are determined again but unlike with the cross section penalty the value of the penalty is added to the value of the previous iteration, the penalty keeps accumulating and never returns to zero. The accumulating cross section penalty is calculated with an accumulating cross section penalty factor (Acspf) in Eq. (4.10), an example is given in Table 4.3. The accumulating cross section penalty is often most effective with a low accumulating cross section penalty factor, a low value allows the member to return in later iterations if this member, despite having a penalty and being inefficient, contributes to a design with much less volume than designs with different topology. Besides, in designs with different topology the member could actually be efficient. This penalty system is described as a 'soft' penalty system. A member with a capacity utilization below threshold number $minCU$ receives only a cross section penalty, a member with a capacity utilization below threshold number $minFinalCU$ receives an accumulating cross section penalty as well as a normal cross section penalty.

$$\begin{aligned} &\text{for } CU < minFinalCU: \\ &Acsp+ = \frac{minFinalCU}{CU} * Acspf \end{aligned} \tag{4.10}$$

minCU = 0.70 with a cspf of 1.0
minFinalCU = 0.30 with an Acspf of 0.25

Iteration	q	$a_{\text{optimized}}$	a_{stock}	CU	csp	Acsp	Total penalty
1	0.1	0.1	1.0	0.1	7.0	0.75	7.75
2	1.2	1.2	1.5	0.8	0.0	0.0	0.75
3	0.8	0.8	1.6	0.5	1.4	0.0	2.15

Table 4.3: Example cross section and accumulating cross section penalty

The accumulating cross section penalty's are added to the length of the elements avoiding the element to be present in the next and or further iteration. Combined with the cross section penalty and joint cost the total 'virtual' length of an element during the optimization is determined with Eq. (4.11).

$$L = L + Acsp + csp + jc \tag{4.11}$$

4.6. Extra conditions for the optimization to stop

The penalty system with cross section and accumulating cross section penalty's influences the optimization of topology and cross sections, extra length is added to inefficient members during the optimization increasing volume share, in further iterations different solutions are presented. However, the penalty system does not guarantee the final design to be efficient with high capacity utilization for all members. The optimization stops when no violation of strain occurs regardless of anything else. Therefore, a minimum average unity, maximum unity check and minimum reuse percentage are introduced as extra conditions that must be satisfied. The optimization keeps iterating until all conditions are met. Every iteration new designs arise due to the penalisation of inefficient members. In case no design can be found meeting all conditions the optimization stops after a maximum allowed number of iterations. The minimum average unity check for reused elements is equal to threshold number min_{CU} to which cross section penalty's are applied, the maximum unity check is equal to 1.0.

4.7. Limited availability of elements from stock

Designing with a limited database of reclaimed items requires to include availability of elements in the optimization, which is one of the key adaptations of Fig. 4.3 to the original optimization method. In this section a hard and soft method is described.

4.7.1. Hard method

The possible stock member list (PSML), with all possible connections, and connected nodes (Cn), with all added connections to the optimization, are updated after every iteration. When a member is added from the PSML to the Cn and is no longer available, it is deleted from the PSML. This member can not be added elsewhere in the structure, it is impossible for elements to be present in the design more often than they are available. When a member in Cn has an optimized cross section close to zero, an thus not present in the design, it is deleted from Cn and added back to the PSML. In new iterations this member can be added elsewhere in the structure. The initial connectivity will be formed with elements which are NOT available from stock after which they will receive a huge accumulating penalty preventing them to be present in the design in any further iterations. An overview of this process is given in Fig. 4.10, a simple example is given in Fig. 4.11.

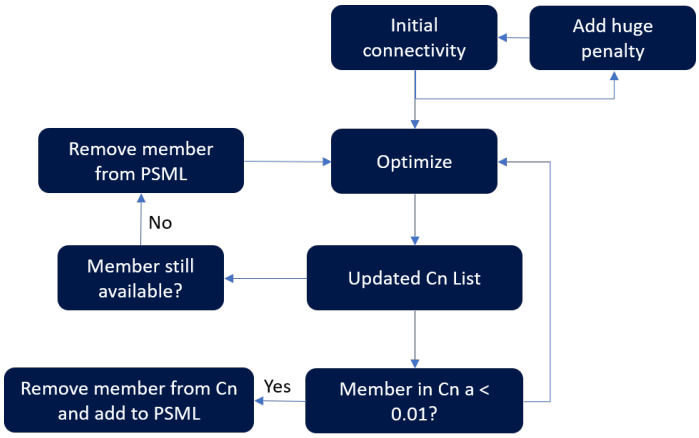


Figure 4.10: Hard method for limited availability of elements from stock

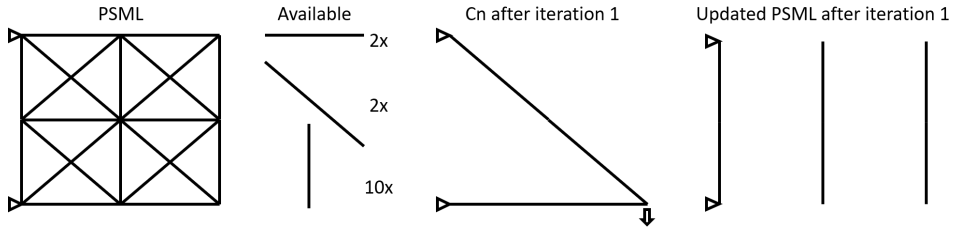


Figure 4.11: Simple example hard method for limited availability

4.7.2. Soft method

After every iteration availability of elements is checked in the optimized structure. When a certain element is present more often than available, the available cross sections from stock will only be assigned to the most efficient members. The other members will receive their optimized cross section and will be considered as new elements, a 'new member penalty' (Nmp) is applied to these members to avoid them being present in the next iteration. Every iteration new designs are generated and the optimization can only stop when a minimal percentage of reuse is achieved. The new member penalty has a constant value just as the joint cost and is added to the length of an element during the optimization with the dual interior point method. An overview of the soft method is given in Fig. 4.12.

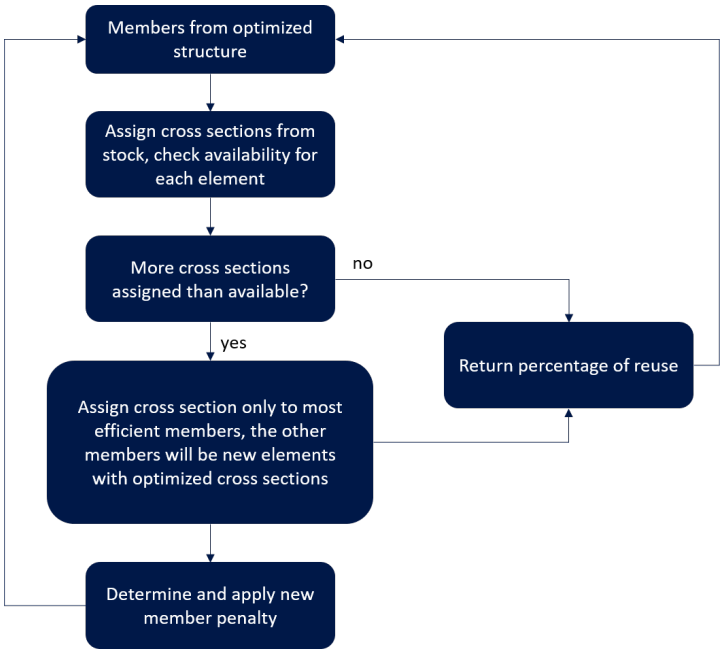


Figure 4.12: Soft method for limited availability of elements from stock

Optimizing assignment of cross sections from stock

With the capacity utilization function described in Section 4.3 the most efficient cross section from stock is assigned to each member. Availability is checked and only the most efficient members receive the cross section from stock. Multiple cross sections can be available for a certain member. Therefore, before less efficient members are considered as new members, it is checked if other cross sections from stock are available and if assigning this cross section would improve overall efficiency of the design. To illustrate this optimization process of assigning cross sections an example is described.

In this example 6 members are considered all with the same length and thus the same cross sections available. The maximum unity check is 1.0 and available cross sections are:

- 1x 1.0
- 2x 2.0

When a unity check is higher than 1.0, 100 is added to the capacity utilization factor (CUf). The CUf for all members is given in Table 4.4.

Member	Internal force (q)	Assigned cross section (A_s)	Capacity utilization factor (CUf)
1	1.0	1.0	1.0
2	0.9	1.0	1.11
3	3.0	2.0	101.5
4	5.0	2.0	102.5
5	1.5	2.0	1.33
6	4.0	2.0	102.0

Table 4.4: Example optimizing assignment of cross sections: Assigned cross sections and capacity utilization factors

The optimization is an iterative process, availability of cross sections is checked starting with the smallest cross section and ending with the largest cross section. In this example the optimization starts with a cross section of 1.0. Availability and CUf for all members with A_s = 1.0 is checked.

Member	Internal force (q)	Assigned cross section (A_s)	Capacity utilization factor (CUf)
1	1.0	1.0	1.0
2	0.9	1.0	1.11

Table 4.5: Example optimizing assignment of cross sections: Members and capacity utilization factors for A_s = 1.0

A cross section of 1.0 is assigned to 2 elements while only 1 is available. This cross section will only be assigned to the member with the lowest CUf, thus most efficient member, which is member 1. For member 2 possible other larger available cross sections are checked and if any exist this member will receive the first cross section larger than its initial assigned cross section, if none exist this member will be a new member and receive its optimized cross section. A cross section of 2.0 is available for member 2 and will be assigned.

In the second iteration, availability of the second smallest cross section is checked, this is A_s = 2.0. Member 2 with q = 0.9 is now also included. Availability and CUf for these members is checked.

Member	Internal force (q)	Assigned cross section (A_s)	Capacity utilization factor (CUf)
2	0.9	2.0	2.22
3	3.0	2.0	101.5
4	5.0	2.0	102.5
5	1.5	2.0	1.33
6	4.0	2.0	102.0

Table 4.6: Example optimizing assignment of cross sections: Members and capacity utilization factors for A_s = 2.0

A cross section of 2.0 is available 2 times but is assigned 5 times. Only the 2 members with lowest CUf, thus most efficient, will receive a cross section of 2.0, these are member 2 and 5. For the other members no other possible larger cross sections are available and they will receive their optimized cross sections and will be new elements. The final assigned cross sections are given in Table 4.7.

Element	1 (Reuse)	2 (Reuse)	3 (New)	4 (New)	5 (Reuse)	6 (New)
q	1.0	0.9	3.0	5.0	1.5	4.0
A_s	1.0	2.0	3.0	5.0	2.0	4.0
UC	1.0	0.45	1.0	1.0	0.75	1.0

Table 4.7: Example optimizing assignment of cross sections: Final cross section for each element

The optimization iterates from smallest to largest cross section since members can receive a larger but not a smaller cross section, this will result in violation of maximum unity check. During the process of checking availability and assigning cross sections, elements with a unity check equal or below 1.0 have a preference above elements with a unity check above 1.0. Assigning cross sections to the members with lowest CUf will

result in the highest average capacity utilization (CU). For a large optimization problem with a large stock with many different elements, computation times can increase significantly.

4.7.3. Discussion and applied method

The hard method will always result in 100% reuse, during the optimization however, only a limited number of solutions is possible. When a certain element is at a certain location and receives an inefficient cross section, this element will not be present in the design of the next iteration. However, this element can not be added at a different location in the design since it has not yet been removed from the Cn and added back to the PSML. A method could be to remove inefficient members from the Cn at all time, a drawback of this method is that no allowance is made for inefficient members to be present in the final design. The number of available members will be very limited in each iteration.

In optimization problems with over 10.000 possible connections and only 100 available elements, it is expected that the optimization will not run properly and would require many iterations to find, if possible, a convenient solution meeting all requirements.

With the soft method designs could be made with less than 100% reuse. Aiming for 100% reuse is possible but could require many iterations and many new member penalty's. The new member penalty combined with joint cost, cross section and accumulating cross section penalty's could reduce effectiveness of the penalty system when too many penalty's must be applied. Determining lower requirements for efficiency and reuse percentage could reduce the number of applied penalty's.
Optimization problems with a large stock of reclaimed items could result in designs with 100% reuse but for problems with a smaller database 80% reuse could also be a convenient result.

The soft method has been chosen since this approach offers the possibility for designs with less than 100% reuse. Besides, by varying the value of the new member penalty, multiple designs can be generated expanding the number of possible designs.

4.7.4. New member penalty

The new member penalty (Nmp) has a constant value and is applied to all new members in the design, avoiding these members to be present in the next iteration. Every iteration availability is checked again and penalty's are redetermined. Similar problems as with the cross section penalty described in Section 4.5.3 could occur, the optimization could switch back and forth between solutions with a low percentage of reuse.

The problem of switching back and forth is solved by applying an accumulating new member penalty in addition to the new member penalty after a specified amount of iterations. In early iterations, where often no efficient designs have been found yet, only new member penalty's are applied. In later iterations new members reappearing receive an accumulating new member penalty. The 'soft' method of penalizing members, similar as with the accumulating cross section penalty, is used with a low value for the accumulating new member penalty.

Just as with all other penalty's, the new member and accumulating new member penalty is added to the length of each element during the optimization with the dual interior point method. The total length of each elements is given with Eq. (4.12).

$$L = L + Acsp + csp + ANmp + Nmp + jc \tag{4.12}$$

4.7.5. Cross section penalty's for new members

Distinction is made between potential efficient and potential inefficient new members, implying whether new members would have been efficient if they were reused members with their most efficient assigned cross section from stock. Therefore, cross section and accumulating cross section penalty's are determined for new members based on their initial assigned cross section from stock. Whether a penalty should be applied and what the value of this penalty should be, is determined similarly as it is for reused members with Eq. (4.8) and Eq. (4.10).
New members can be present in further iterations but will only be when an efficient cross section could be assigned. This prevents new members to be inefficient reused members in further iterations.

4.8. Different penalty systems

Different systems with different formula's for applying cross section penalty's were considered given in Fig. 4.13. System A determines penalty's based on the absolute difference of the optimized and assigned cross section. System B and C determine the value of penalty's based on the capacity utilization of members, the difference is that with system C the cross section as well as the accumulating cross section penalty is applied to members with a capacity utilization below threshold number minFinalCU, the accumulating cross section penalty factor has a lower value than with system B. System D makes no distinction between the degree of inefficiency of members, penalty's are simply determined with the penalty factors. An advantage of this system could be that members with extreme inefficiency do not receive extreme penalty's, allowing them to be present in later iterations.

System A

$$for\ minFinalCU < UC < minCU:$$

$$csp = abs\left(a_{optimized} - a_{assigned}\right) * cspf$$

$$for\ UC < minFinalCU:$$

$$Acsp \mathrel{+}= abs\left(a_{optimized} - a_{assigned}\right) * Acspf$$

System B

$$for\ minFinalCU < UC < minCU:$$

$$csp = \frac{minCU}{CU} * cspf$$

$$for\ UC < minFinalCU:$$

$$Acsp \mathrel{+}= \frac{minFinalCU}{CU} * Acspf$$

System C

$$for\ UC < minCU:$$

$$csp = \frac{minCU}{CU} * cspf$$

$$for\ UC < minFinalCU:$$

$$Acsp \mathrel{+}= \frac{minFinalCU}{CU} * Acspf$$

System D

$$for\ minFinalCU < UC < minCU:$$

$$csp = cspf$$

$$for\ UC < minFinalCU:$$

$$Acsp \mathrel{+}= Acspf$$

Figure 4.13: Formula's for penalty system A, B, C and D

All systems were tested for the final optimization tool with all adaptations and additions to the member adding scheme of He et al. (2019). Input parameters were varied and different designs were obtained. Designs with highest average unity check, highest percentage of reuse, least number of members and least volume are compared for the four systems. Tests were performed for 7 optimization problems shown in Fig. 4.14 with 7 different stocks of available elements given in Fig. 4.15, the stock of reclaimed elements for optimization problem 7 contains all steel members of the Provinciehuis Zuid-Holland.

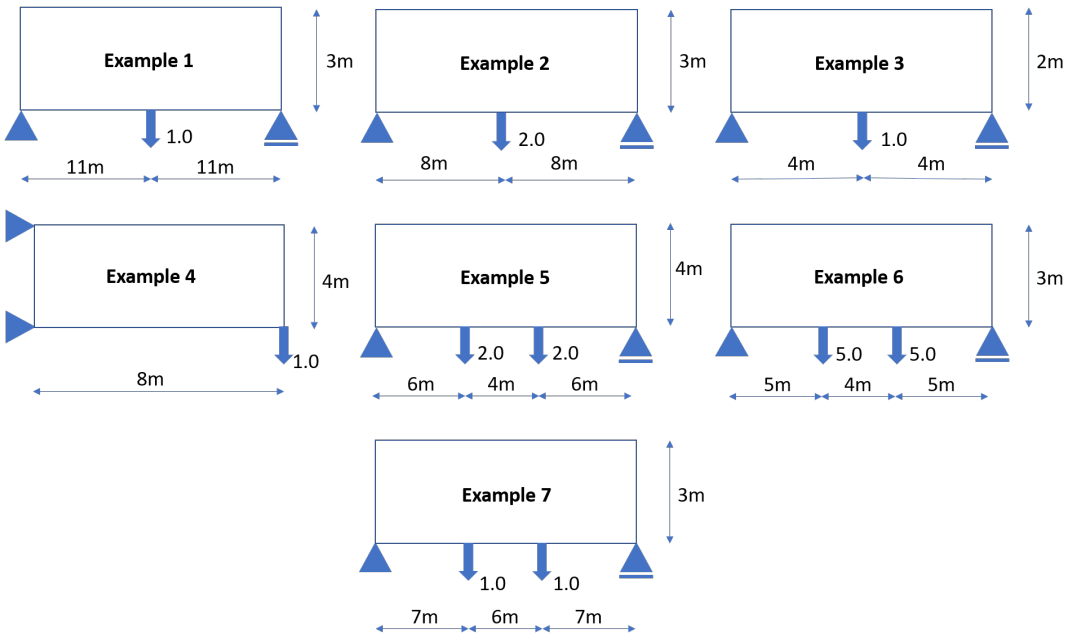


Figure 4.14: Load cases for testing system A, B, C and D

Stock A		
L	A	n
4.0	0.6	4
4.5	0.7	8
3.5	0.4	4
4.8	0.7	12
3.8	0.6	10
6.0	0.8	8
3.2	0.5	4
5.2	0.8	12

Stock B		
L	A	n
2.0	1.4	4
4.5	2.2	6
3.5	1.8	10
4.8	2.5	10
3.8	1.6	6
6.0	4.0	8
3.2	1.5	4
5.2	2.6	4

Stock C		
L	A	n
1.4	0.5	18
1.4	0.7	18
1.4	0.8	16
1.4	0.3	18
1.4	1.0	16
1.0	1.0	14

Stock D		
L	A	n
2.0	0.3	4
2.2	0.8	8
2.8	1.2	4
3.5	1.1	4
4.0	1.3	4
2.0	0.2	8
2.8	0.5	6

Stock E		
L	A	n
2	3.3	4
2.2	4.1	6
2.8	5.5	6
3.5	5.8	8
4.0	8.8	6
2.2	5.6	4
3.2	6.0	8
2.5	7.5	4
4.5	7.0	8
6.0	9.5	4

Stock F		
L	A	n
2	3.3	4
2.2	4.1	6
2.8	5.5	6
3.5	5.8	8
4.0	8.8	6
2.2	5.6	4
3.2	6.0	8
2.5	7.5	4
4.5	7.0	8
6.0	9.5	4

Figure 4.15: Stocks A to F for Load cases 1 to 6

An overview of all test results is given in Fig. 4.16. System A and B are the worst performing systems, especially for optimization problems with stocks with large cross sections and large deviation of cross sections. System A could be interpreted as a system penalizing volume increase over inefficiency of members, the larger the cross sections the larger the volume increase will be for inefficient members. With large cross sections penalty's will easily become very high, no allowance is made for members to reappear in later iterations. On the other hand, small cross sections result in very low penalty's.

System B penalizes members based on efficiency, however the accumulating cross section penalty can be

ineffective in many situations. With large deviations of optimized and assigned cross sections, the accumulating penalty quickly become very high. With low values for the accumulating cross section penalty influence of the penalty is too little, members reappear in the next iteration. As for system A no results could be obtained for example 5 and 6.

System C is awarded as the best performing system, many different designs were generated for all examples and often the design with least volume, highest average unity check, highest percentage of reuse and least amount of members, compared to the other systems, could be obtained. System C is not inferior to system D for example 1 and 4, difference in volume and number of members are only small. System D performs well for almost all examples, however, not as good as system C and is expected neither to do so for other optimization problems and stocks of reclaimed elements. No distinction is made in terms of inefficiency of members, penalty's are awarded too randomly reducing effectiveness of the penalty system. Therefore, system C is implemented in the optimization method for calculating penalty's. The best performing designs of system C for example 3 are provided in Appendix A with a comparison to designs with the original member adding scheme.

Example 1	System A - 1 Design				System B - 1 Design				System C - 4 Designs				System D - 4 Designs			
Design:	Vol	UC	Re	Mems	Vol	UC	Re	Mems	Vol	UC	Re	Mems	Vol	UC	Re	Mems
Least volume	100	0.75	84	31	105	0.76	75	40	100	0.78	83	30	99	0.76	82	34
Highest UC	100	0.75	84	31	105	0.76	75	40	100	0.78	83	30	104	0.8	78	36
Highest Reuse %	100	0.75	84	31	105	0.76	75	40	100	0.78	83	30	99	0.76	82	34
Least members	100	0.75	84	31	105	0.76	75	40	100	0.78	83	30	99	0.76	82	34

Example 2	System A - 1 Design				System B - 1 Design				System C - 3 Designs				System D - 4 Designs			
Design:	Vol	UC	Re	Mems	Vol	UC	Re	Mems	Vol	UC	Re	Mems	Vol	UC	Re	Mems
Least volume	130	0.84	80	15	136	0.77	93	15	106	0.9	83	12	127	0.76	93	14
Highest UC	130	0.84	80	15	136	0.77	93	15	106	0.9	83	12	127	0.76	93	14
Highest Reuse %	130	0.84	80	15	136	0.77	93	15	127	0.76	93	14	127	0.76	93	14
Least members	130	0.84	80	15	136	0.77	93	15	106	0.9	83	12	127	0.76	93	14

Example 3	System A - 3 Design				System B - 4 Designs				System C - 14 Designs				System D - 8 Designs			
Design:	Vol	UC	Re	Mems	Vol	UC	Re	Mems	Vol	UC	Re	Mems	Vol	UC	Re	Mems
Least volume	25	0.89	100	38	25	0.92	90	38	23	0.95	100	32	23	0.95	100	32
Highest UC	25	0.92	100	38	25	0.92	90	38	25	0.98	100	26	23	0.95	100	32
Highest Reuse %	25	0.89	100	38	25	0.89	100	38	26	0.94	100	28	25	0.92	100	38
Least members	25	0.89	100	38	25	0.88	100	34	26	0.96	100	26	23	0.95	100	32

Example 4	System A - 3 Designs				System B - 3 Designs				System C - 8 Designs				System D - 8 Designs			
Design:	Vol	UC	Re	Mems	Vol	UC	Re	Mems	Vol	UC	Re	Mems	Vol	UC	Re	Mems
Least volume	46	0.79	80	20	46	0.81	79	19	46	0.75	85	20	45	0.77	80	20
Highest UC	46	0.79	80	20	51	0.85	86	22	50	0.87	77	26	48	0.85	76	21
Highest Reuse %	47	0.77	85	20	51	0.85	86	22	47	0.8	91	23	49	0.75	90	21
Least members	47	0.77	85	20	46	0.81	79	19	46	0.76	85	20	46	0.81	83	18

Example 5	System A - 0 Designs				System B - 0 Designs				System C - 7 Designs				System D - 5 Designs			
Design:	Vol	UC	Re	Mems	Vol	UC	Re	Mems	Vol	UC	Re	Mems	Vol	UC	Re	Mems
Least volume									186	0.71	100	17	194	0.78	100	17
Highest UC									194	0.78	100	17	194	0.78	100	17
Highest Reuse %									186	0.71	100	17	194	0.78	100	17
Least members									194	0.78	100	17	194	0.78	100	17

Example 6	System A - 0 Designs				System B - 0 Designs				System C - 2 Designs				System D - 2 Designs			
Design:	Vol	UC	Re	Mems	Vol	UC	Re	Mems	Vol	UC	Re	Mems	Vol	UC	Re	Mems
Least volume									451	0.77	90	21	446	0.79	84	25
Highest UC									451	0.77	90	21	506	0.8	84	25
Highest Reuse %									451	0.77	90	21	446	0.79	84	25
Least members									451	0.77	90	21	446	0.79	84	25

Example 7	System A - 1 Design				System B - 2 Designs				System C - 3 Designs				System D - 2 Designs			
Design:	Vol	UC	Re	Mems	Vol	UC	Re	Mems	Vol	UC	Re	Mems	Vol	UC	Re	Mems
Least volume	150	0.72	92	12	130	0.84	92	12	130	0.84	92	12	143	0.74	93	15
Highest UC	150	0.72	92	12	130	0.84	92	12	130	0.84	92	12	143	0.74	93	15
Highest Reuse %	150	0.72	92	12	130	0.84	92	12	130	0.84	100	12	163	0.71	100	14
Least members	150	0.72	92	12	130	0.84	92	12	130	0.84	92	12	163	0.71	100	14

Figure 4.16: Comparison of penalty system A, B, C and D

5

Illustrations, examples and guidelines

The optimization tool for designing with reused elements described in Chapter 4 consists of several functions. In this chapter, examples are provided illustrating effectiveness and influence of individual and combinations of functions on the optimization. Examples are given for one of the three load cases in Fig. 5.1. In Section 5.6 Guidelines are provided for all input parameters to perform successful optimizations. An example is provided in Section 5.8 for a truss with a span of 22m. Designs optimized with the original member adding scheme of He et al. (2019) and member adding scheme with reused elements are compared to a standard design made in SCIA. A comparison between the developed method en method of Brütting et al. (2019b) for designing with reused elements is provided in Section 5.7.

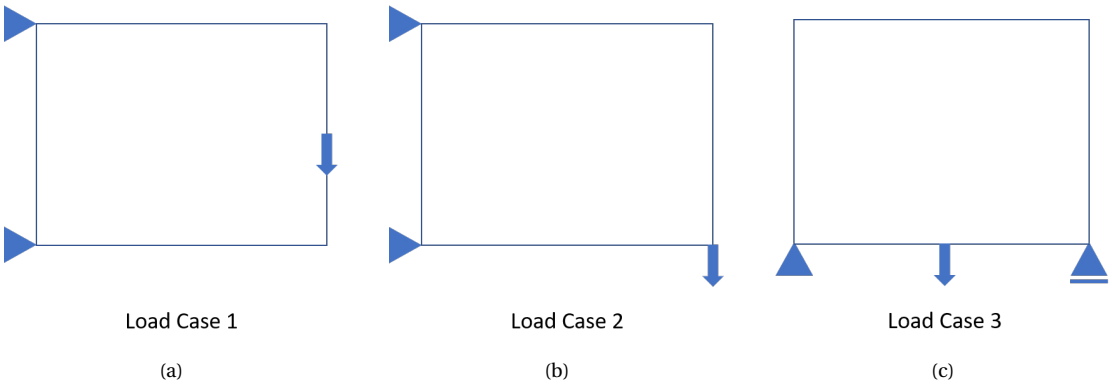


Figure 5.1: Load cases for testing adaptations and additions to member adding scheme of He et al. (2019)

5.1. Possible stock member list and scaling

The number of members in the PSML depends on the available lengths from stock and values for $Ł_{short}$ and $Ł_{cut}$. More members in the PSML result in more possible solutions and could result in a more efficient design. An example is given in Fig. 5.2. Load case 1 is considered with a load of 1.0 and joint cost of 1.0 in a 6x2 grid. The available lengths are:

$L = 1.20, 2.24(2x1), 2.82(2x2), 3.16(1x3), 3.61(2x3)$

To create a stable initial structure all members with $L < 3.17$ should be in the initial connected structure. The original member adding scheme in Fig. 5.2(a) is compared with the adapted scheme with $Ł_{cut} = 0.2$ in Fig. 5.2(b) and $Ł_{cut} = 0.5$ in Fig. 5.2(c). The adapted scheme with $Ł_{cut} = 0.2$ gives a different less efficient result with higher volume than the original scheme, $Ł_{cut} = 0.5$ results in the same design as the original scheme since all elements required for this design are in the PSML. The only difference is that there are 12 members in the structure instead of 8, the members of 4m consist out of 2 members of respectively 1m and 3m. Members in compression are plotted in red, members in tension in blue.

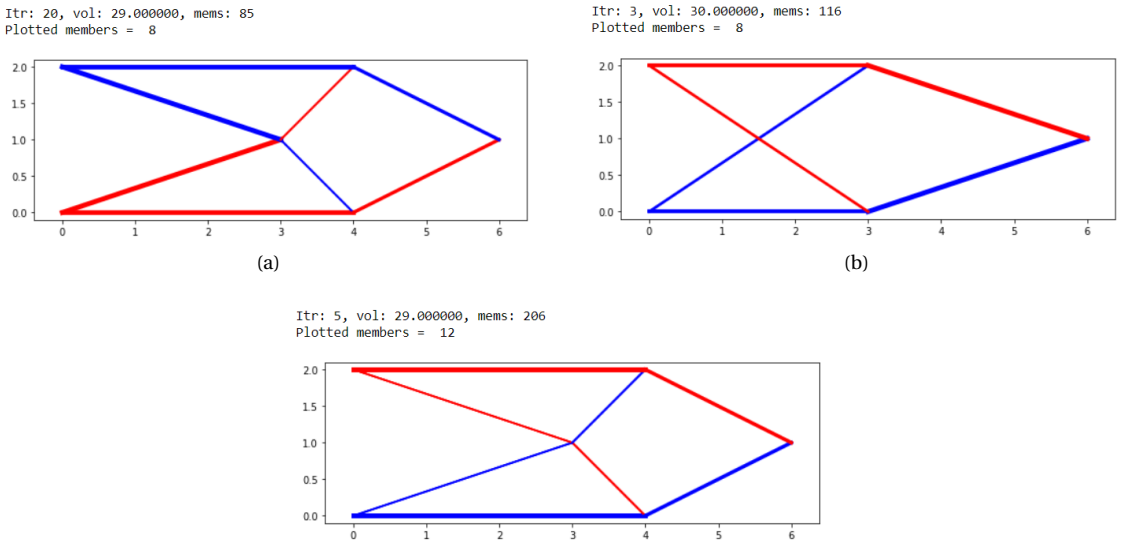


Figure 5.2: Result: original member adding scheme (a), PSML with cut off length of 0.2 (b) and PSML with cut off length of 0.5 (c)

5.1.1. Scaling

The number of possible designs increases with the number of members included in the PSML. Number of members could be increased by applying a scaling factor, implying a denser nodal density. An example is given for a 4x2 grid in Fig. 5.3. In Fig. 5.3(b) a scaling factor of 6 is applied, the grid, length of available elements, $Ł_{short}$ and $Ł_{cut}$ are multiplied by the scaling factor. Figure Fig. 5.3(b) gives a much more convenient design than Fig. 5.3(a) with lower volume and less members. Downscaling the design results in a structure with only lengths available from stock and a maximum $Ł_{cut}$ of 10 cm.

The available lengts are $L = 2m, 1x3m, 2x3m, 5x4m$ with an initial connectivity of $L \leq 2x3$. The joint cost penalty is 1.0 and the maximum $Ł_{cut}$ 0.1m.

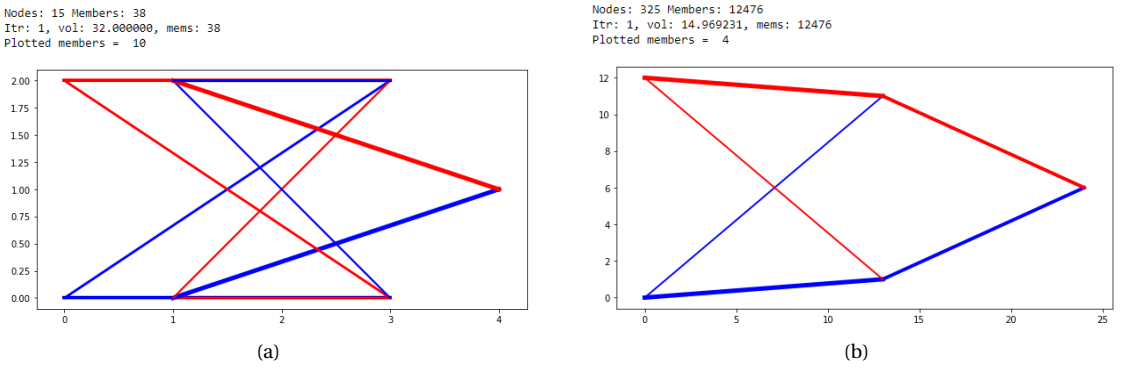
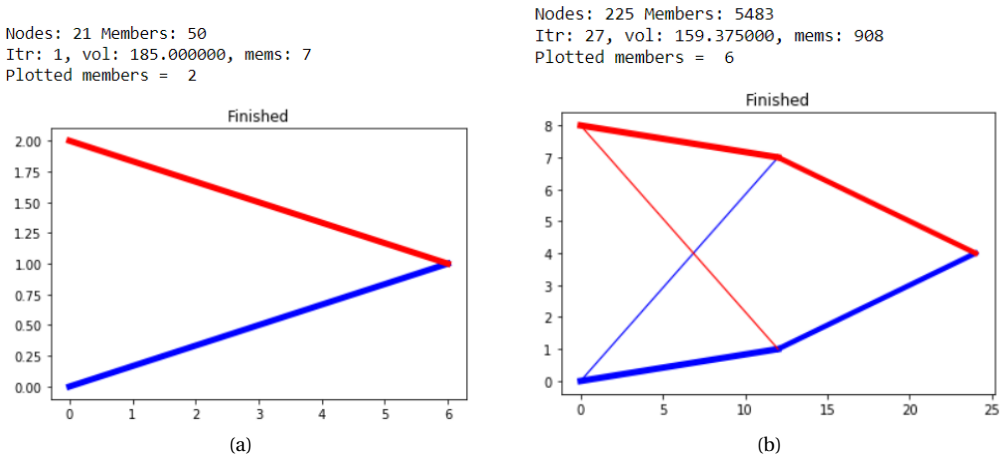


Figure 5.3: Result: PSML with a scaling factor of 1 (a) and PSML with a scaling factor of 6 (b)

The scaling factor could also be applied to obtain, after a suitable result has been found, a more efficient solution. In Fig. 5.4(a) a convenient design has been given but a more efficient design with lower volume could be obtained by applying a scaling factor of 4 in Fig. 5.4(b).

The available lengths are: $L = 2m, 2.5m, 3m, 3.5m, 4m, 6m$. The initial connectivity consist out of elements with a length of 6x1, the joint cost penalty is 1.0 and the maximum $Ł_{cut}$ = 0.1m.



(a) (b)

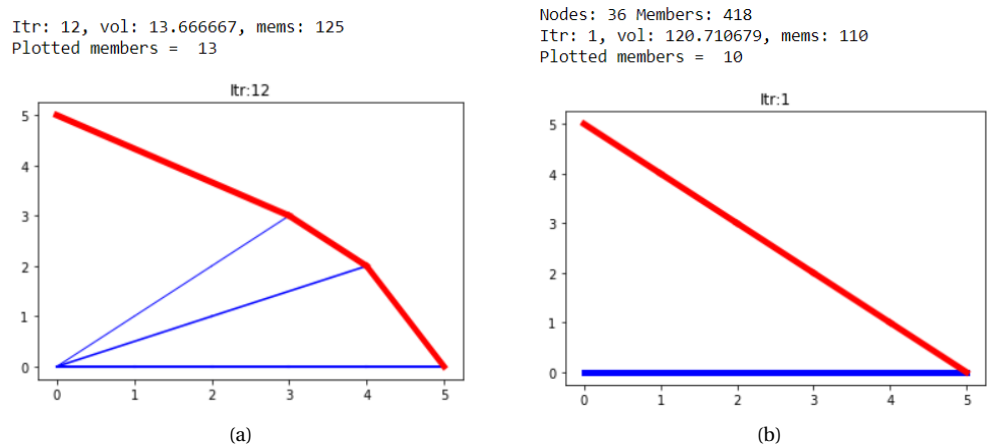
Figure 5.4: Example possible stock member list with scale factor of 1 (a) and 4 (b)

5.2. Possible stock member list, capacity utilization and matrix method

In this section examples are given for the member adding scheme with de combined adaptations and additions of capacity utilization and matrix method combination of possible stock member list, capacity utilization and matrix method. Illustrations are given for the influence of recalculating the structure in Section 5.2.1 and initial connectivity in Section 5.2.2. In all examples lengths and cross sections are not linked to each other, every available cross section can be assigned to every length. Examples are given for load case 1 or 2 from Fig. 5.1.

5.2.1. Recalculating the structure

Every iteration, cross sections are assigned to the optimized cross sections and the structure is recalculated for virtual displacements and strain with the matrix method. An example is given in Fig. 5.5 to illustrate the effectiveness of this method. In this example only elements with a cross section of 100 are available resulting in an oversized structure after assignment of elements. The structure is verified for virtual strain and no violation occurs. The optimization stops after the first iteration and the design of Fig. 5.5(b) is presented. In the original member adding scheme the optimization continues until iteration 12 and the design of Fig. 5.5(a) is presented with different topology and more members.



(a) (b)

Figure 5.5: Example original member adding scheme (a) and adapted scheme with CU and only cross sections available of 10(b)

Oversized structures meet requirements for violation of strain and stop the optimization. However, optimization problems with high loads and small available cross sections never meet these requirements. An example

is given in Fig. 5.6 for load case 2 with only cross sections of 0.1 available. A pointload of 1.0 is applied.

After the first iteration the optimized structure consists out of members with cross sections of about 0.5. The unity check of all elements is 5.0 and large displacements are recalculated, violation of strain occurs and new members are added. A new structure is optimized and cross sections are assigned, again all with a high unity check. From iteration 7 new optimal structures are formed similar to the previous ones. Iterations continue and eventually new added members barely influence the optimization. New added connected members simply receive an optimized cross section close to zero, are not visible in the plot, and no cross sections from stock are assigned. Displacements of the structure and unity check of assigned cross sections remain high. Eventually, so many members are added that even these members with optimized cross sections close to zero and no assigned cross sections result in a structure were no violation of strain occurs, this process can be seen from iteration 14 till iteration 62 where the design of the structure does not change. The final structure has been found but all unity checks of all members with assigned cross section are too high.

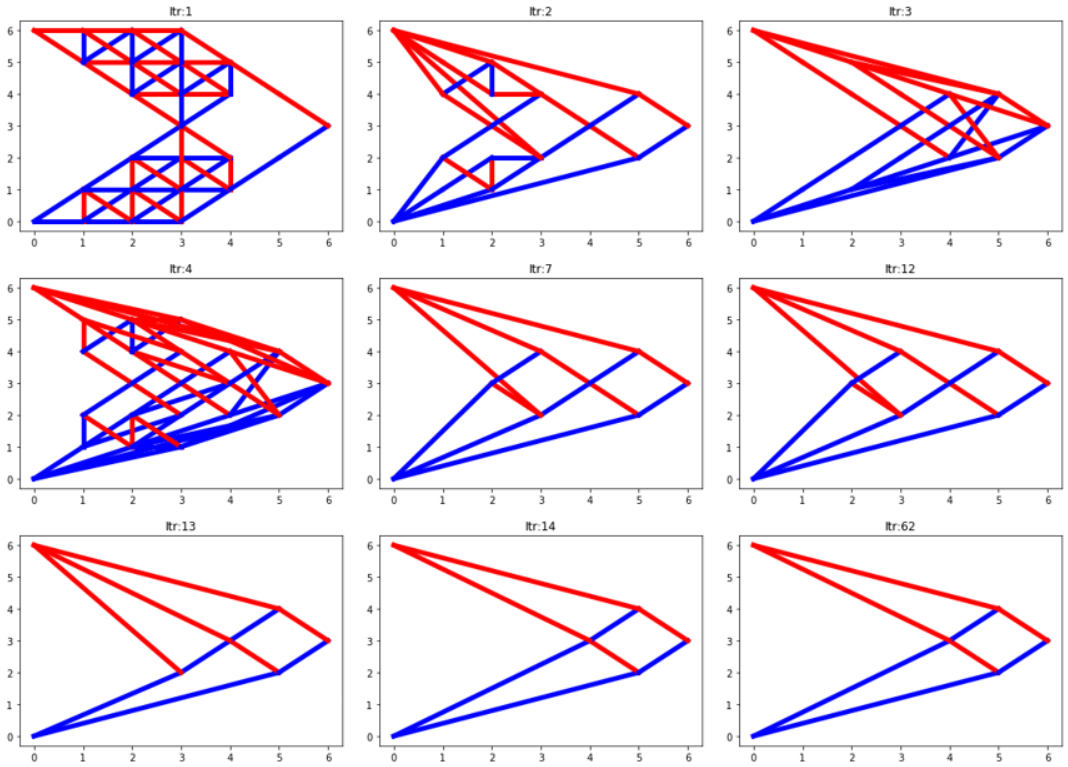


Figure 5.6: Example adapted scheme with CU and too small available cross sections

5.2.2. Influence of initial connectivity

Stocks with only large available cross sections could meet requirements for virtual strain early, resulting in unnecessarily oversized structures. An example is given for load case 2 Fig. 5.7. A point load of 1.0 is applied and only cross sections of 1.0 are available, the joint cost penalty is zero. In the figures on the left in Fig. 5.7 in iteration 3 and 4, many members are present in the optimized structure to which all a cross section of 1.0 is assigned. In iteration 4 no violation of strain occurs and the optimization stops resulting in an oversized structure with 29 members with a volume of 41.1, a design with much less members would have resulted in a design with much less volume. The original member adding scheme presents a design with only 13 members with small cross sections and a volume of 13.7 shown in Fig. 5.7 on the right.

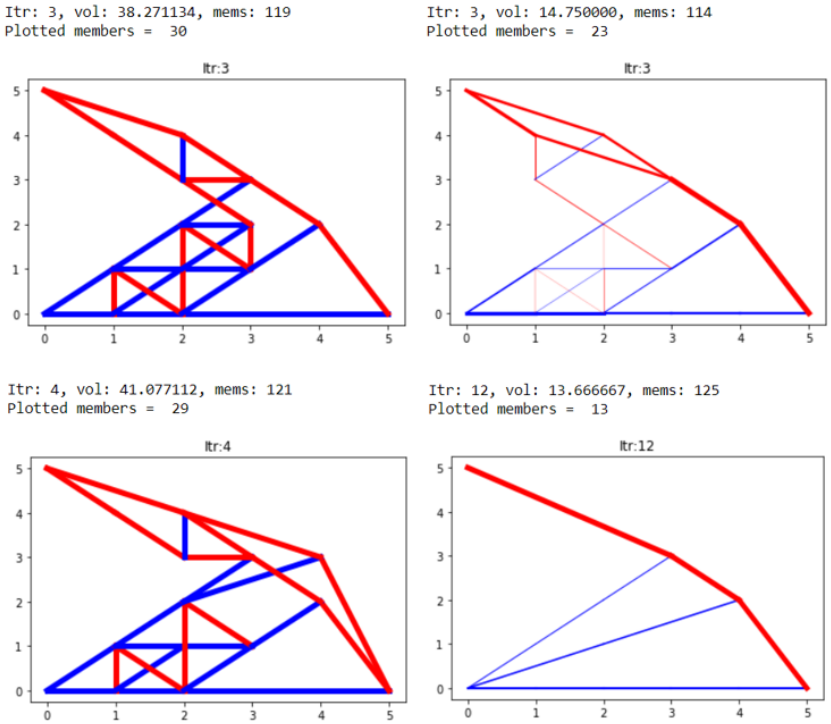
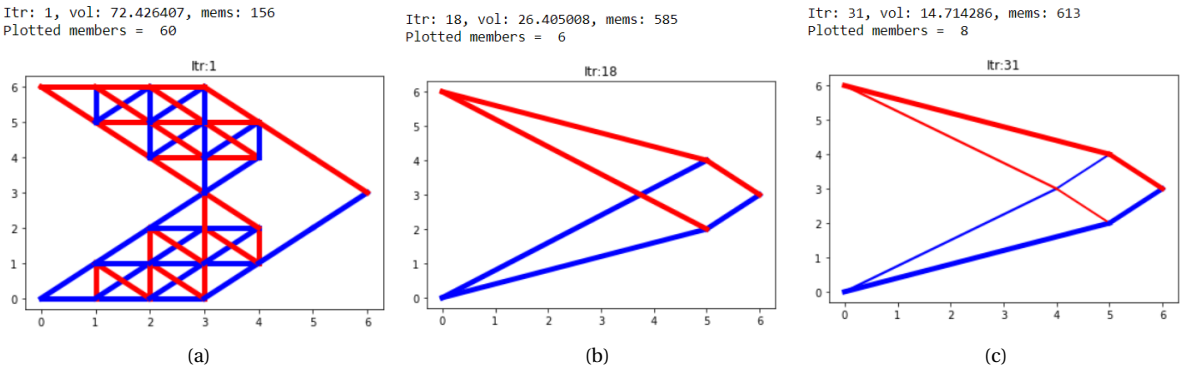


Figure 5.7: Example adapted scheme with too large available cross sections (left) and original member adding scheme (right)

In the original member adding scheme initial connectivity had no influence on the final result, however, with the adaptation and addition of implementing assignment of cross sections from stock and recalculating displacements it does. To illustrate this influence an example is given in Fig. 5.8 for load case 1, only cross sections of 1.0 available for a point load of 1.0. In Fig. 5.8(a) the initial connectivity consists of members with a length 1m or 1.41m. In the first iteration small internal forces are present in most members resulting in an oversized structure, requirements for virtual strain are met and the optimization stops. In Fig. 5.8(b) the initial connectivity has been adapted to elements with a length of 6x3 only, including elements going from the supports to the point load. Internal forces are high, violation of strain occurs and new members are added. 18 Iterations are required and the final structure is presented with 6 elements with much less volume than the structure of Fig. 5.8(a). However, compared to the optimized structure with the original member adding scheme in Fig. 5.8(c) the structure is still oversized, a structure with less than 6 members would have resulted in a design with lower volume and no violation of strain either.

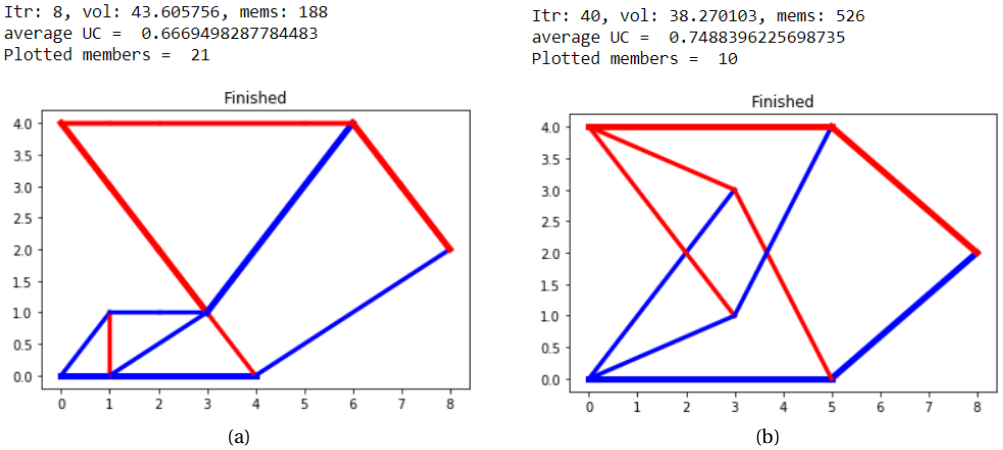


(a) (b) (c)

Figure 5.8: Example adapted scheme with initial connectivity < 1.42 (a) and initial connectivity 6x3 (b). Original optimization scheme with initial connectivity < 1.42 (c)

A reduced initial connectivity can avoid issues with oversized structures and can, in some situations, improve influence of the member adding proces. An example is given in 5.9 for a 8x4 grid, in Fig. 5.9(a) the initial

connectivity is created with members with a length smaller or equal to 1.41m, in Fig. 5.9(b) with members with a length between 2m and 2.82m. Fig. 5.9(b) gives a much more convenient result with less members and less volume.



(a) (b)

Figure 5.9: Example adapted scheme with CU and MM with initial connectivity < 1.42 (a), and initial connectivity 2m and 2x2m (b)

Negative influence of initial connectivity

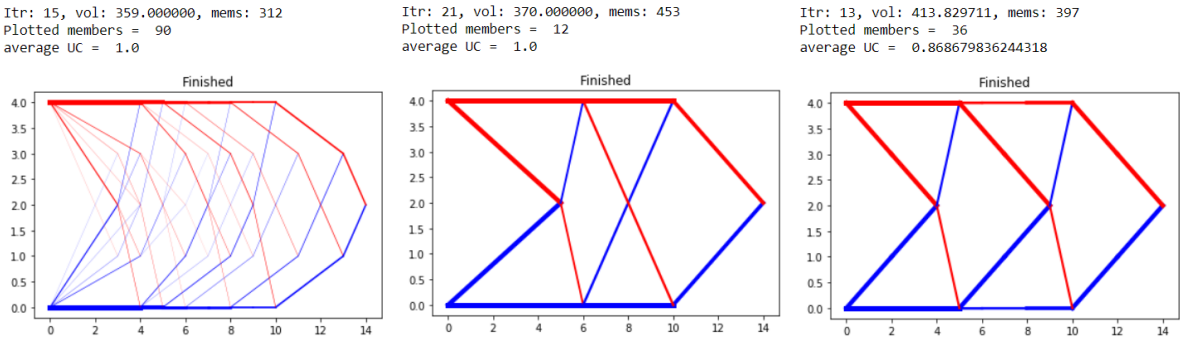
Designing with reused members often results in oversized elements, requirements for virtual strain are met early and no members are added any more. Many potential efficient members are not considered resulting in designs with larger volume or inefficient topology. Therefore, in many situations it is beneficial to include all members in the initial connectivity, the member adding principle is rejected increasing computation times but quality of designs improves.

An example with a more effective optimization with all members included in the initial connectivity is given in Fig. 5.10 for load case 1 for a 4x14 grid. A large variety of cross sections is available for all available lengths. Initial connectivity is created with lengths < 4m. available lengths are:

$L = 2, 2.24(1x2m), 4, 4.12(4x1m), 4.47(4x2m), 5, 5.5, 6, 6.7$

Optimized designs with the original member adding scheme are given in Fig. 5.10(a) and Fig. 5.10(b) without and with joint cost respectively. The optimized structure with the adapted optimization method with possible stock member list, capacity utilization and matrix method is given in Fig. 5.10(c), a convenient design is presented with only a small increase in volume compared to Fig. 5.10(b). However, a defect is that the structure consists out of 36 members, members of 2x4m consist out of two members of 1x2m while members of 2x4m are actually available. Initial connectivity is causing the problem, 397 members were included in the connected node list in the final iteration, not including members with a length of 2x4m. With an average unity check of 0.86 the design is slightly oversized, no violation of strain occurs and members of 2x4m were not added to the connected node list.

Changing the initial connectivity to lengths < 5m, with all elements required for the design of Fig. 5.10(c) resulted in the design of Fig. 5.11 shown on scale, only 1 iteration was required. A contradictory method since the principle of the original member adding scheme reducing computation times is rejected. However, a more effective method with increased quality of the design.



(a) (b) (c)

Figure 5.10: Original member adding scheme jc = 0 (a), Original member adding scheme jc = 1 (b), Adapted scheme with CU, MM and PSML jc = 1 (c),

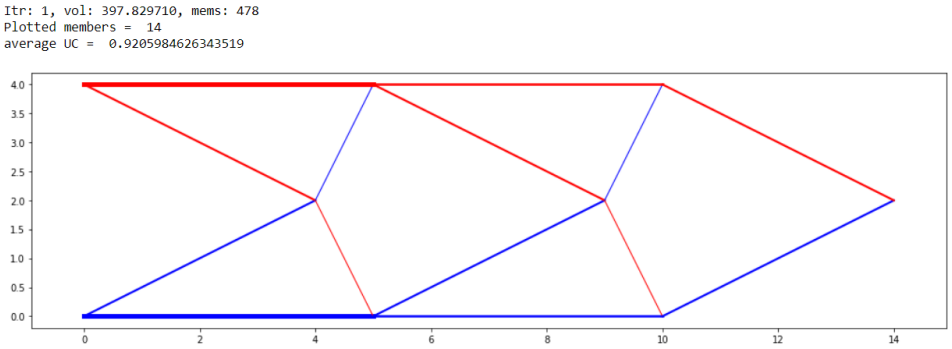
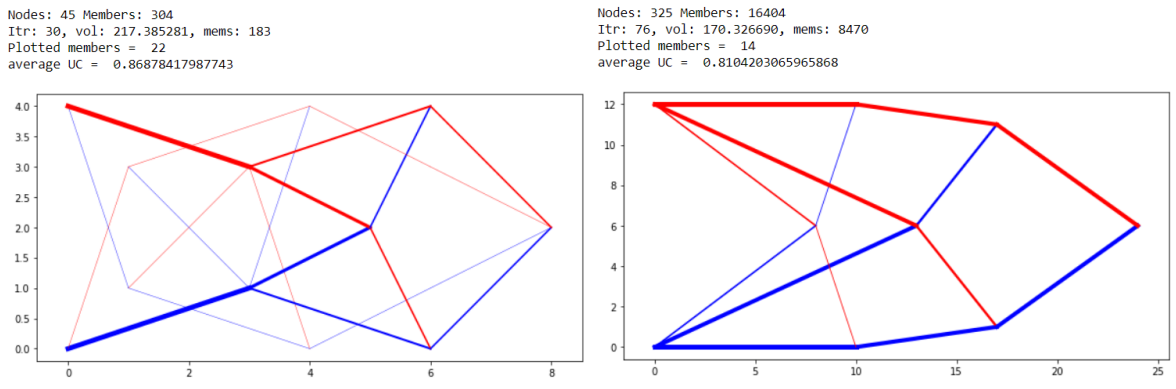


Figure 5.11: Adapted scheme including only required members in PSML and full initial connectivity

The principle of the member adding scheme with a reduced initial connectivity should not be rejected completely. It can be effective in some situations, especially to reduce computation times. An example is given in Fig. 5.12 for load case 1 for a 8x4 grid with and without scaling factor. The initial connectivity is formed with elements with length < 2.85m. With a scaling factor of 1.0 an efficient design was found in terms of average unity check, however, many members are present increasing volume. Applying a scaling factor of 3.0 results in enormous increase of possible members, computation times were reduced by using a reduced initial connectivity. A solution with less members and less volume was found.



(a) (b)

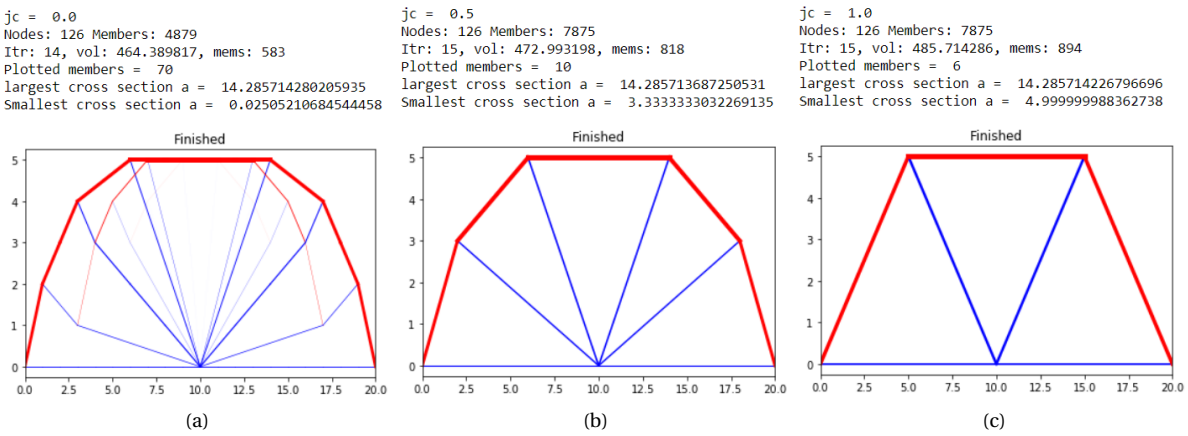
Figure 5.12: Example 8x4 grid adapted scheme with CU, MM and PSML with reduced initial connectivity with scaling factor of 1 (a) 3 (b)

5.3. Influencing optimization of cross sections

Three methods introduced to influence the optimization of cross sections are joint cost, a maximum cross section and an cross section penalty. Examples are provided to illustrate influence of and problems raising for all three methods.

5.3.1. Joint cost penalty

Joint cost penalizes the presence of any member in the design, volume share of every member is increased. Varying the joint cost influences the number of members present in the design and thus the cross sections. Examples are given in Fig. 5.13 for load case 3 with the original member adding scheme with joint cost of 0.0, 0.5 and 1.0. A point load of 10.0 is applied, smallest and largest cross section present in the structure are given. Designs with less members have larger cross sections.

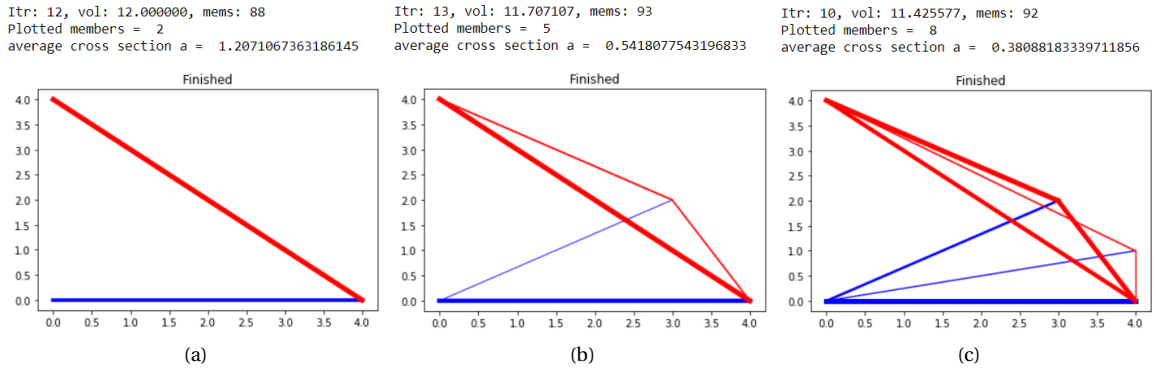


(a) (b) (c)

Figure 5.13: Smallest and largest cross section for original member adding scheme with jc of 0.0 (a), 0.5 (b) and 1.0 (c)

5.3.2. Maximum cross section

Introducing a maximum cross section influences the number of members being present in the design and size of corresponding members. An example is given in Fig. 5.14 for load case 2 for a 4x4 grid and joint cost penalty of 1.0. In Fig. 5.14(a) no restrictions for maximum cross section are defined. In Fig. 5.14(b) and Fig. 5.14(c) maximum allowed cross sections are 1.0 and 0.6. Lower maximum cross sections results in an increase of number of members and decrease of average cross section.



(a) (b) (c)

Figure 5.14: Original member adding scheme with no maximum cross section (a), a maximum cross section of 1.0 (b) and maximum cross section of 0.6 (c)

5.3.3. Cross section penalty

Deviations of optimized and assigned cross sections result in inefficient and oversized structures. Cross section penalty's effectively avoid inefficient members to be present in the next iteration. However, as described

in ... the optimization can switch back and forth between solutions, an example is given in Fig. 5.15 for load case 2.

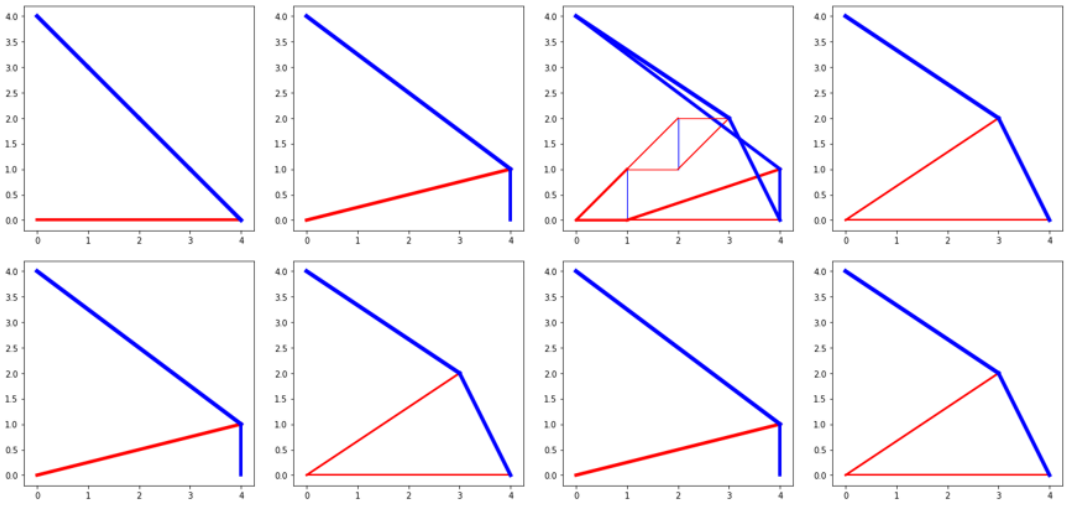
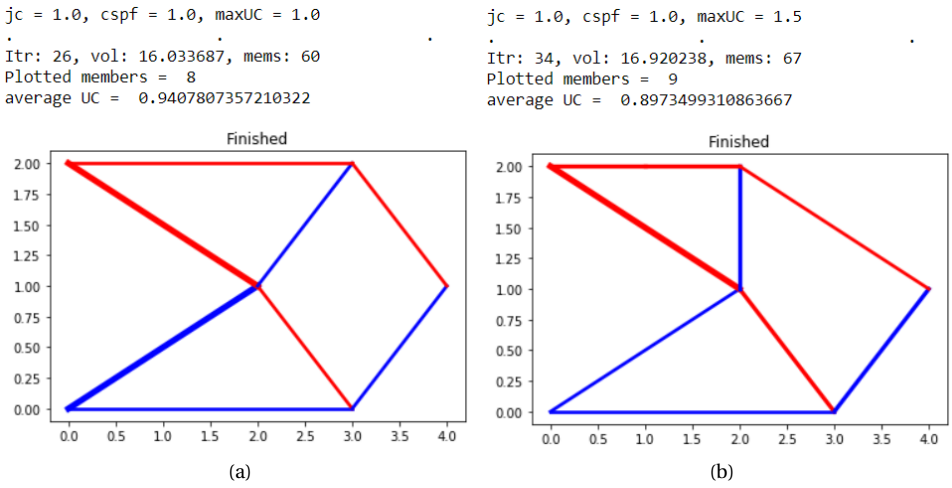


Figure 5.15: Adapted scheme with cross section penalty: Switching back and forth between solutions

5.3.4. Varying penalty's and maximum cross section

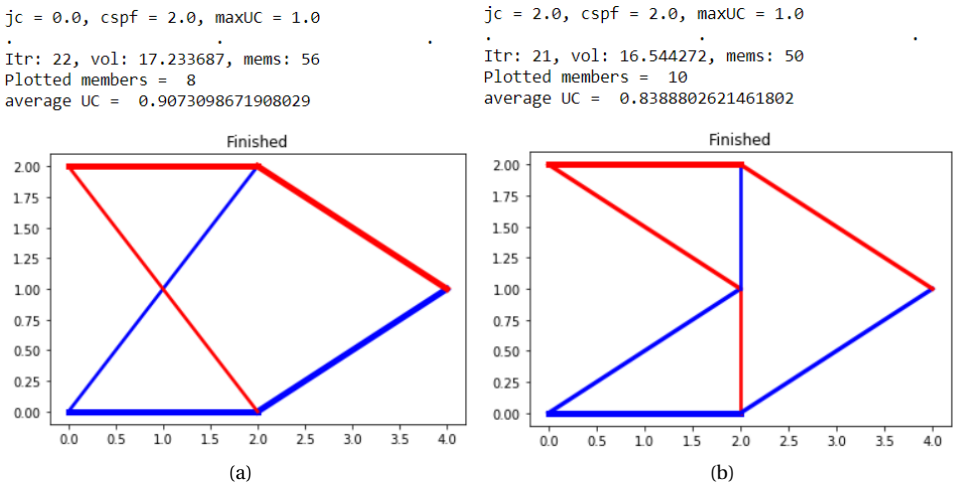
Cross section penalty, joint cost penalty and maximum unity check influence the optimization by penalizing inefficient members and forcing the optimization to create designs with less or more members. Different combinations of these parameters can result in different designs, some combinations have a positive influence on the quality of designs and other negative. One penalty can make the other penalty less effective resulting in a different design. In Fig. 5.16 and Fig. 5.17 examples are presented for a 4x2 grid for the adapted optimization scheme with capacity utilization, matrix method and possible stock member list, with a limited set of available lengths and cross sections. All cross sections can be assigned to all lengths. 3 Different designs were obtained.

In Fig. 5.16 a higher maximum unity check of 1.5 in Fig. 5.16(b) resulted in a different design than a maximum unity check of 1.0 in Fig. 5.16(a), different cross sections were assigned resulting in different penalty's for inefficient members resulting in a different final design. Varying the joint cost and cross section penalty resulted in two different designs in Fig. 5.17, the design without joint cost in Fig. 5.17(a) resulted in a more efficient design with higher average unity check.



(a) (b)

Figure 5.16: Example load case 1 with a maximum unity check of 1.0 (a) and 1.5(b)

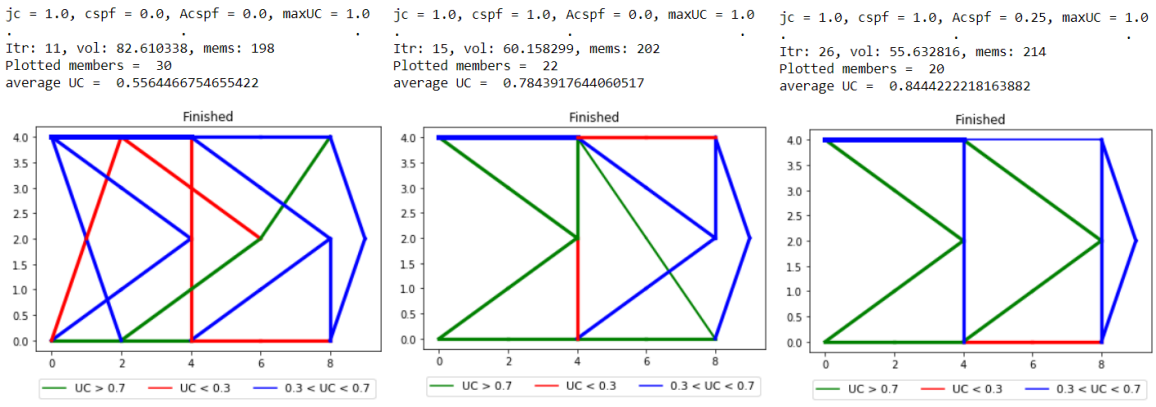


(a) (b)

Figure 5.17: Example load case 1 with cross section penalty without joint cost (a) and cross section penalty with joint cost (b)

5.3.5. Accumulating cross section penalty

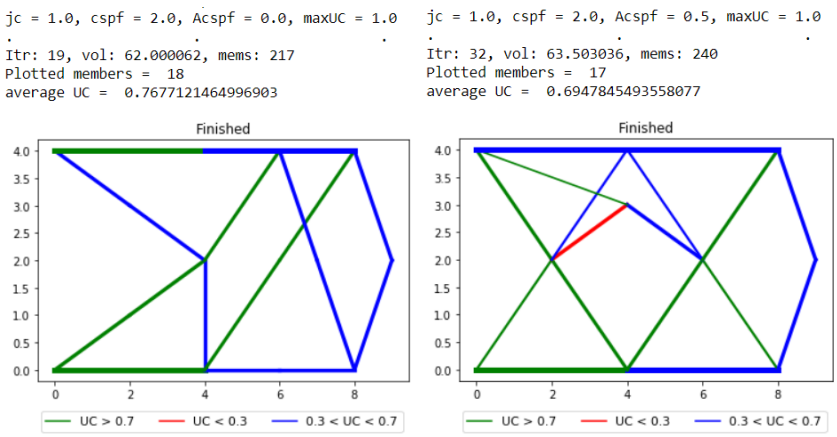
Stocks of reclaimed items defined with elements with individual length and cross section make the optimization more complex. Elements with a specific length and cross section should be located and assigned as efficient as possible. Cross section penalty's avoid members to be present in the final design, but can not avoid switching back and forth of solutions and exert often too little influence on the optimization. Accumulating cross section penalty's can avoid switching back and forth an penalize very inefficient members harder avoiding them to be present in further iterations. An example of the influence of both penalty's is given in Fig. 5.18. The design without penalty's given in Fig. 5.18(a) performs worst with the largest volume and lowest average unity check. Introducing a cross section penalty in Fig. 5.18(b) increases efficiency significantly, applying an accumulating cross section penalty in Fig. 5.18(c) could decrease volume and increase average unity check even more.



(a) (b) (c)

Figure 5.18: Example of influence of csp and Acsp for a 9x4 grid for load case 1

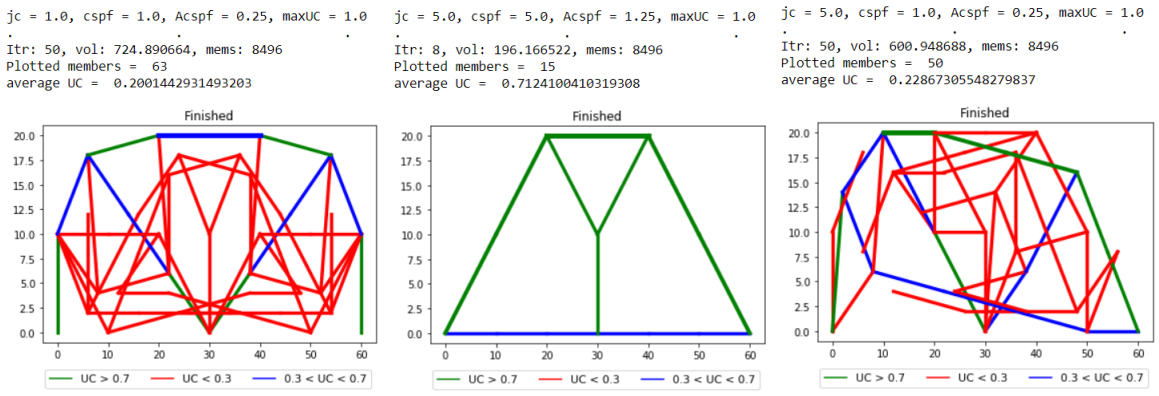
Varying the value of penalty's can result in different designs, penalty's influence the effectiveness of each other making it difficult to declare which combination of penalty's works best. An example is given in Fig. 5.19 for the same optimization problem of Fig. 5.18, different values for the cspf and Acspf resulted in different designs, the design without Acsp of Fig. 5.19(a) is more efficient in terms of volume and average unity check than the design with Acsp of Fig. 5.19(b).



(a) (b)

Figure 5.19: Example of varying values of the cspf and Acspf to obtain different designs for a 9x4 grid for load case 1

The effectiveness of penalty's depend on the value of the penalty factor relative to lengths of available elements from stock. A penalty with a value of 1.0 added to elements with a length of 1m and 100m would result in an increase of volume of respectively 100% and 1%, the volume share for the element of 100m increases only slightly, the penalty will be less effective. A higher penalty is required when only large elements are available. Values should not be too high, often 'soft' penalty's result in the most convenient designs. It should be possible for members to be included in the design if, despite having a penalty, these members contribute to a design with less volume.
In Fig. 5.20 an example is given for a 60x20 design grid for load case 3, only elements with a length above 10m are available. In Fig. 5.20(a) low penalty factors were not effective enough, after 50 iterations still no result could be obtained. Only increasing joint cost in Fig. 5.20(c) was effective neither, increasing joint cost and cross section penalty factors in Fig. 5.20(b) resulted in an efficient design with only few members.



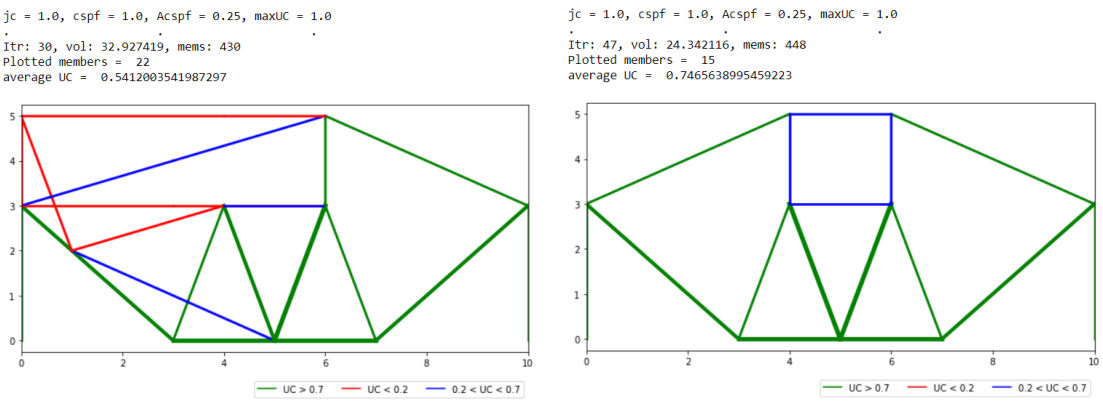
(a) (b) (c)

Figure 5.20: Example of influence of larger penalty's for larger elements for a 60x20 grid for load case 3

5.4. Extra conditions

The optimization with the original member adding scheme of (He et al., 2019) stops when no violation of strain in the PSML occurs. In Fig. 5.21(a) a truss is optimized for load case 3, after 30 iterations a design is found verified for virtual strain. However, in Fig. 5.21(b) a much more efficient design is found with the implementation of extra conditions for minimum average unity check. Another extra 17 iterations were required to generate the design of in Fig. 5.21(b) with an average unity check above threshold number minCU. Colors for the members in the plot are determined with threshold numbers minCU and minFinalCU. Members in red with an unity check below threshold number minFinalCU receive an accumulating and standard cross

section penalty, members in blue with an unity check below threshold number minCU only receive a cross section penalty.

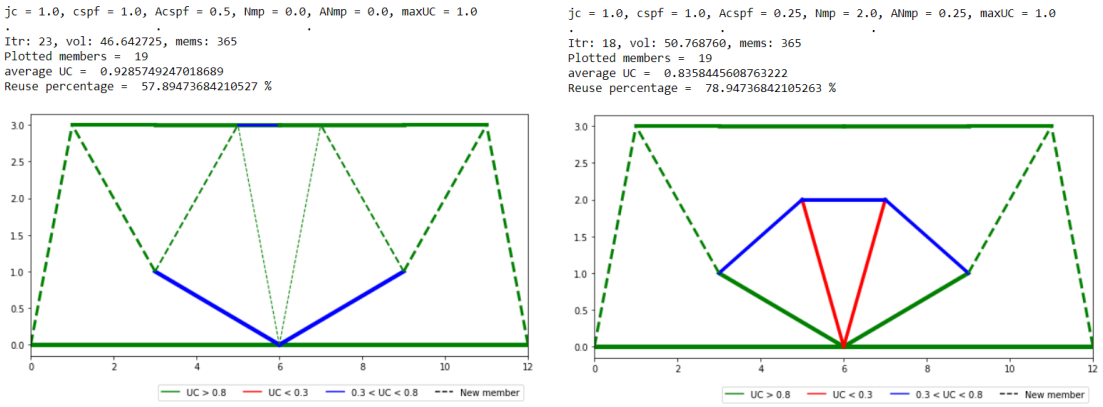


(a) (b)

Figure 5.21: Example of adapted optimization scheme without (a) and with (b) extra conditions for the optimization to stop

5.5. Limited availability

The method of assigning available cross sections, taking availability of elements into account, as efficient as possible, described in Section 4.7.2, allows for new elements to be in the structure besides reused elements. A new member and accumulating new member penalty is introduced to penalize new members, an example of the effectiveness of this penalty is given in Fig. 5.22. Without new member penalty's an efficient design was obtained in Fig. 5.22(a) but only 58% of the elements are reused. With new member penalty's the reuse percentage could be increased to 79% in Fig. 5.22(b). The average unity check dropped from 0.93 to 0.83. However, in Fig. 5.22(a) the average unit check is raised significantly due to the new members with an unity check of 1.0.



(a) (b)

Figure 5.22: Example of adapted optimization scheme without (a) and with (b) new member penalty

Similar as with the cross section penalty described in Section 4.5.3, introducing only a new member penalty and no accumulating new member penalty could result in switching back and forth between solutions. An example is given in Fig. 5.23, no members are present with a unity check below minFinalCU thus no members receive an accumulating cross section penalty. Only cross section and new member penalty's are applied and the optimization switches back and forth between two solutions from iteration 20 to iteration 24.

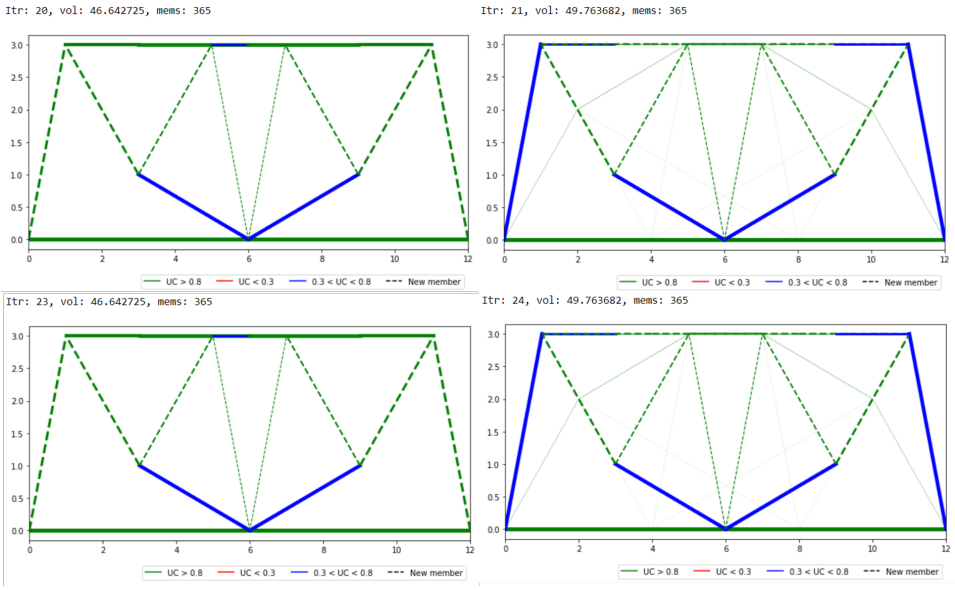
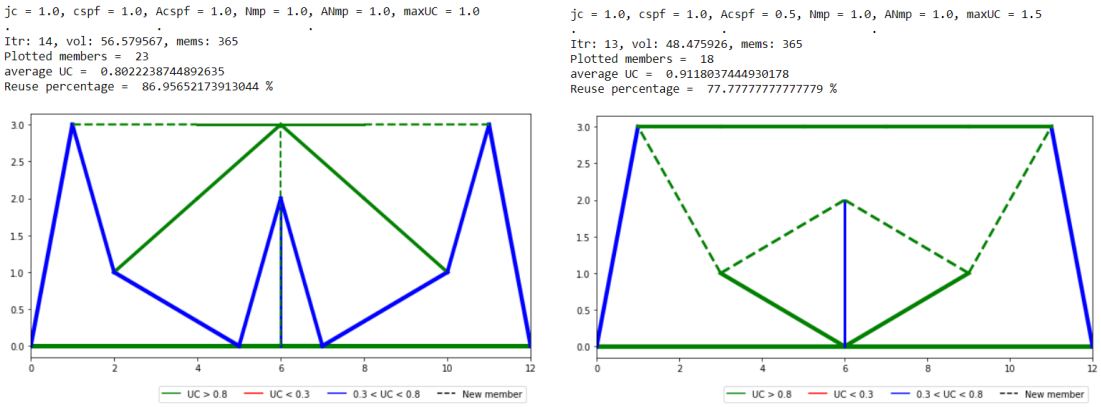


Figure 5.23: Switching back and forth between solutions with a low percentage of reuse

5.5.1. Cross section penalty's for new members

Distinction between potential efficient and potential inefficient new member is made by applying cross section penalty's to new members based on their initial assigned cross section from stock. In Fig. 5.24 the same example as in Fig. 5.22 is optimized now including cross section penalty's for new members. Two different designs could be obtained, Fig. 5.24(a) with a higher percentage of reuse and Fig. 5.24(b) with lower volume.



(a) (b)

Figure 5.24: Example of adapted optimization scheme with new member penalty and cross section penalty's for new members

5.6. Guidelines

The optimization is sensitive for adjustments of all input parameters. In some situations high values for penalty's and few members included in the initial connectivity have the best influence on the optimization and in other situations low values for penalty's and many members included in the initial connectivity. Values of penalty's increase or reduce the effect of other penalty's. What the best combination of all input parameters is depends on the load case and available elements from stock. A guideline for the optimization, value of threshold numbers, penalty's and iterations from which penalty's should be applied is provided in Section 5.6.1 and Section 5.6.2 to perform a successful optimization. Keep in mind that these are guidelines and no guarantee for convenient results, efficiency of possible designs mainly depends on the available elements from stock and design problem.

5.6.1. Optimization guidelines

The goal of every optimization is to design trusses with the lowest volume, highest percentage of reuse and highest average unity check as possible. It is also preferred to obtain designs with as few elements as possible to provide designs with easier constructibility. One should start with high threshold numbers for minCU and minReuse and gradually decrease these numbers if no convenient results can be obtained. A guideline is provided below to perform the optimization, an overview is presented in Fig. 5.25.

1. Evaluate the stock with available elements and the number of members in the PSML. If only few members are present in the PSML, the cut off length L_{cut} could be increased and large elements from stock could be cut into multiple sections. Number of elements could also be influenced by applying scaling factors.
2. Determine the initial connectivity. Start the optimization with only short elements present in the initial connectivity to reduce computation times. A constraint for the initial connectivity is that a stable structure must be formed, otherwise the optimization will be unable to run.
3. Determine threshold numbers for minCU, minFinalCU and minReuse. Aim for an efficient design with 100% reuse. Start with values of 0.90 for minCU, 0.40 for minFinalCU and 1.0 for minReuse. maxUC should be set on 1.0 or 2.0.
4. Determine iterations when penalty's are applied. These iterations should be determined based on the initial connectivity. Guidelines are presented in Table 5.1.
5. Determine values for all penalty's. These penalty should be determined based on the average length of available elements from stock. Guidelines are presented in Table 5.2
6. Optimize and try to find any efficient and convenient results.
7. No solutions found? Adapt the initial connectivity and iterations when penalty's are applied. When no solution is found either, lower threshold numbers minCU to 0.80 or minReuse to 0.90. Lower the threshold numbers until convenient results are found. Often including all elements in the initial connectivity results in the most efficient designs.

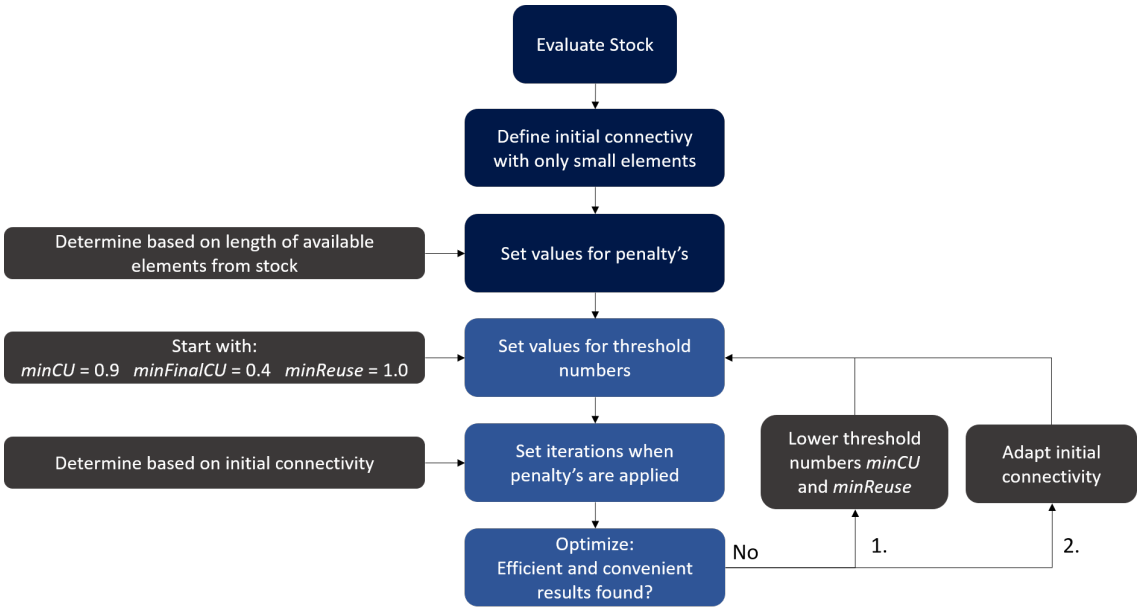


Figure 5.25: Optimization guideline for finding the best combination of initial connectivity, threshold numbers and iterations when penalty's are applied.

5.6.2. Penalty guidelines

Values of penalty's

Penalty's are added to the length of an element node during the optimization with the dual interior point method. Extra 'virtual' length increases volume share avoiding an element to be present in the next and or further iterations. The influence of a penalty depends on the length of an element. A penalty of 1.0 result in a volume increase of 1% for a member with a length of 100.0 and a volume increase of 100% for a member with a length of 1.0. Therefore, value of penalty's should be determined based on the average length of element available from stock. Guidelines are provided in Table 5.2.
Values of penalty's should not be too high, penalty's should be 'soft' penalty's allowing penalized elements to be present in further iterations. This gives the opportunity for a few inefficient members to be present in the design if with these members included, despite having a penalty, designs with much less volume can be obtained. Besides, in different layouts members could actually be efficient.

Iterations from which penalty's are applied

Different designs are obtained when penalty's are applied from different iterations. Most importantly are iterations from which accumulating penalty's are applied. It should be avoided that members are penalized in early iterations while they could actually be very efficient in later iterations. Such situations mainly occur in early iterations with a limited set of elements included in the initial connectivity. A limited number of solutions is possible and designs can be inefficient. In later iterations when more elements are added to the initial connectivity, an increased number of solutions is possible. Elements which were inefficient in early iterations could actually be efficient in later iterations in structures with different layouts. Therefore, the iterations from which penalty's are applied should be determined based on the initial connectivity. Guidelines are provided in Table 5.1. When all elements are included in the initial connectivity, accumulating penalty's could be applied early.

If initial connectivity includes few elements	
Penalty	Apply from iteration:
csp	1
Acsp	15
csp new elements	1
Acsp new elements	15
Acsp for UC >1.0	25
Nmp	10
ANmp	25

If initial connectivity includes all elements	
Penalty	Apply from iteration:
csp	1
Acsp	3
csp new elements	1
Acsp new elements	3
Acsp for UC >1.0	10
Nmp	3
ANmp	10

Table 5.1: Guidelines for iteration from which penalty's should be applied

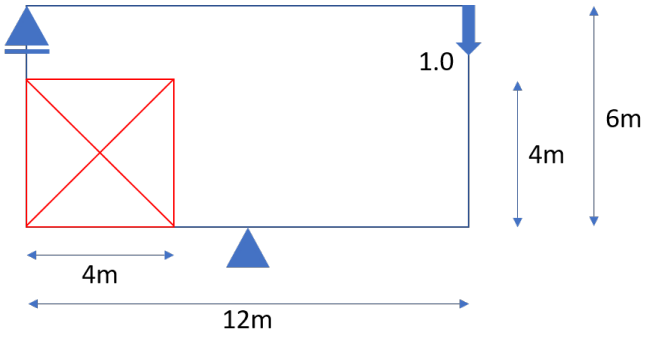
If 0m $<L_{stock}<$10m	
Penalty	Value
jc	1.0 - 3.0
cspf	1.0 - 3.0
Acspf	0.25 - 1.0
Nmp	1.0 - 3.0
ANmp	0.25 - 1.0

If 10m $<L_{stock}<$20m	
Penalty	Value
jc	3.0 - 6.0
cspf	3.0 - 6.0
Acspf	1.0 - 2.0
Nmp	3.0 - 6.0
ANmp	1.0 - 2.0

Table 5.2: Guidelines for values of penalty's

5.6.3. Keep iterating

When an optimized structure meets all requirements for virtual strain, efficiency and reuse percentage, the design is shown and plotted in Python. However, often the design could be improved by performing more iterations and applying more penalty's. A maximum number of iterations can be specified after which the optimization stops. All designs meeting all requirements are displayed. An example is given for the optimization problem and stock of Fig. 5.26. No elements are allowed to be present in the red box. Maximum tensile and compressions stresses are respectively 1.0 and 0.7.



L	A	n
1.5	0.4	6
1.2	0.5	6
1.8	0.6	8
2.5	0.7	6
2	0.8	6
3	1.2	6
2.8	1.7	10
4.2	2	8

Figure 5.26: Optimization problem: Combination of truss and cantilever

For the combination of value's for penalty's in Fig. 5.27 and Fig. 5.28 the first design meeting all requirements for efficiency and reuse percentage is obtained after iteration 79 with an average unity check for all reused elements of 0.77 with a reuse percentage of 83% shown in Fig. 5.27. The optimization continued until iteration 200 and 13 different designs were found. The final one, most efficient one and the one with highest reuse percentage is obtained after iteration 176 shown in Fig. 5.28. The average unity check could be increased to 0.84 and the reuse percentage to 91%, a volume decrease of 9.2 was achieved compared to the design of Fig. 5.27.

```
jc = 5.0, cspf = 3.0, Acspf = 0.5, Nmp = 3.0, ANmp = 1.0, maxUC = 1.0
.                    .                    .
Itr: 79, vol: 99.574647, mems: 958
Plotted members =  24
average UC =  0.8065103577529635
average UC reused members =  0.7678124293035562
maximum unity check =  0.9999999999974181
minimum unity check =  0.34416980744640185
Reuse percentage =  83.33333333333334 %
```

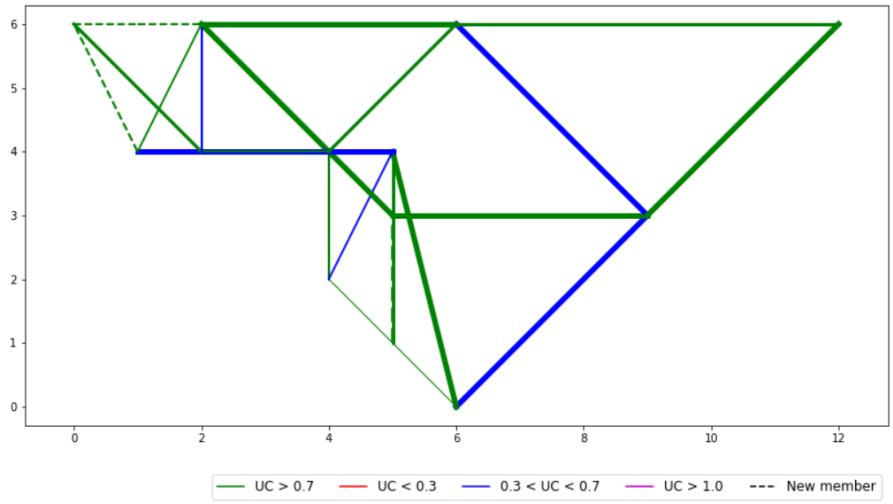


Figure 5.27: Combination of truss and cantilever design 1

```
jc = 5.0, cspf = 3.0, Acspf = 0.5, Nmp = 3.0, ANmp = 1.0, maxUC = 1.0
.                    .                          .
Itr: 176, vol: 90.321799, mems: 958
Plotted members =  22
average UC =  0.8581910153937471
average UC reused members =  0.8440101169331218
maximum unity check =  0.9999999999969827
minimum unity check =  0.5824308845275253
Reuse percentage =  90.9090909090909 %
```

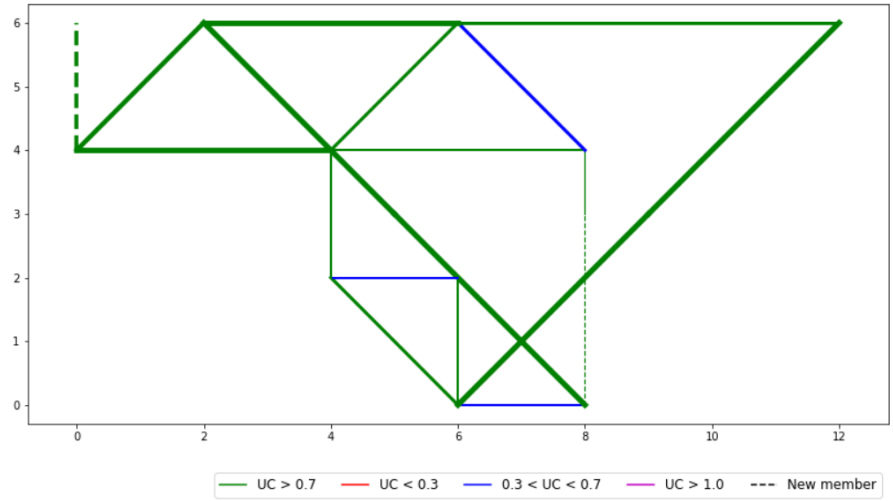


Figure 5.28: Combination of truss and cantilever design 2

5.7. Element assignment with and without penalty system

In Section 3.5 an existing method of Brütting et al. (2019b) for designing with reused elements was described. The optimization consists a number of steps. Topology is optimized with assignment of elements from stock after which a geometry optimization is performed. The method of Brütting et al. (2019b) is used as inspiration for the optimization tool developed in this paper. Element assignment is extend with a penalty system to increase efficiency. Geometry optimization is excluded.

In this section two examples are provided to compare the optimization method of Brütting et al. (2019b) with the developed optimization method in this paper. The method of Brütting et al. (2019b) is approached by performing only 1 iteration with element assignment with the optimization tool, no geometry optimization is performed. The obtained design is compared with designs obtained after performing multiple iterations with multiple combinations of parameters. In summary, optimization are performed with and without penalty system.
Example 1 contains a database with a limited number of different elements but large number of availability of individual elements. Example 1 is the same example as example 3 from Section 4.8. Example 2 contains a database with a larger number of different elements but lower availability of individual elements. Loading- and boundary conditions are provided in Fig. 5.29. In example 2, supports are located 2m from the bottom of the design domain.

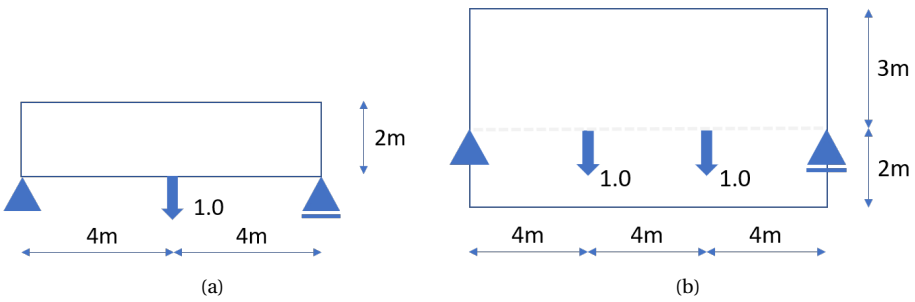


Figure 5.29: Loading- and boundary conditions for example 1 (a) and example 2 (b)

Example 1

```
Itr: 1, vol: 24.953310, mems: 74
Plotted members =  36
average UC =  0.8848911643025729
average UC reused members =  0.8848911643025729
Reuse percentage =  100.0 %
```

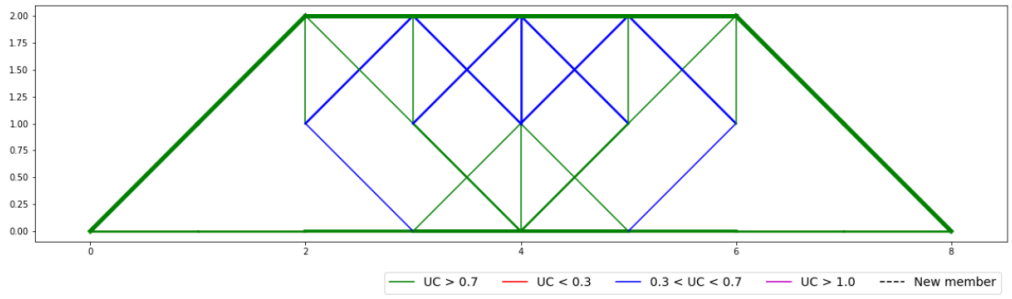


Figure 5.30: Example 1: Design 1 obtained with approached method of Brütting et al. (2019b) for loading- and boundary conditions of Fig. 5.29(a)

```
jc = 1.0, cspf = 1.0, Acspf = 0.35, Nmp = 2.0, ANmp = 1.0, maxUC = 2.0
.                    .                    .
Itr: 16, vol: 23.425556, mems: 74
Plotted members =  32
average UC =  0.9465386549424488
average UC reused members =  0.9465386549424488
Reuse percentage =  100.0 %
```

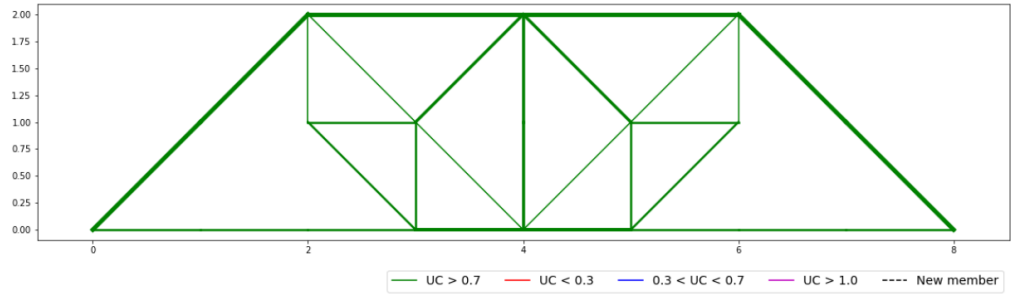


Figure 5.31: Example 1: Design 2 obtained with developed method with penalty system for loading- and boundary conditions of Fig. 5.29(a)

```
jc = 1.0, cspf = 1.0, Acspf = 1.0, Nmp = 1.0, ANmp = 0.25, maxUC = 1.0
.                    .                    .
Itr: 42, vol: 24.673559, mems: 74
Plotted members =  26
average UC =  0.9785070213346531
average UC reused members =  0.9785070213346531
maximum unity check =  0.9999999999979675
minimum unity check =  0.8988395870796176
Reuse percentage =  100.0 %
```

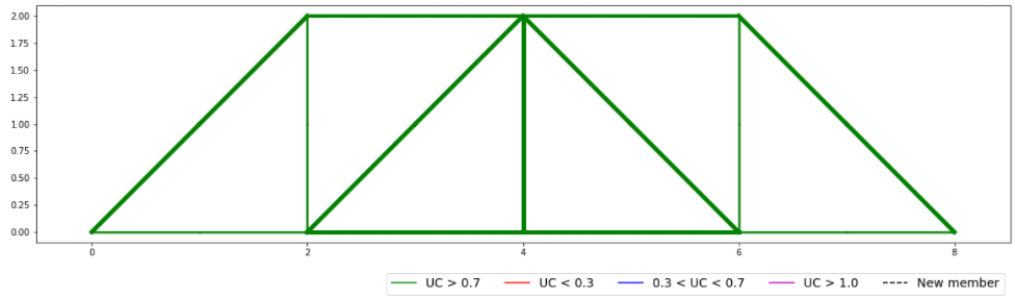


Figure 5.32: Example 1: Design 3 obtained with developed method with penalty system for loading- and boundary conditions of Fig. 5.29(a)

Example 2

```
Itr: 1, vol: 60.492443, mems: 635
Plotted members =  25
average UC =  0.6305164323258636
average UC reused members =  0.5381455404073294
Reuse percentage =  80.0 %
```

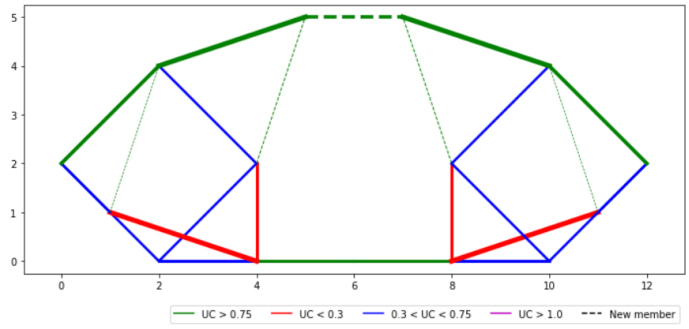


Figure 5.33: Example 2: Design 1 obtained with approached method of Brütting et al. (2019b) for loading- and boundary conditions of Fig. 5.29(b)

```
jc = 1.0, cspf = 1.0, Acspf = 0.25, Nmp = 1.0, ANmp = 0.25, maxUC = 1.0
.                  .                  .
Itr: 32, vol: 50.327976, mems: 635
Plotted members =  22
average UC =  0.7815385911781035
average UC reused members =  0.7596924502959139
Reuse percentage =  90.9090909090909 %
```

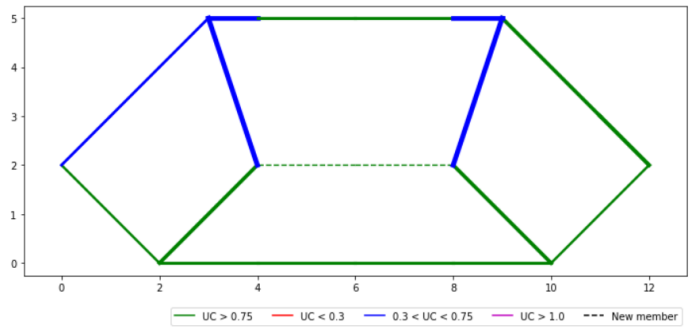


Figure 5.34: Example 2: Design 2 obtained with developed method with penalty system for loading- and boundary conditions of Fig. 5.29(b)

```
jc = 1.0, cspf = 3.0, Acspf = 0.25, Nmp = 1.0, ANmp = 0.5, maxUC = 1.0
.                  .                  .
Itr: 48, vol: 54.542949, mems: 635
Plotted members =  18
average UC =  0.7852490505973245
average UC reused members =  0.7852490505973245
Reuse percentage =  100.0 %
```

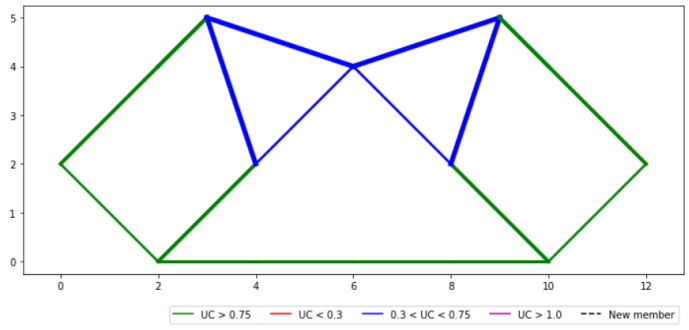


Figure 5.35: Example 2: Design 3 oobtained with developed method with penalty system for loading- and boundary conditions of Fig. 5.29(b)

5.7.1. Analysis and conclusion

One-time optimization with element assignment in example 1 in Fig. 5.30 results in an efficient design with an average unity check of 0.88 with 100% reuse. However, by performing multiple iterations and applying penalty's to inefficient and new members designs could be improved. In Fig. 5.31 in design 2 volume and number of elements could be reduced and average unit check could be increased. In Fig. 5.32 in design 3 number of members could be increased even further but at cost of a small volume increase compared to design 1.

With limited availability of elements from stock in example 2, the influence of the penalty system was much higher. The optimization without penalty system in design 1 in Fig. 5.33 results in an inefficient design with an average unity check of 0.54 with 80% reuse. With penalty system, volume could be decreased by 17% in design 2 in Fig. 5.34 with 90% reuse in a design with 3 less members, an average unity check of 0.79 was obtained. In Fig. 5.35 in design 3, a reuse percentage of 100% could be obtained with a volume reduction of 10% compared to design 1 with 80% reuse. An overview of the results is provided in Fig. 5.36.

The two step optimization approach of Brütting et al. (2019b) has been adapted, extended and improved. In the iterating process multiple designs can be generated in which members can disappear and reappear from the design in further iterations. With limited availability, element assignment can be combined with the implementation of new elements in the design. In example 2, reuse percentage could be increased from 80 to 90 and 100% by applying penalty's to new elements. In example 1 as well as example 2, the penalty system effectively penalized inefficient elements resulting in designs with higher efficiency and lower volume.

Optimization problem 1

Design	Volume	Members	Reuse %	Average UC	Average UC Reused Elements
1	25	36	100	0.88	0.88
2	23	32	100	0.95	0.95
3	23	26	100	0.98	0.98

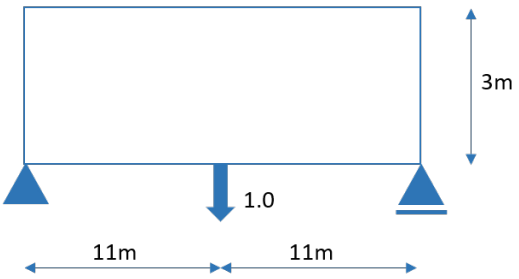
Optimization problem 2

Design	Volume	Members	Reuse %	Average UC	Average UC Reused Elements
1	60	25	80	0.63	0.54
2	50	22	91	0.78	0.76
3	55	18	100	0.78	0.78

Figure 5.36: Element assignment with and without penalty system: Overview results example 1 and 2

5.8. Example 22m Truss

In this section an example is provided to show and test the capabilities of the developed optimization tool. A truss is optimized in a design grid of 22x3 shown in Fig. 5.37. In Section 5.8.1 the truss is optimized with the original member adding scheme of He et al. (2019) with and without joint cost. In Section 5.8.1 the truss is optimized with reused elements. A stock with 8 different elements has been defined, with cross sections varying from 0.4 to 0.8. In Section 5.8.3 a standard truss has been designed in SCIA. A force of 1.0 is applied with maximum tension and compression stresses of respectively 1.0 and 0.7. Conclusions on this example are provided in Section 5.8.4.



L	A	n
4.0	0.6	4
4.5	0.7	8
3.5	0.4	4
4.8	0.7	12
3.8	0.6	10
6.0	0.8	8
3.2	0.5	4
5.2	0.8	12

Figure 5.37: Optimization problem 22m truss

5.8.1. Optimization with original member adding scheme

In Fig. 5.38, Fig. 5.39 and Fig. 5.40 the truss is optimized with the original member adding scheme of He et al. (2019) without joint cost, joint cost of 1.0 and joint cost of 3.0. All possible members are included in the possible member without any restrictions for the optimization of cross sections. Volume and number of members is given for all structures. Members in compression are plotted in red, members in tension in blue.

Original Member Adding Scheme

Volume	Members
73	102

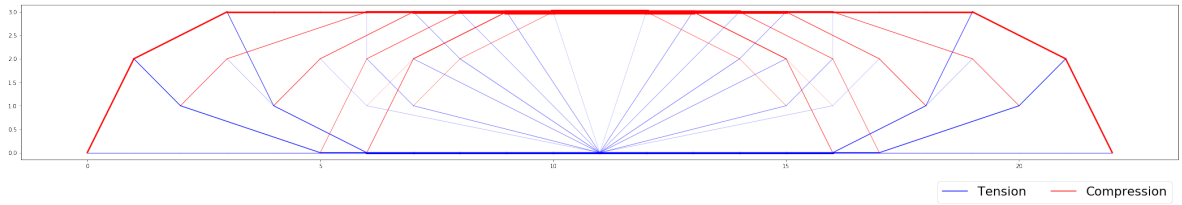


Figure 5.38: Optimized truss with original member adding scheme

Original Member Adding Scheme with joint cost penalty of 1.0

Volume	Members
76	25

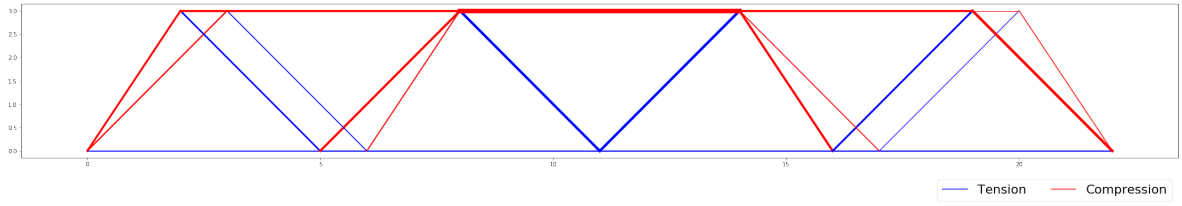


Figure 5.39: Optimized truss with original member adding scheme with joint cost of 1.0

Original Member Adding Scheme with joint cost penalty of 3.0

Volume	Members
81	6

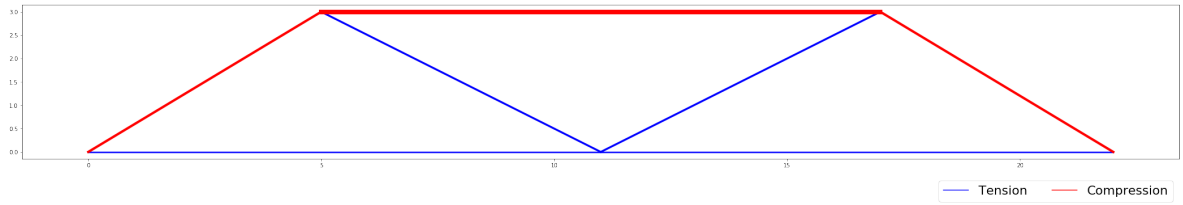


Figure 5.40: Optimized truss with original member adding scheme with joint cost of 3.0

5.8.2. Optimization with reused elements

In Fig. 5.41 till Fig. 5.45 5 different optimized trusses with reused elements are presented. Only elements from stock with corresponding cross sections are included in the possible stock member list. Volume, number of members, percentage of reuse and average unity check are given. Intermediate iterations of design 5 can be found in Appendix B.

Design 1

Volume	Number of Members	Reuse Percentage	Average UC	Average UC Reused elements
106	32	97	0.73	0.73

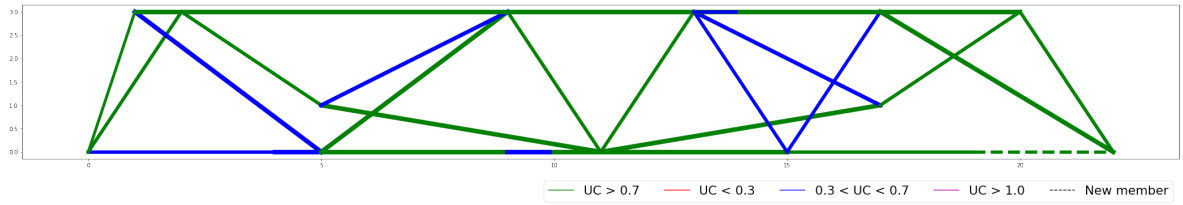


Figure 5.41: Optimized truss with reused elements 1

Design 2

Volume	Number of Members	Reuse Percentage	Average UC	Average UC Reused elements
98	31	94	0.79	0.77

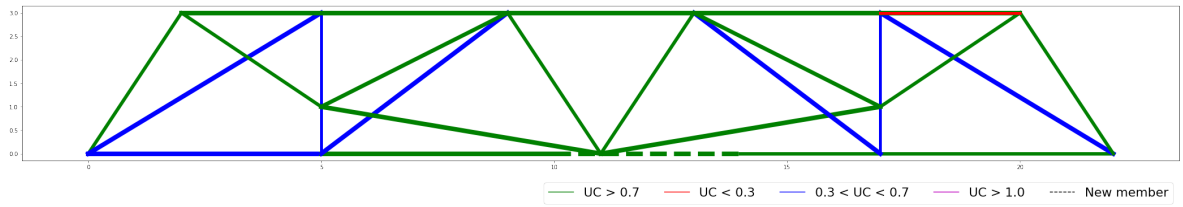


Figure 5.42: Optimized truss with reused elements 2

Design 3

Volume	Number of Members	Reuse Percentage	Average UC	Average UC Reused elements
106	33	91	0.75	0.72

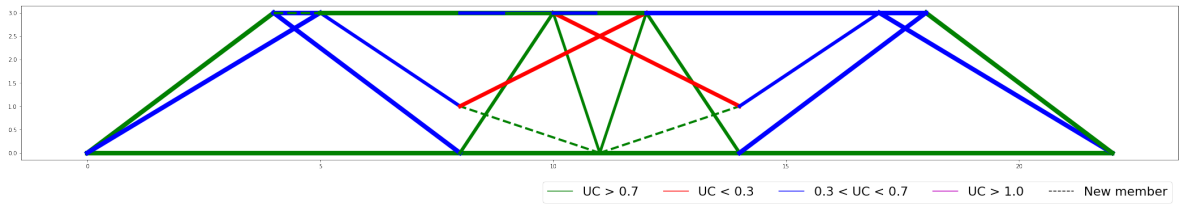


Figure 5.43: Optimized truss with reused elements 3

Design 4

Volume	Number of Members	Reuse Percentage	Average UC	Average UC Reused elements
94	29	90	0.79	0.77

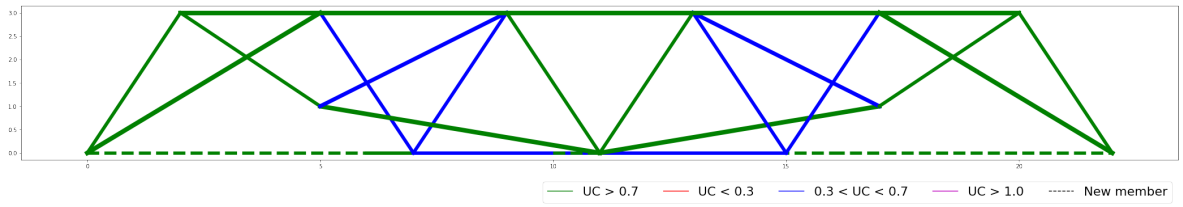


Figure 5.44: Optimized truss with reused elements 4

Design 5

Volume	Number of Members	Reuse Percentage	Average UC	Average UC Reused elements
94	29	76	0.81	0.75

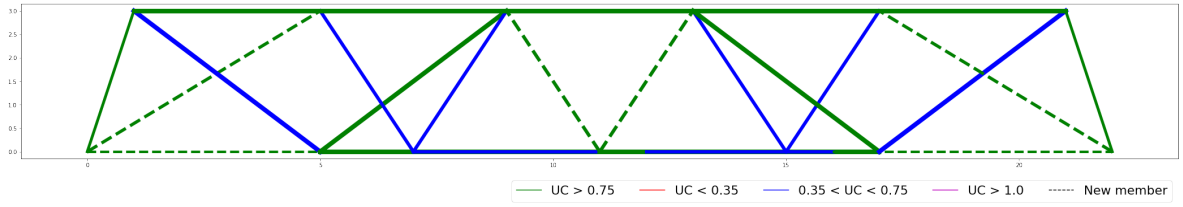


Figure 5.45: Optimized truss with reused elements 5

5.8.3. Design with SCIA

To compare the optimized trusses with a traditional truss, a design is made in SCIA shown in Fig. 5.46. The truss is verified for deflections and maximum tensile and compression stresses, maximum compression stresses are 0.7 times the maximum tension tresses similar to the examples in Section 5.8.1 and Section 5.8.2.

Volume	Number of Members	Average UC
79	14	0.96

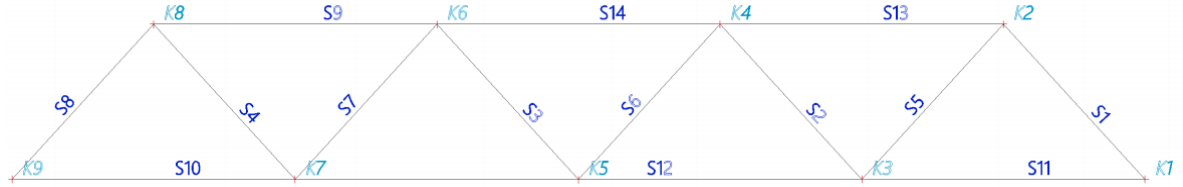


Figure 5.46: Truss designed in SCIA

5.8.4. Analysis and conclusions

To compare the optimized trusses with the original member adding scheme, standard design in SCIA and optimized trusses with reused elements, an overview is given in Fig. 5.47. The truss optimized with the original member adding scheme without joint cost has a volume reduction of 6.6% compared to the standard design in SCIA. With denser grids, higher volume reductions may be obtained. Simplified designs with a joint cost of 1.0 and 3.0 resulted in respectively a volume reduction of 3.1% and volume increase of 3.0% compared to the design in SCIA.

The optimized designs with reused elements have a volume increase of 20 to 30% compared to the standard design. However, high reuse percentages could be obtained. Designs 4 contains 90% reused elements with an average unity check of 0.77 in which 29 of available 62 available elements are used. Volume increase could be reduced to 17% compared to the standard design in SCIA.

The designs with reused elements contain 29 to 33 members compared to 14 members in the standard design. However, efficiency and number of elements depends on the available elements from stock. The stock contains elements with a maximum length of 6m and maximum cross section of 0.8, many elements are required to obtain equilibrium without violating maximum tensile and compression stresses. Upper and lower beams contain overlapping elements. Overlap in the upper beam can clearly be seen in design 2 in Fig. 5.42 and design 3 in Fig. 5.43.

In conclusion, the optimization was effective and many different designs were generated. With high reuse percentages and small volume increase, it could be possible to obtain reductions in costs and carbon emissions, however, studies are required to take deconstruction, construction and transport of reclaimed elements into account.

Optimized Designs Original Member Adding Scheme

Design	Volume	Members			
Jc = 0.0	73	102			
Jc = 1.0	76	25			
Jc = 3.0	81	6			

Optimized Designs Member Adding Scheme with Reused Elements

Design	Volume	Members	Reuse %	Average UC	Average UC Reused Elements
1	106	32	97	0.73	0.73
2	98	31	94	0.79	0.77
3	106	33	91	0.75	0.72
4	94	29	90	0.79	0.77
5	94	29	76	0.81	0.75

Standard Design with SCIA

Design	Volume	Members	Reuse%	Average UC	
SCIA	79	14	-	0.96	

Figure 5.47: Overview optimized and standard designs with and without reused elements

6

Case Study

In this chapter the new steel roof structure of Ahoy is redesigned with the developed optimization tool, the elements of the steel structure of Provinciehuis Zuid-Holland are used as a database. Information about loading- and boundary conditions of Ahoy and steel elements of Provinciehuis Zuid-Holland are obtained via personal contact with Lagendijk and Slui (2021).

6.1. Optimization problem: Ahoy

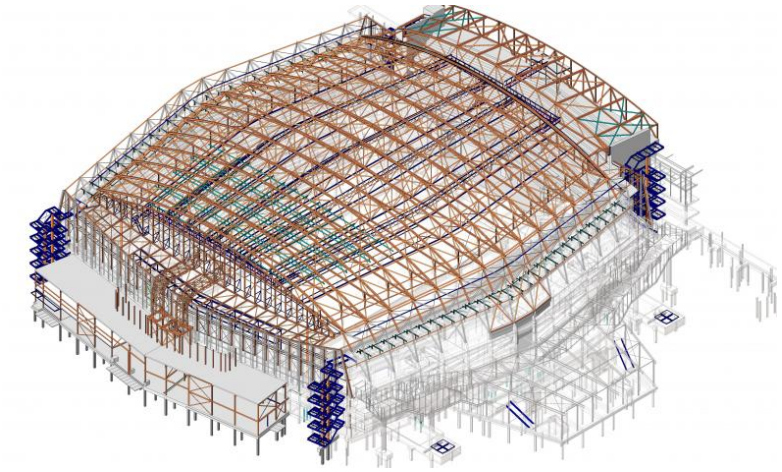


Figure 6.1: New roof structure Ahoy Stadium

In 2010 the existing roof structure of Ahoy was replaced by a new structure to increase capacity of the stadium from 12.000 to 15.000 spectators. The new roof was mounted on top of the existing roof after the existing roof was removed. The roof contains a total of 14 trusses with a length varying from 72m to 82.5m with corresponding height varying from 5.4m to 6.4m. The two outermost trusses, with length of 72m are redesigned in this case study. The trusses are designed for a rigging load of 200 KN, a schematic overview is given in Fig. 6.2.

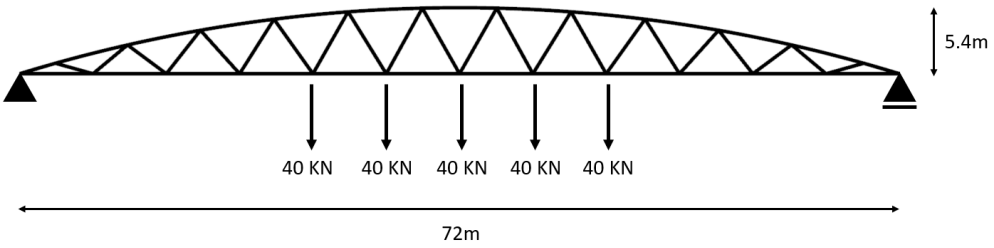
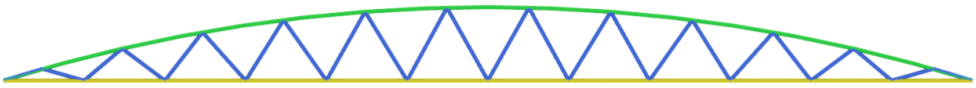


Figure 6.2: Rigging load outermost trusses Ahoy Stadium

The two outermost trusses contain 3 different types of profiles. The lower beam HE300B, the upper beam HE400B and the diagonals CHS139.7X5.0. An overview is given in Fig. 6.3 with properties of profiles, self-weight of the structure in KN/m and volume of the complete truss.



Element(s)	Profile	Area (mm2)	Weight (Kg/m)	Total Length (m)	Total weight (KN/m)	Total Volume (m3)*1e-3
	CHS 139.7X5.0	2116	16.6	110	1.57	232.8
	HE400B	14908	158	73	1.17	1088.3
	HE300B	19778	119	72	0.25	1424.0
					+= 1.86	+ = 2745.1

Figure 6.3: Steel profiles outermost trusses Ahoy Stadium

For the optimization problem, loading conditions are based on the rigging load, self-weight of the original structure, permanent load and snow loading. Loading factors for permanent and variable loads are taken into account.

Distributed loads				
Load	Number	Unit	Load factor	Number ∗ Load factor
Permanent	7.5	KN/m	1.2	9.0
Snow	3.92	KN/m	1.5	5.88
Self-weight	1.86	KN/m	1.2	2.23
		KN/m	Sum	17.11

Table 6.1: Distributed loads optimization problem Ahoy

Point Loads				
Load	Number	Unit	Load factor	Number ∗ Load factor
Rigging	40	KN/m	1.5	60.0

Table 6.2: Point loads optimization problem Ahoy

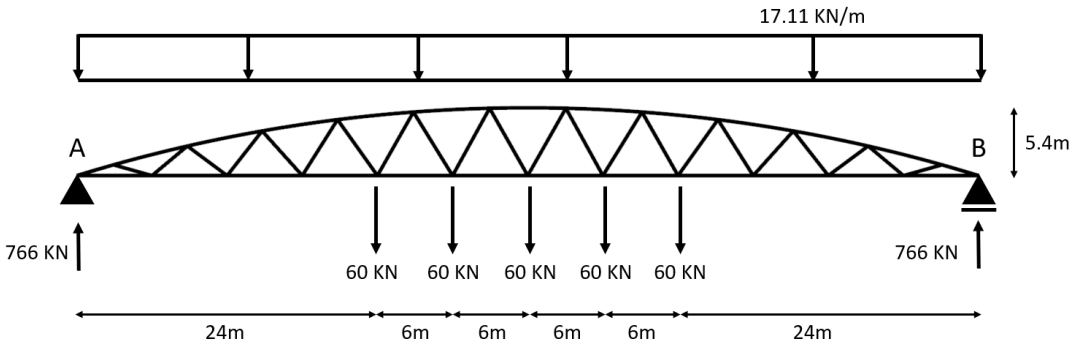


Figure 6.4: Loading conditions Ahoy

Distributed loads can not be incorporated in the optimization. Therefore, the value of the point loads is increased taking the distributed load into account. A total of 5 point loads calculated with Eq. (6.1) are applied at distance of 12m, an overview is given in Fig. 6.5. The optimized structure will only contain normal forces and no bending moments, at every point load vertical and horizontal force equilibrium must be established.

Adding more point loads to the structure will make the optimization problem too complex. The placement of the point loads, without taking support reactions into account, result in a similar bending moment around A. Bending moment at A due to the original loading conditions of Fig. 6.4 is calculated with Eq. (6.2), bending moment at A due to the loading conditions for the optimization is calculated with Eq. (6.3). With support reactions, the bending moment at A will be zero.

Point loads for the loading conditions in Fig. 6.5:

$$\begin{aligned} \sum F_{vertical} &= 5 * 60 + 17.11 * 72 = 1532\text{KN} \\ \text{Point load} &= \frac{\sum F_{vertical}}{5} = 306.4\text{KN} \end{aligned} \tag{6.1}$$

Moment at A due to the loading conditions of Fig. 6.4:

$$\sum M_A = 60 * (24 + 30 + 36 + 42 + 48) + 17.11 * 72 * \frac{72}{2} = 55149.1\text{KNm} \tag{6.2}$$

Moment at A due to the loading conditions of Fig. 6.5:

$$\sum M_A = 306.4 * (12 + 24 + 36 + 48 + 60) = 55152\text{KNm} \tag{6.3}$$

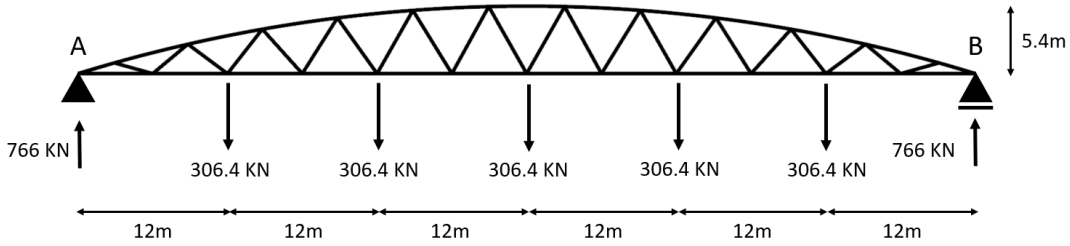


Figure 6.5: Loading conditions optimization Ahoy

To reduce computation times the problem is modelled with symmetry, the optimization grid is split in half and boundary conditions are defined by three roller supports. The grid of 36x6 contains three point loads, the value of the point load at the centre of the truss is divided by two.

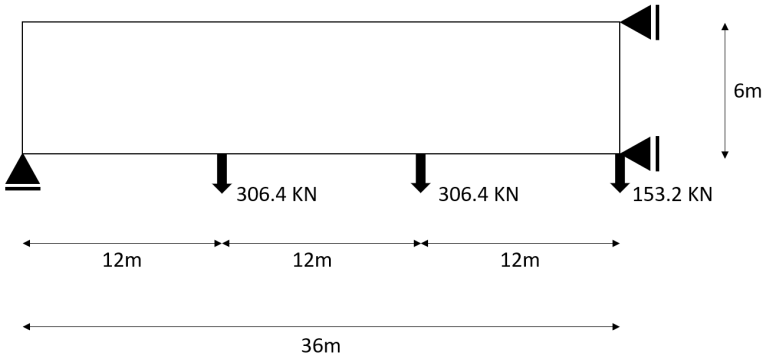


Figure 6.6: Optimization grid, boundary conditions and loads for modelling with symmetry

6.2. Database: Provinciehuis Zuid-Holland

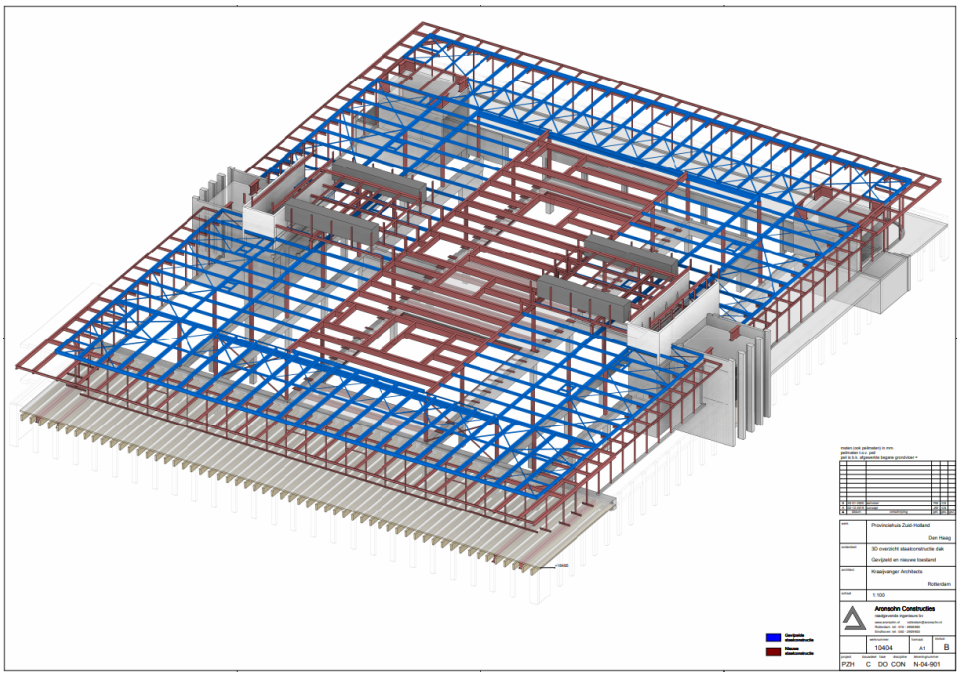


Figure 6.7: Steel structure Provinciehuis Zuid-Holland.

The Provinciehuis Zuid-Holland in the Hague is being renovated, the existing structure is replaced by a new structure. The existing structure, shown in Fig. 6.7, will be used as a donor skeleton in this case study to optimize the design of the trusses of Ahoy. The Structure contains a large variety of steel HEA, IPE, UNP and L profiles. The stock contains 96 elements with different length or cross section with a total of 236 elements. The optimization is performed with two edited databases, A and B. The raw data contains many large beams with length of 18m, 28 or 45m. Some of these beams are cut into multiple sections to increase variety and number of elements. Elements with length close to each other within a range of 5% are cut to the same length, to decrease the number of different available elements while increasing availability per element. A large variety of lengths of elements results in a PSML with many members, however, when all members are only available once or twice many new member penalty's are required to obtain solutions with high percentages of reuse, the penalty system is less effective.

Difference between stock A and B is the number of different and total elements. Stock A contains less different elements but a larger total amount of elements than stock B, large elements are cut into more sections than in stock B. All elements included in the databases can be found in Appendix C.1.

Database	Number of different elements	Total number of elements
Original	96	236
A	55	403
B	65	373

Figure 6.8: Original and edited databases with elements of Provinciehuis Zuid-Holland

The optimization problem is modelled with symmetry. Therefore, the availability of all elements from stock for the design of 1 truss is divided by 2. Elements which are only available once can not be used and elements available 3 times can only be used once. Edited Stock A and B are edited again to increase effectiveness of the optimization method and to increase efficiency and reuse percentage of the designs. Smallest and/or largest cross sections are removed to decrease difference in cross sections reducing the chances of assignment of inefficient elements. Elements with low availability are filtered to decrease the number of different

elements and reduce the need for new member penalty's. Edited stocks for performing the optimizations with symmetry are provided in Fig. 6.9, all elements included in the databases can be found in Appendix C.2

Stock	Trusses	Min Area	Max Area	Min Length	Min Availability	Number of different elements	Total number of elements
A1	1	1.0	8.0	0.0	2	27	147
A2	1	0.0	15.0	3.0	2	38	171
B1	1	1.0	10.0	0.0	2	34	139
A3	2	1.0	15.0	0.0	1	30	72
A4	2	0.0	15.0	3.0	2	20	62
B2	2	1.0	10.0	0.0	2	18	51

Figure 6.9: Edited databases for performing optimizations with symmetry

6.3. Results

A total of 4 different optimizations have been performed, analysed and compared with each other. The truss of Ahoy is optimized without reused elements with the original member adding scheme of (He et al., 2019) in Section 6.3.1. Optimizations are performed with and without joint cost. Optimizations with reused elements are performed in Section 6.3.2 without scaling factor and in Section 6.3.3 with scaling factor. Optimizations are performed for the design of 1 and 2 trusses. The designs are analysed and compared with each other in Section 6.3.4. Maximum tensile and compression stresses are respectively 355 and 248 N/mm2.

6.3.1. Optimizations with original member adding scheme

Optimization with jc of 0.0

Volume	Members
535	311

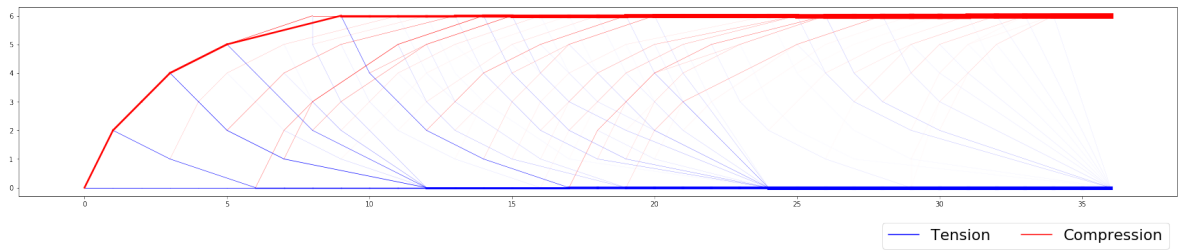


Figure 6.10: Optimization Ahoy with original member adding scheme with jc of 0.0

Optimization with jc of 3.0

Volume	Members
553	21

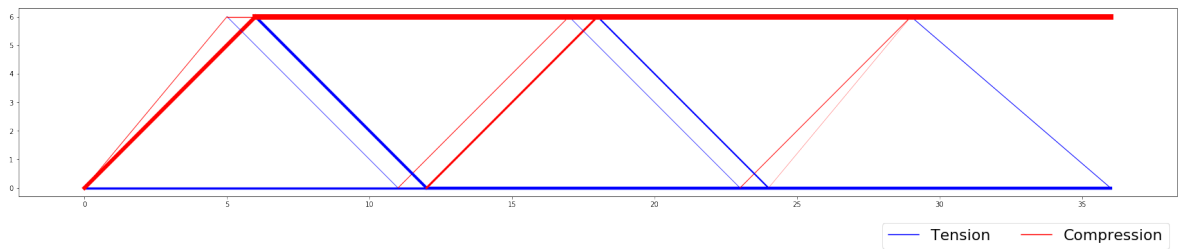


Figure 6.11: Optimization Ahoy with original member adding scheme with jc of 3.0

6.3.2. Optimizations with reused elements

In the design grid of 36x6, about 10.000 elements are present in the PSML for the provided edited databases of Fig. 6.9. This optimization problem is too complex, no solutions could be obtained with a minimum average unity check of 0.60. The combination of optimization grid, boundary- and loading conditions and available elements result in an optimization problem with complexity beyond capabilities of the design tool.

6.3.3. Optimizations with reused elements with scaling factor

Scaling factors of 1/2 and 1/3 have been applied to decrease complexity by decreasing the number of nodes and elements in the PSML. A scaling factor of 1/2 decreases the number of nodes from 216 to 54 and number of elements in the PSML from +/- 11.000 to +/- 1200. A scaling factor of 1/3 decreases number of nodes to 24 and number of elements in the PSML to +/- 300. An illustration of scaling the design domain with a scaling factor of 1/2 is provided in Fig. 6.12 and Fig. 6.13. A range of the number of elements in the PSML is provided for the available databases of Fig. 6.9.

Optimization with scaling factor of: 1.0
- Number of nodes: 216
- Number of elements in PSML: 10.000 - 12.000

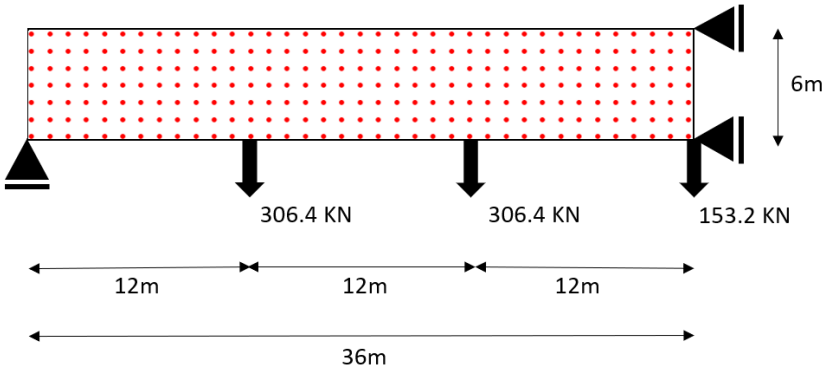


Figure 6.12: Nodes and elements: Optimization problem Ahoy with scaling factor of 1.0

Optimization with scaling factor of: 1/2
- Number of nodes: 54
- Number of elements in PSML: 900 - 1500

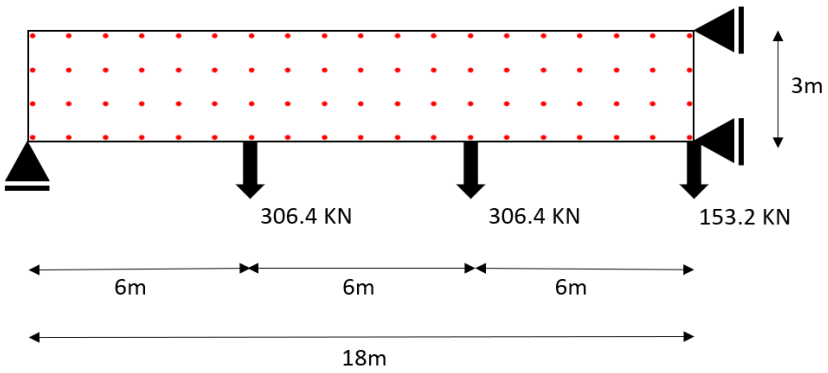


Figure 6.13: Nodes and elements: Optimization problem Ahoy with scaling factor of 1/2

Optimizations have been performed for the design of 1 truss and 2 trusses, modelled with symmetry the number of available elements are divided by 2 and 4. The corresponding edited stock with reclaimed elements is given for each optimization. Index numbers of reused elements are plotted in the graphs, index numbers refer to the index numbers of the elements in the database which can be retrieved from Appendix C.2. A total of 10 designs is presented, 6 designs for 1 truss and 4 designs for 2 trusses.

Design 1 - 1 Truss

Stock	Scale factor	Volume	Members	Reuse %	Average UC	Average UC Reused elements
A1	½	730	21	95	0.77	0.76

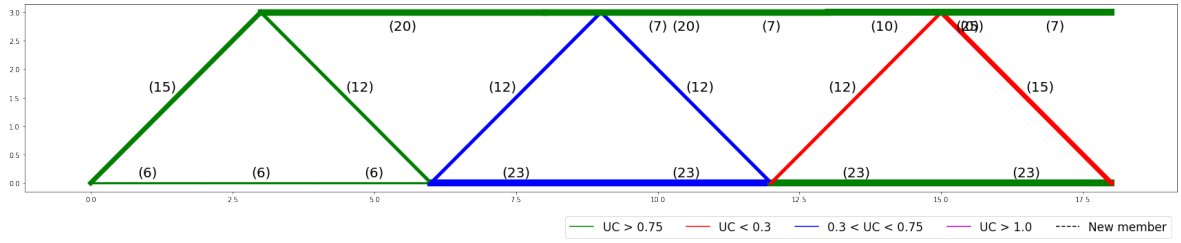


Figure 6.14: Optimization 1 - Ahoy with reused elements - Stock A1

Design 2 - 1 Truss

Stock	Scale factor	Volume	Members	Reuse %	Average UC	Average UC Reused elements
A1	½	716	21	86	0.80	0.77

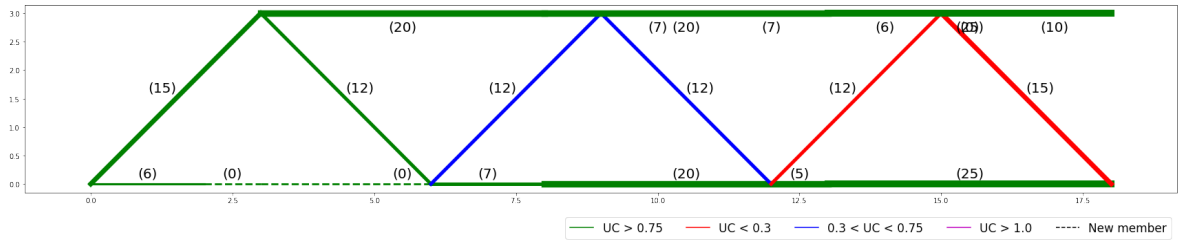


Figure 6.15: Optimization 2 - Ahoy with reused elements - Stock A1

Design 3 - 1 Truss

Stock	Scale factor	Volume	Members	Reuse %	Average UC	Average UC Reused elements
B1	½	681	18	89	0.77	0.75

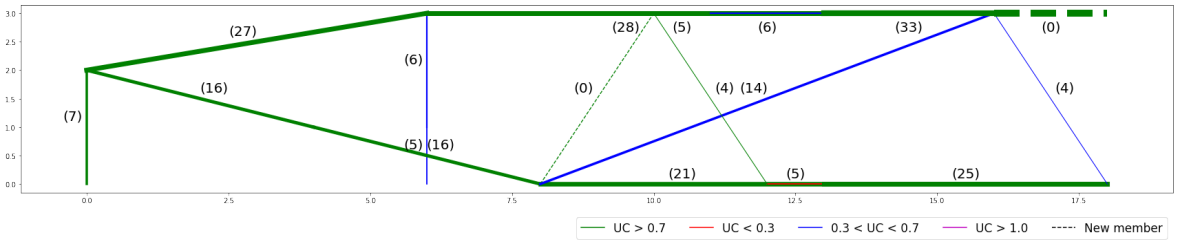


Figure 6.16: Optimization 3 - Ahoy with reused elements - Stock B1

Design 4 - 1 Truss

Stock	Scale factor	Volume	Members	Reuse %	Average UC	Average UC Reused elements
B1	½	748	19	100	0.70	0.70

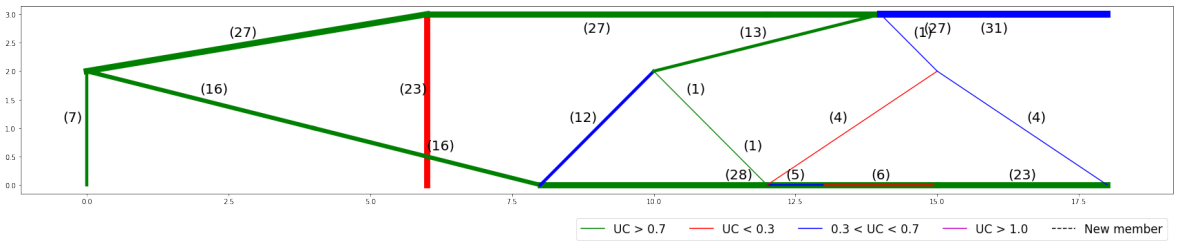


Figure 6.17: Optimization 4 - Ahoy with reused elements - Stock B1

Design 5 - 1 Truss

Stock	Scale factor	Volume	Members	Reuse %	Average UC	Average UC Reused elements
A2	⅓	789	23	100	0.75	0.75

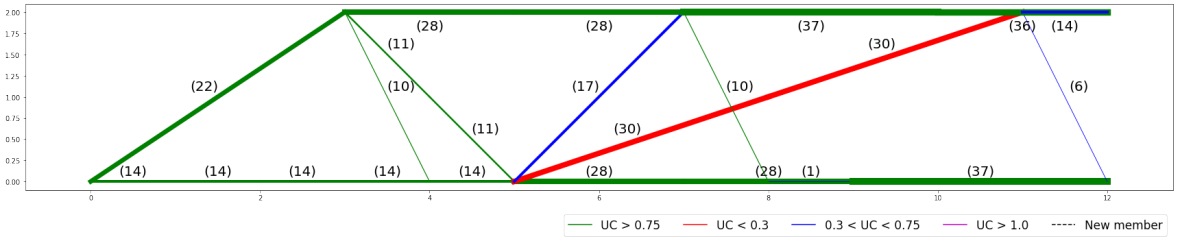


Figure 6.18: Optimization 5 - Ahoy with reused elements - Stock A2

Design 6 - 1 Truss

Stock	Scale factor	Volume	Members	Reuse %	Average UC	Average UC Reused elements
A2	⅓	732	24	92	0.78	0.76

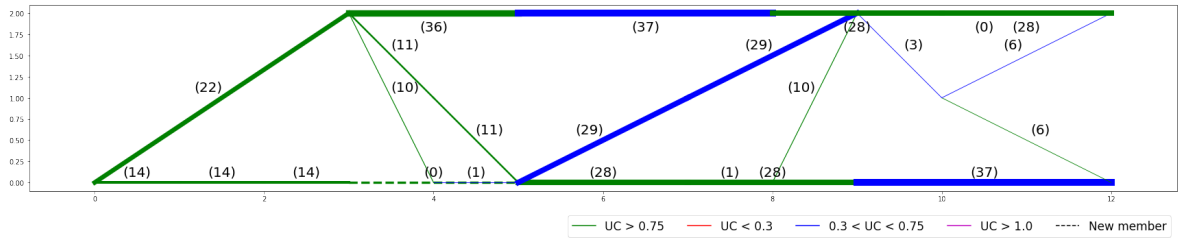


Figure 6.19: Optimization 6 - Ahoy with reused elements - Stock A2

Design 7 - 2 Trusses

Stock	Scale factor	Volume	Members	Reuse %	Average UC	Average UC Reused elements
A3	½	705	18	67	0.84	0.76

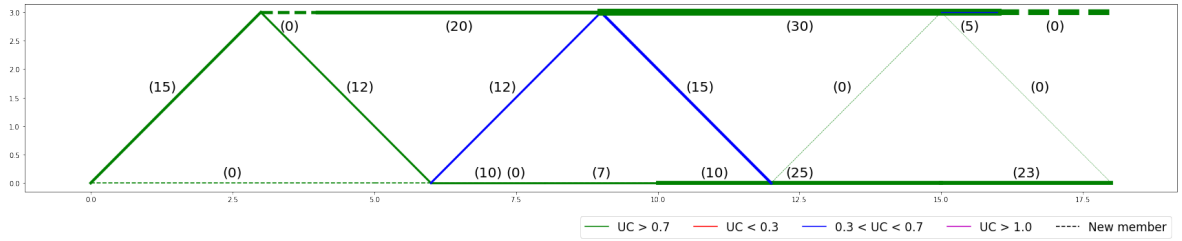


Figure 6.20: Optimization 7 - Ahoy with reused - elements Stock A3

Design 8 - 2 Trusses

Stock	Scale factor	Volume	Members	Reuse %	Average UC	Average UC Reused elements
A3	½	725	17	65	0.82	0.72

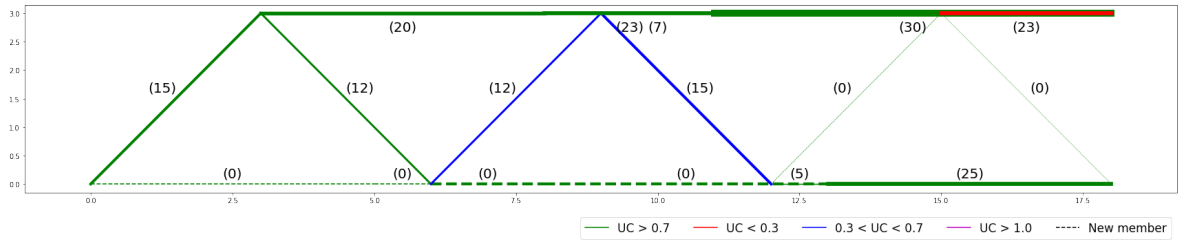


Figure 6.21: Optimization 8 - Ahoy with reused - elements Stock A3

Design 9 - 2 Trusses

Stock	Scale factor	Volume	Members	Reuse %	Average UC	Average UC Reused elements
B2	½	750	20	85	0.77	0.73

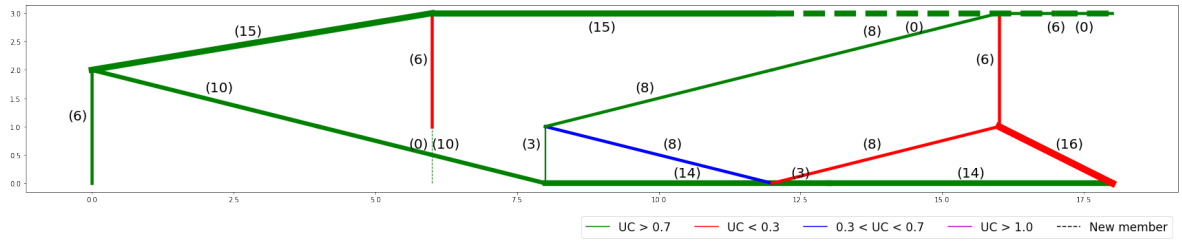


Figure 6.22: Optimization 9 - Ahoy with reused - elements Stock B2

Design 10 - 2 Trusses

Stock	Scale factor	Volume	Members	Reuse %	Average UC	Average UC Reused elements
A4	⅓	753	23	87	0.79	0.76

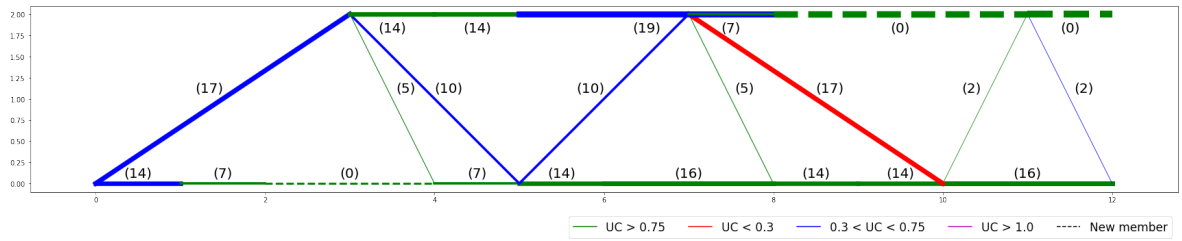


Figure 6.23: Optimization 10 - Ahoy with reused - elements Stock A4

6.3.4. Analysis and conclusion

The optimization problem in this case study is at the limit of the capabilities of the optimization tool. The combination of available elements and boundary- and loading conditions make the optimization problem complex, many elements are present in the PSML and the stock of elements contains a large variety of small and large cross sections. Without reducing the nodal density no results can be obtained. However, scaling the problem was effective to reduce complexity. By varying penalty's and the stock with available elements, many designs could be obtained with different topology. On average, all designs are not inferior to each other. Difference is reuse percentages results in difference in volume and average unity check, designs with lower reuse percentage contain more oversized elements increasing volume.

Reuse percentages of 100% and 87% could be obtained for the design of respectively 1 and 2 trusses. In the design for 1 truss in Fig. 6.14 23 of 139 available elements are used, in the design for 2 trusses in Fig. 6.18 23 of 62 available elements are used. Compared to the optimization with the original optimization method without reused elements, volume increases could be reduced to 30%. The number of members is for the designs with reuse about equal to the design with the original optimization method with joint cost. An overview is provided in Fig. 6.24, best and worst performing designs are highlighted in green and red.

Compared to the design of the existing roof structure of Ahoy, large volume reductions of almost 50% are achieved. These volume reductions are not realistic, wind loads and material factors are not taking into account as well as buckling, nodal instability and deflection limits. The optimized structures are only verified for compression and tensile stresses with a reduction of 70% for maximum compression stresses.

Design New Roof Structure Ahoy						
Design	Volume	Members				
Existing	1373	46				

Optimized Designs Original Member Adding Scheme						
Design	Volume	Members				
Jc = 0.0	535	311				
Jc = 3.0	553	21				

Optimized Designs Member Adding Scheme with Reused Elements – 1 Truss							
Design	Stock	Volume	Members	Reuse %	Average UC	Average UC Reused Elements	Scale factor
1	A1	730	21	95	0.77	0.76	½
2	A1	716	21	86	0.80	0.77	½
3	B1	681	18	89	0.77	0.75	½
4	B1	748	19	100	0.70	0.70	½
5	A2	789	23	100	0.75	0.75	⅓
6	A2	732	24	92	0.78	0.76	½

Optimized Designs Member Adding Scheme with Reused Elements – 2 Trusses							
Design	Stock	Volume	Members	Reuse %	Average UC	Average UC Reused Elements	Scale factor
7	A3	705	18	67	0.84	0.76	½
8	A3	725	17	65	0.82	0.72	½
9	B2	750	20	85	0.77	0.73	½
10	A4	753	23	87	0.79	0.76	⅓

Figure 6.24: Overview and comparison of existing and optimized structure and optimized structure with reused elements

Overlapping elements

The optimized structures contain many overlapping and intersecting elements, especially many overlapping elements in design 1 and 2 for database A1 in Fig. 6.14 and Fig. 6.15. The upper beam seems of uniform thickness, however, multiple elements are overlapping and take up part of the load. The internal load distribution for the topology of Fig. 6.14 and Fig. 6.15 is given in Fig. 6.25. Closer to the centre of the truss loads increase significantly thus requiring larger thickness of elements.

Overlapping and intersecting members are included in the original member adding scheme of He et al. (2019), to increase applicability of the optimization tool intersecting and overlapping members should be avoided, however, this is out of scope for this research.

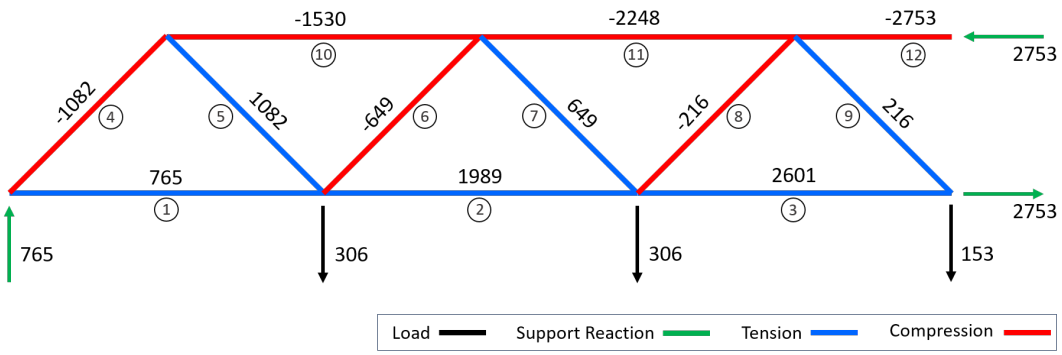


Figure 6.25: Optimized Truss Fig. 6.14 and Fig. 6.15: Loads, support reactions and internal loads in KN

Optimizing average unity check
Internal loads in the diagonals of the truss in Fig. 6.25 are highest in diagonal 4 and 5. However, in Fig. 6.14 diagonal 4 and 9 have received the larger cross section. Diagonal 4 is in compression and requires a larger cross section thus element 15 is assigned. Element 15 has been assigned to diagonal 9 either while element 12, which is available four times, has been assigned to diagonals 5, 6, 7 and 8. Assignment of element 15 with larger cross section to diagonal 5 with larger internal force could be more logically. However, maximum average unity check is optimized during element assignment. Different assignment would be less efficient with lower average unity check. Besides, inefficient elements receive penalty's: with assignment of element 15 to diagonal 5, the unity check would be 0.57 and element 5 would receive a penalty. Optionally, element assignment could be adjusted after the final iteration.

Influence of invisible elements
In the optimization, a threshold number of 1e-2*MaxArea is used for cross sections to be invisible. When assigning a cross section with a size of 1e-2*MaxArea results in the highest capacity utilization, elements retain their optimized cross section and receive no assigned cross section. When the optimized cross section is smaller than 1e-2*MaxArea elements are regarded as non existing and not visible in the design. When the maximum available cross section (MaxArea) is relative large, it could occur that elements with a cross section smaller than 1e-2*MaxArea do actually contribute to internal force equilibrium. These situations occurred in the optimization of the trusses of Ahoy with Provinciehuis Zuid-Holland as database. An example is provided in Fig. 6.26. In the highlighted red dot in the design no equilibrium of internal forces is established, a third elements is required which is invisible in the design.

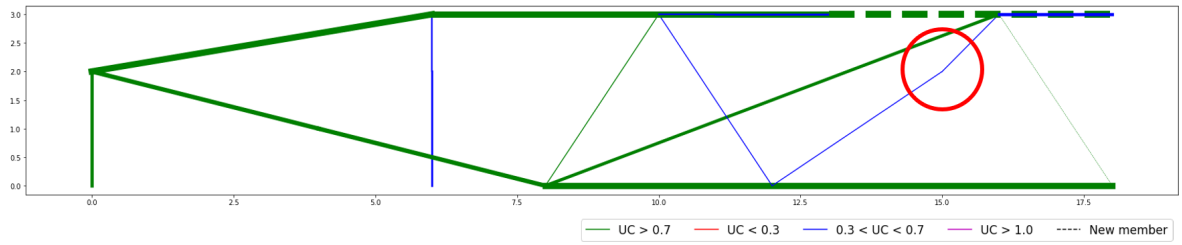


Figure 6.26: Design without equilibrium of internal forces

In conclusion, disregarding buckling, nodal instability, and intersecting and overlapping members; efficient designs could be obtained with high percentages of reuse. Scaling the design domain by a factor 1/2 and 1/3 proved to be effective methods to reduce complexity. By editing the databases with available elements and varying the optimization parameters, multiple different designs could be generated for the roof structure of Ahoy with elements reclaimed from Provinciehuis Zuid-Holland.

7

Discussion

In this chapter the developed design tool and results of examples and case study are discussed. Points of discussion are the effectiveness and limitations of the design tool, suitability of topology optimization for designing with reuse, suitability of the member adding scheme and efficiency of results.

Member adding scheme with reused elements
In this paper, the existing topology optimization method of He et al. (2019) has been extend with new functions able to design with reused elements. To the authors knowledge, only few methods exist in literature for designing with standard or reused elements. The method of Brütting et al. (2019b) described in Section 3.5 used for inspiration has been adapted, extended and improved. An iterative optimization process has been developed in which elements can disappear and reappear in further iterations. Designs can be generated with limited availability of reused elements resulting in designs with less than 100% reuse. In Section 5.7 the design tool of Brütting et al. (2019b) has been approached and compared to the developed design tool. Example 1 and 2 illustrate effectiveness of the new optimization method. For large as well as limited availability, average unity check and reuse percentage could be increased and volume and number of elements decreased. Average unity check could be increased from 0.88 to 0.98 and 0.63 to 0.78. The method of Brütting et al. (2019b) however, includes a geometry optimization in which material waste by cutting of elements could be minimized.

Topology optimization, reuse and volume reductions
Topology optimization results in designs with less volume compared to standard designs. Combined with reused elements in this design tool, this all depends on availability of elements and level of complexity. In the case study in Section 6.3 and the example of the 22m truss in Section 5.8, volume reductions could not be obtained. However, volume increase could be limited to 20-30%. Volume reduction due to topology optimization could partly compensate for volume increase due to oversized elements. In examples with large availability and limited complexity, such as the example in Appendix A, efficient designs with average unity check up to 0.98 could be obtained. However, compared to a standard optimized design, volume increases of 20% were observed. Limited diversity of available elements from stock resulted in less optimal topology than in the standard design.
Topology optimization often results in designs with more members and extraordinary layout compared to standard designs. When all reused elements are oversized, large volume increases could be obtained compared to standard designs. Standard topology with less elements and less oversized elements could result in designs with reused elements with less volume.

Member adding scheme

- An advantage of the member adding scheme of He et al. (2019) is the use of discrete elements. A drawback is that only limited geometry information can be included. The advantage of linear programming is at cost of limited geometry information of elements included in the optimization resulting in oversized elements.

- In the original member adding scheme of He et al. (2019) initial connectivity has no influence on the obtained results, a reduced initial connectivity decreases computation times significantly. In the developed optimization method however, initial connectivity does influence the optimization. A reduced

initial connectivity often has a negative influence on the optimization. Oversized elements cause requirements for virtual strain to be met early. Many potentially efficient members in the PSML are not added to the reduced initial connectivity thus not considered in the design. Optimizations with all members included in the initial connectivity often result in more efficient and/or convenient designs. Member adding should be turned off, meaning the ground structure method of Dorn (1964) should be used.

Adaptations and Additions
Many adaptations and additions made to the member adding scheme of He et al. (2019) are, to the authors knowledge, new from existing literature. New functions applicable for designing with reused elements but could also be used for other applications. New functions are discussed under the bullet points below.

- **Possible stock member list**
  Varying the maximum Lcut and Lshort is an effective method to influence the total number of elements in the PSML. Increasing Lcut increases the number of elements. However, no penalty's are applied based on Lcut lengths thus material waste.
  Illustrated in the example of the 22m truss in Section 5.8 designs often contain overlapping elements. This is caused by the presence of overlapping elements in the PSML. However, overlapping elements are required in the PML and PSML to be able to consider all available elements, thus lengths, from stock. To avoid overlap of elements, methods such as the method of Cui et al. (2018) should be used.

- **Element assignment**
  The extended problem formulation with element assignment and capacity utilization, based on the two step approach of Brütting et al. (2019b), is an efficient method to assign cross sections as efficient as possible. Two considerations were made which result in assignment of oversized elements. First of all, a unity check below or equal to the maximum unity check is preferred which could lead to situations where a much larger cross sections than necessary must be assigned. Second, situations occur where cross sections from stock must be assigned to elements with very small internal forces. Elements are only regarded as non existing and invisible in the design when assignment of a cross section of 1e-2$*$maxArea is most efficient.
  Fig. 6.26 in the case study in Chapter 6 illustrated that situations can occur where invisible elements actually do influence the design, designs are generated without equilibrium of internal forces. When the maximum available cross section is large, the threshold number for elements to be invisible should be decreased to 1e-3$*$maxArea or smaller.

- **Recalculating the structure**
  Nodal displacements are recalculated for the assigned cross sections with the matrix method. To reduce computation times, programming of the matrix method may be done more efficiently. During calculation, an error 'singular matrix' may arises, stiffness matrix K is not invertible and the optimization is interrupted. This occurs only rarely when no unique solution exist. In this case, the python function linalge.solve is not able to solve the equation.

- **Extra conditions**
  Over sized designs often meet requirements for virtual strain but are very inefficient in terms of average unity check and reuse percentage. With extra conditions, the optimization keeps iterating and efficient designs are presented. Every iteration different designs are generated due to the penalization of inefficient and new members in previous iterations. Example 1 and 2 in Section 5.7 illustrated the effectiveness of iterating with the penalty system compared to optimizing without penalty system and extra conditions.

- **Penalty system**
  For smaller design problems the penalty system is an effective method to influence the optimization of cross sections. For larger design problems, such as the the case study in Chapter 6 with many members included in the PSML, the penalty system is less effective. Penalized members can easily be replaced by members close by in the design domain which requires many penalty's to be applied.

  During the iterative optimization process, optimal topology can differ much. Designs can contain in one iteration many different elements compared to designs of previous iterations, with which internal load distribution is different. In one iteration, an element can be efficient while in the next iteration

the same elements is inefficient. An illustration is provided in Appendix B for the example of the 22m truss. Values for penalty's are low such that penalty's are soft penalty's, allowing elements to be present in further iterations. Final designs often contain elements with penalty's. With increasing complexity however, topology can differ much for many iterations in succession. Penalization of penalty's can be considered as randomly. Often no designs are found with high efficiency.

Values of penalty's are determined based on the average length of elements, a more effective method could be to determine penalty's based on individual length of elements. Longer elements result in larger volume increase when inefficient and should receive higher penalty's. However, designs with few members, thus more longer elements, could be preferred over designs with many members, number of members could be preferred over volume increase.

Joint cost is the most important penalty to influence the number of elements thus average cross section of elements in the design. All other penalty's are important as well, however influence of penalty's and suitable value's for penalty's are difficult to predict. Therefore as many combinations of value's of penalty's should be considered. However, with a large number of different penalty's, number of combinations can increase rapidly often up to 500 combinations increasing computation times.

Complexity
In the ground structure method of Dorn (1964) and member adding scheme of He et al. (2019), an increasing number of nodes and elements in the ground structure increase complexity and thus computation times significantly . Complexity of optimization problems in the developed optimization tool is not only influenced by grid size and number of nodes. All input increasing or decreasing complexity are:

1. Grid size and number of nodes
2. Number of members in the PSML
3. Number of different elements available from stock
4. Availability of individual elements from stock
5. Number of point loads

Explanation is provided under the bullet points below.

- Grid size, number of nodes and number of members in the PSML correlate with each other. Increasing grid size and thus number of nodes results in an increase of the number of members in the PSML. However, a large grid does not necessarily mean a large number of members in the PSML. When only large lengths are available, the number of members will be limited. Parameters to influence and vary the number of members in the PSML are Lcut and Lshort. Increasing Lcut will increase the number of members.

- The number of different elements from stock influences efficiency of assignment of elements from stock. With a large stock with large deviations between size of cross sections, computation times increase and element assignment is often inefficient. Many cross section penalty's are required to obtain efficient results. Similarly, with limited availability of elements from stock, many new member penalty's are required to obtain results with high reuse percentages.

- At every point load equilibrium of internal and external forces must be established. An increasing number of point loads limits design freedom and increases complexity. Applying multiple points loads influences the internal force distribution: Larger differences between smallest and largest internal force occur. Elements assignment is often less efficient and requires more cross section penalty's to obtain efficient results.

- Increasing complexity reduces effectiveness of the optimization tool and quality of the results, it becomes more difficult to obtain designs with high percentage of reuse and average unity check. Examples and illustrations in Chapter 5 showed effectiveness of individual and combinations of adaptations and additions for simple optimization problems.

Example 1 in Section 5.7 illustrated that with low complexity designs with an average unity check of 0.95 with 100% reuse could be obtained. Computation times were low with only 74 members included in the ground structure. Availability of individual elements was high and available cross sections matched very well with the required cross sections.

The case study in Chapter 6 showed that no results could be obtained for a very complex optimization problem. The problem contained a grid size of 72x6, 5 point loads and many different available elements from stock with low availability of individual elements. Methods to reduce complexity were effective. Number of nodes was reduced from 432 to 54 and number of possible members from 26000 to 1000. Editing the stock of available elements could reduce the number of different elements from 96 up to 18 and increase average availability of individual elements from 2.5 up to 4.5. With reduced complexity multiple different designs could be obtained.

Methods to reduce complexity can be effective. However, with a limited number of nodes and possible elements number of different solutions is limited. Solutions could be less optimal, further away from global optima. Other drawbacks are that hinges are required at the centre of the structure to connect both 'half' designs and elements available 3 times from stock can only be used once in each design. An advantage however, is that only symmetric designs can be presented.

Optimal solutions
The design tool contains two optimizations: topology optimization and optimization of element assignment. Individually, both optimizations result in global optima. Minimal volume is achieved for the provided ground structure and applied penalty's. Maximum average unity check is achieved for the provided internal forces and available elements from stock. However, it could be argued whether the combined optimizations result in global optima for designs with reused elements in terms of minimal volume, minimum number of elements or maximum average unity check. An infinite number of combinations of parameters can be inserted resulting in a large number of different designs. Designs are provided when all requirements are met, not when a certain optimum is achieved. The optimization tool only stops when a maximum number of iterations is reached. Whether a design with a global optimum is within the generated designs is unknown.

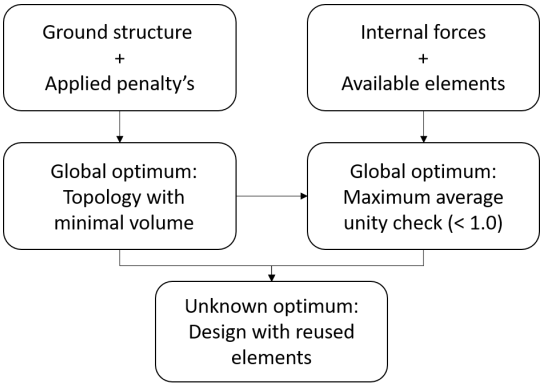


Figure 7.1: Global optima and unknown optima

Applicability
The developed design tool can be used for the design of steel trusses with reused elements. Optimizations can be performed with small or large databases of available elements. Design domains can be large or small, however, efficiency of results decreases with increasing complexity.

The optimized trusses can not be used in practice. Nodal instability, buckling and self-weight must be included. Overlapping and intersecting elements must be excluded. Unstable nodes arise between elements connected connected behind each other in a straight line. Buckling is only taken into account by reducing the maximum compression stresses to 70% of the maximum tensile stresses, independent on the ratio of length and cross section. Self-weight is not included and in the case study only taken into account by an extra external load. Optimization problems with high loads and small available cross sections often results in structures with many overlapping members, each members taking a part of the internal load. Intersecting as well as overlapping members make construction complicated requiring extra labour.

Besides designing with reused elements, the tool could also be used to design with standard elements. Assignment of elements could be performed without taking availability into account.

8

Conclusions and recommendations

8.1. Conclusions

In this section the main research question is answered.

- "To what extend can a topology optimization method be used to simplify the complex design process with reused elements for the design of steel trusses?"

To answer this question a design tool has been developed. To develop the tool, two important sub questions had to be answered:

1. “Which existing topology optimization method is most suitable to use for the proposed optimization method”

2. “Which adaptations and additions must be made to the existing optimization method to develop the proposed optimization method”

Answers to the subquestions:

- In Section 3.6 it was concluded that the member adding scheme of He et al. (2019) was most suitable. A method with linear programming with discrete elements. Adaptations and additions were required to this method to include geometry information of reused elements. Length and size of elements is fixed posing a complex optimization problem.

- Geometry information of reused elements is implemented by filtering the ground structure and by using a two step optimization approach with element assignment based on the method of Brütting et al. (2019b). A penalty system is developed penalizing inefficient and new elements in the designs. A parametric tool with an iterating optimization process is developed presenting designs when requirements are met for violation of virtual strain, minimum average unity check and minimum reuse percentage. Volume is minimized in every iteration with the optimization algorithm. The designs are verified for maximum compression and tensile stresses.

Answers to the main research question:

- The discrete topology optimization method of He et al. (2019), with all adaptations and additions, proved to be an effective method to automatize and simplify the design process. Designs may not be global optima but can provide the user many design alternative with different volume, number of elements, reuse percentage and average unity check. With limited complexity, designs can be generated within seconds with an average unity check close to the maximum unity check. With increasing complexity however, effectiveness of the design tool decreases. Complexity is influenced by grid size, number of possible elements, availability of individual and different elements and number of point loads.

- The developed design tool is an improvement of the existing method of Brütting et al. (2019b). Two examples in Section 5.7, where the method of Brütting et al. (2019b) was approached, showed effectiveness of iterating with the penalty system. Compared to the approached method of Brütting et al. (2019b), volume and number of elements could be decreased and reuse percentage and average unity check increased. Average unity check could be increased from 0.88 to 0.98.

- Topology optimization with new elements results in designs with reduced volume compared to standard designs with new elements. In the developed optimization tool combined with reused elements, volume increase or decrease depends on the number, suitability and diversity of cross sections of available elements. This emphasizes the importance of organized documentation of buildings and elements available for reuse.

  In the case study and 22m example with limited availability, volume of designs with new and reused elements was compared. In the case study volume increase could be reduced to 30%. Reuse percentages of 100% and 87% could be obtained for the design of 1 and 2 trusses. Respectively 23 of 139 available elements and 23 of 62 available elements were used in the designs. In the example of the 22m truss, volume increase could be reduced to 20%. 97% reuse was obtained in which 30 of 60 available elements were used in the design.

8.2. Recommendations

Recommendations are made concerning the design tool and designing with reuse in general.

- Applicability of the design tool should be extend by including constraints for intersection, overlap, buckling and nodal instability of elements. Self-weight should be included as well.

- A graphical user interface could be added to make the optimization tool more accessible for people not familiar with Python. Limit State Peregrine, a tool illustrated in Section 3.4, can be used as an example.

- The penalty system is key in generating a variety of efficient designs. Determining what the best values are for penalty's and what the most suitable iterations are for penalty's to be applied is often a process of guessing and trail an error. Guidelines are provided but urge for the necessity of many combinations of values of penalty's increasing computation times with no guarantee to success. With increasing complexity effectiveness of the penalty system decreases. Effort could be put in an alternative or adapted more advanced penalty system to improve quality of the optimization.

- Analysis of the reduction of embodied carbon, energy and construction costs for designs with reused elements compared to designs with new elements could give insight in attractiveness of designing with reused elements.

- Designing with reused elements should be made more accessible. In the case study in this research, it was difficult to obtain data of buildings planned for demolition. Information could not be gathered from platforms such as Madaster and New Horizon due to privacy legislation. It should be more accessible to experiment with designing with reused elements from different buildings.

Bibliography

Rafaque Ahmad and Hari K Voruganti. Structural topology optimization: Methods and applications. In Advances in Applied Mechanical Engineering, pages 643–654. Springer, 2020.

A. Allwood, M. A bright future for uk steel: A strategy for innovation and leadership trough up-cycling and integration. Technical report, University of Cambridge, 2016.

Martin P Bendsøe. Optimal shape design as a material distribution problem. Structural optimization, 1: 193–202, 1989.

Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. Convex optimization. Cambridge university press, 2004.

Jan Brütting, Catherine De Wolf, and Corentin Fivet. The reuse of load-bearing components. In IOP Conference Series: Earth and Environmental Science, volume 225, page 012025. IOP Publishing, 2019a.

Jan Brütting, Joseph Desruelle, Gennaro Senatore, and Corentin Fivet. Design of truss structures through reuse. In Structures, volume 18, pages 128–137. Elsevier, 2019b.

Jan Brütting, Gennaro Senatore, Mattias Schevenels, and Corentin Fivet. Optimum design of frame structures from a stock of reclaimed elements. Frontiers in Built Environment, 6:57, 2020.

Philip Crowther. Developing an inclusive model for design for deconstruction. In Proceedings of the CIB Task Group 39-Deconstruction Meeting, pages 1–26. CIB, Int Council for Research and Innovation in Bd, 2001.

Huiyong Cui, Haichao An, and Hai Huang. Truss topology optimization considering local buckling constraints and restrictions on intersection and overlap of bar members. Structural and Multidisciplinary Optimization, 58(2):575–594, 2018.

WS Dorn. Automatic design of optimal structures. J. de Mecanique, 3:25–52, 1964.

Cyrille F Dunant, Michał P Drewniok, Michael Sansom, Simon Corbey, Julian M Allwood, and Jonathan M Cullen. Real and perceived barriers to steel reuse across the uk construction value chain. Resources, Conservation and Recycling, 126:118–131, 2017.

H Fairclough and M Gilbert. Layout optimization of simplified trusses using mixed integer linear programming with runtime generation of constraints. Structural and Multidisciplinary Optimization, pages 1–23, 2020.

Aurélie Favier, Catherine De Wolf, Karen Scrivener, and Guillaume Habert. A sustainable future for the european cement and concrete industry: Technology assessment for full decarbonisation of the industry by 2050. Technical report, ETH Zurich, 2018.

H. Ferland. Solid waste management and greenhousegases: a life-cycle assessment of emissions and sinks. Resource Recycling, January, 23 (available at: http://yosemite.epa.gov/oar/globalwarming.nsf/content/ActionsWasteToolsSWMGHGreport.html)., 2006.

Aboma Wagari Gebisa and Hirpa G Lemu. A case study on topology optimized design for additive manufacturing. In IOP Conference Series: Materials Science and Engineering, volume 276, page 012026. IOP Publishing, 2017.

Matthew Gilbert and Paul Shepherd. Doing more with less: Layout optimisation of structures, 2019. URL `https://www.youtube.com/watch?v=_MoiYTqx5fQ&t=3s`. YouTube video.

Matthew Gilbert and Andrew Tyas. Layout optimization of large-scale pin-jointed frames. Engineering computations, 2003.

Mark Gorgolewski. Designing with reused building components: some challenges. Building Research & Information, 36:175–188, 2008.

Xu Guo, Weisheng Zhang, and Wenliang Zhong. Doing topology optimization explicitly and geometrically—a new moving morphable components based framework. Journal of Applied Mechanics, 81(8), 2014.

Linwei He, Matthew Gilbert, and Xingyi Song. A python script for adaptive layout optimization of trusses. Structural and Multidisciplinary Optimization, 60(2):835–847, 2019.

Peter Hopkinson, Han-Mei Chen, Kan Zhou, Yong Wang, and Dennis Lam. Recovery and reuse of structural products from end-of-life buildings. In Proceedings of the Institution of Civil Engineers-Engineering Sustainability, volume 172, pages 119–128. Thomas Telford Ltd, 2018.

Scot W. Horst. Minnesota demolition survey: Phase two report. Technical report, The Athena Institute, 2004.

Endrit Hoxha, Guillaume Habert, Sébastien Lasvaux, Jacques Chevalier, and Robert Le Roy. Influence of construction material uncertainties on residential building lca reliability. Journal of cleaner production, 144: 33–47, 2017.

Aykut Kentli. Topology optimization applications on engineering structures. Truss and Frames—Recent Advances and New Perspectives, pages 1–23, 2020.

Niklaus Kohler and Uta Hassler. The building stock as a research object. Building Research & Information, 30 (4):226–236, 2002.

Paul Lagendijk and Casimir Slui. Interview: Information about loading- and boundary conditions of the roof structure of ahoy, Apr 2021.

Yuan Liu, Shurong Zhuo, Yining Xiao, Guolei Zheng, Guoying Dong, and Yaoyao Fiona Zhao. Rapid modeling and design optimization of multi-topology lattice structure based on unit-cell library. Journal of Mechanical Design, 142(9), 2020.

JA Norato, BK Bell, and Daniel A Tortorelli. A geometry projection method for continuum-based topology optimization with discrete elements. Computer Methods in Applied Mechanics and Engineering, 293:306–327, 2015.

M Ohsaki and N Katoh. Topology optimization of trusses with stress and local constraints on nodal stability and member intersection. Structural and Multidisciplinary Optimization, 29(3):190–197, 2005.

T. Rau. Gebruikershandleiding, Algemene handleiding van het Madaster Platform. Madaster, 2017.

BIO Intelligence Service. Sectoral resource maps. Prepared in response to an informationhub request, European Commission, DG Environment, 2013.

Angelo Simone. An Introduction to the Analysis of Slender Structures. TU Delft, 2007.

Tomasz Sokół. Topology optimization of large-scale trusses using ground structure approach with selective subsets of active bars. In 19th International Conference on Computer Methods in Mechanics. Warsaw, Poland, 2011.

O. Wainwright. The case for ... never demolishing another building. The Guardian, 2020.

Danièle Waldmann. Demountable construction enables structural diversity. Adjacent open access, 2017.

H. Welleman. Work, energy methods and influence lines. Bouwen met Staal, Zoetermeer, Netherlands, 2016.

Tomás Zegard and Glaucio H Paulino. Grand—ground structure based topology optimization for arbitrary 2d domains using matlab. Structural and Multidisciplinary Optimization, 50(5):861–882, 2014.

Tuo Zhao. An implementation of the ground structure method considering buckling and nodal instabilities. 2015.

A

Penalty system C: Example 3

Optimized designs with reused elements generated with penalty system C:

Design 1:

```
jc = 1.0, cspf = 1.0, Acspf = 0.35, Nmp = 2.0, ANmp = 0.5, maxUC = 2.0
.                    .                    .
Itr: 19, vol: 23.425556, mems: 74
Plotted members =  32
average UC =  0.946538654944439
average UC reused members =  0.946538654944439
maximum unity check =  1.0000000000329905
minimum unity check =  0.8485281373593984
Reuse percentage =  100.0 %
```

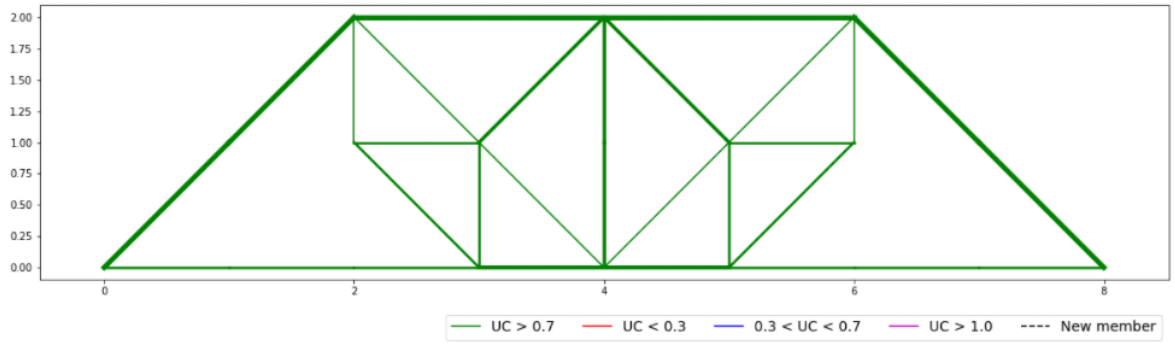


Figure A.1: System C Example 3.1 loadcase 3

Design 2:

```
jc = 1.0, cspf = 1.0, Acspf = 1.0, Nmp = 1.0, ANmp = 0.25, maxUC = 1.0
.                    .                    .
Itr: 42, vol: 24.673559, mems: 74
Plotted members =  26
average UC =  0.9785070213346531
average UC reused members =  0.9785070213346531
maximum unity check =  0.9999999999979675
minimum unity check =  0.8988395870796176
Reuse percentage =  100.0 %
```

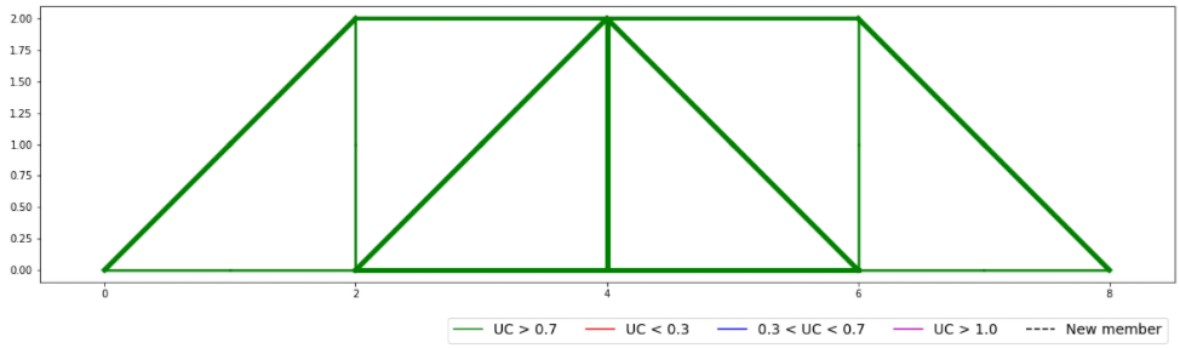


Figure A.2: System C Example 3.2 loadcase 3

Design 3:

```
jc = 1.0, cspf = 1.0, Acspf = 0.25, Nmp = 2.0, ANmp = 0.25, maxUC = 1.0
.                    .                    .
Itr: 33, vol: 24.036154, mems: 74
Plotted members =  28
average UC =  0.9363391482470502
average UC reused members =  0.9363391482470502
maximum unity check =  0.9999999997919652
minimum unity check =  0.8000000000559115
Reuse percentage =  100.0 %
```

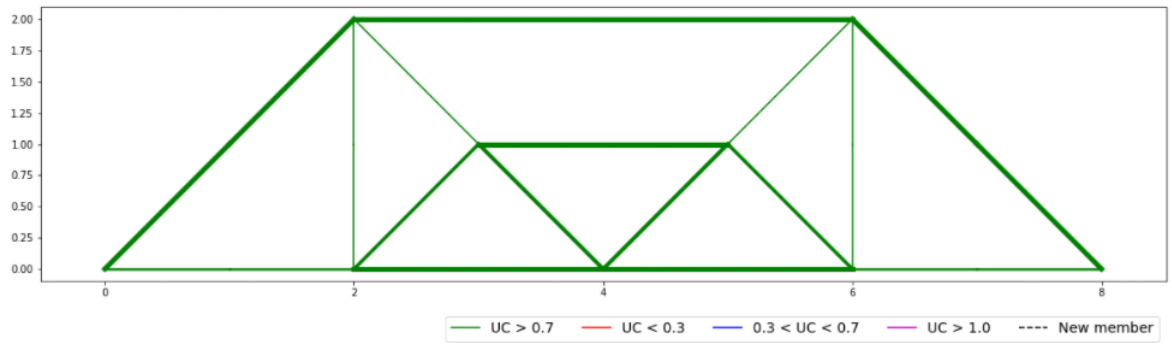


Figure A.3: System C Example 3.3 loadcase 3

Design 4:

```
jc = 1.0, cspf = 3.0, Acspf = 3.0, Nmp = 2.0, ANmp = 0.5, maxUC = 1.0
.                       .                       .
Itr: 12, vol: 24.992965, mems: 74
Plotted members =  26
average UC =  0.9638576696913266
average UC reused members =  0.9638576696913266
maximum unity check =  0.9999999999966157
minimum unity check =  0.8928571428489893
Reuse percentage =  100.0 %
```

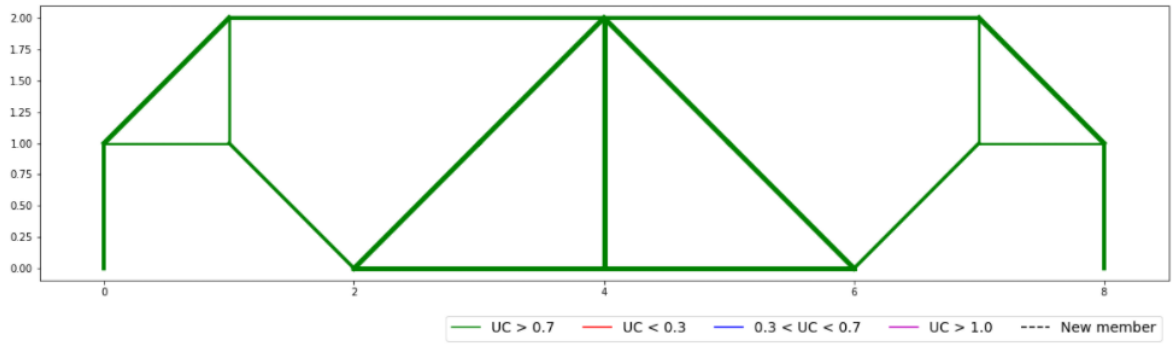


Figure A.4: System C Example 3.4 loadcase 3

Optimized designs with original member adding scheme:

Design 1:

```
jc =  0.0
Nodes: 27 Members: 226
Itr: 13, vol: 19.428571, mems: 86
Plotted members =  26
```

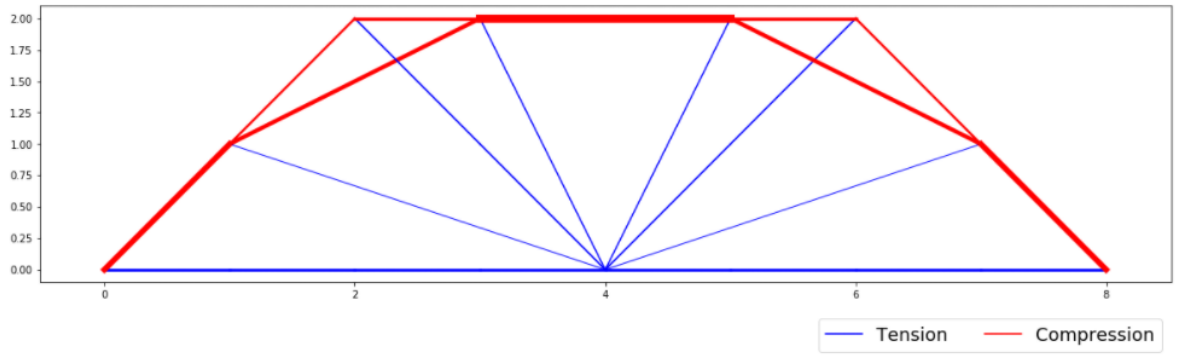


Figure A.5: System C Example 3: design with original member adding scheme with jc = 0.0

Design 2:

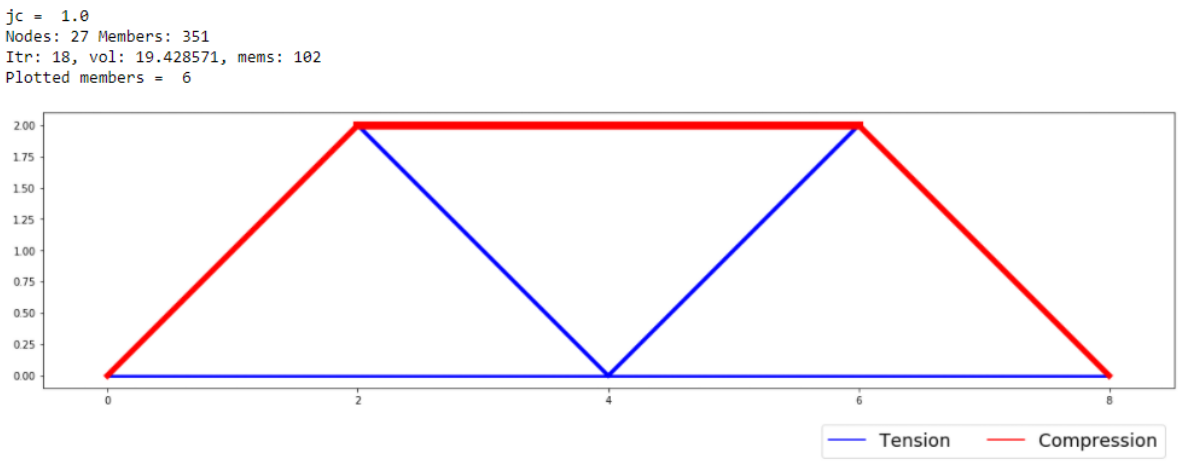


Figure A.6: System C Example 3: design with original member adding scheme with jc = 1.0

Comparison

Volume increases of 20% are observed for the designs with reused elements compared to the optimized designs. However, average unity checks of 0.93 to 0.98 could be achieved for the four different designs with reused elements. Volume increase is obtained due to limited availability of different lengths from stock. Only lengths of 1.0 and 1.41m are available with a maximum Lcut of 0.10. The maximum cross section is 1.0 resulting in designs with more elements. The maximum cross section in both optimized designs has a value of 1.41.

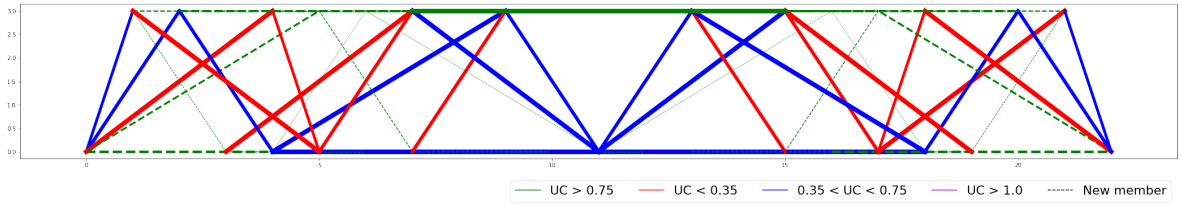
Optimized designs	**Volume**	**Number of elements**	**Reuse %**	**Average unity check**
Design 1	19.4	26	-	1.0
Design 2	19.4	6	-	1.0
Designs with reuse	**Volume**	**Number of elements**	**Reuse %**	**Average unity check**
Design 1:	23.4	32	100	0.95
Design 2:	24.7	26	100	0.98
Design 3:	24.0	28	100	0.93
Design 4:	25.0	26	100	0.96

Table A.1: Overview designs example 3: Original member adding scheme and member adding scheme with reused elements

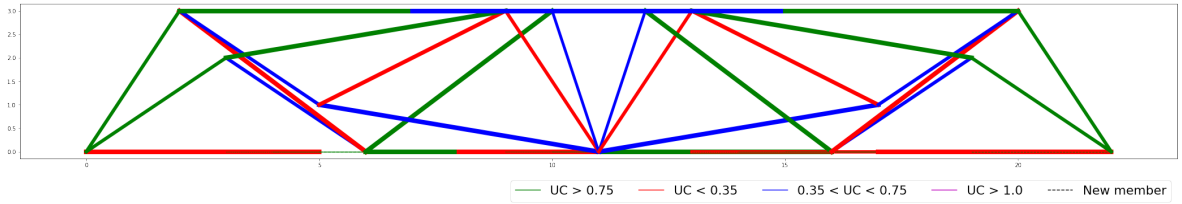
B

Iterations 22m Truss: Example 5

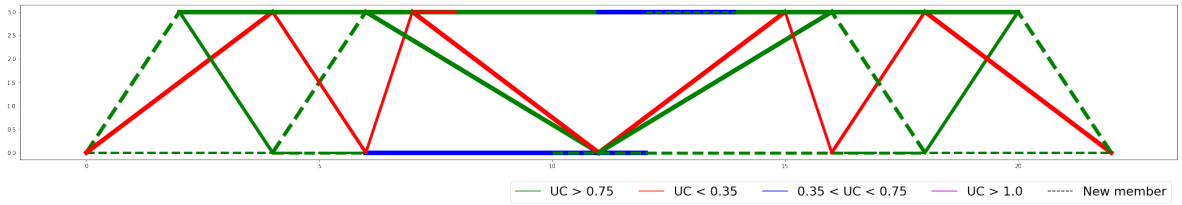
Iteration 1:



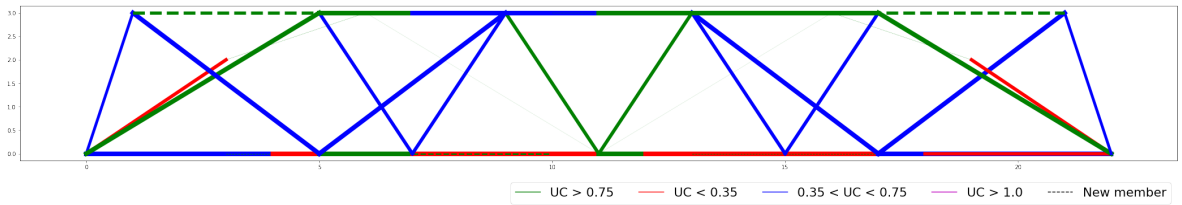
Iteration 2:



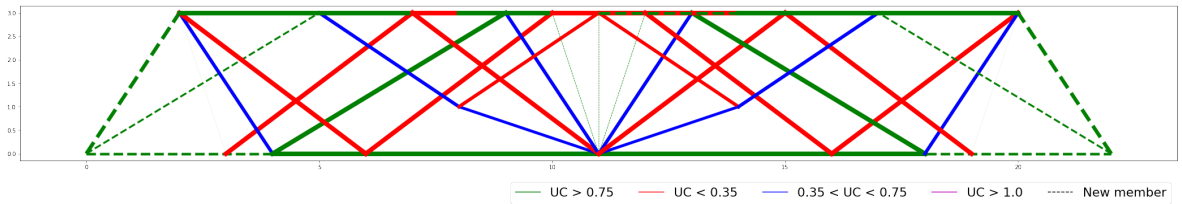
Iteration 3:



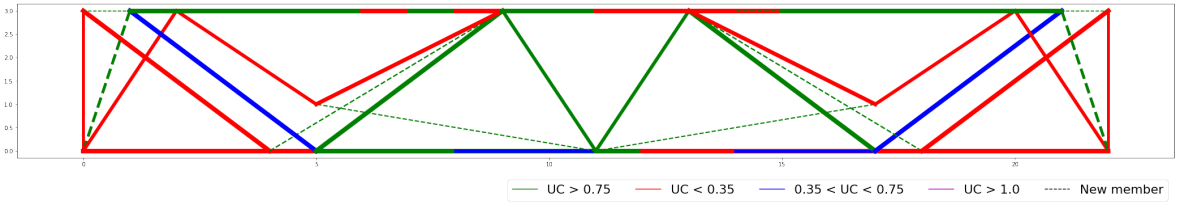
Iteration 4:



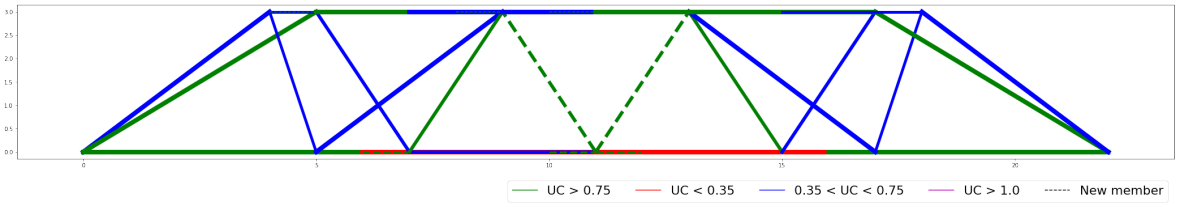
Iteration 5:



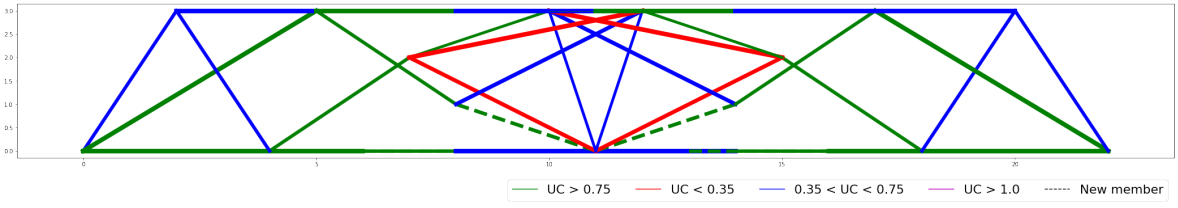
Iteration 6:



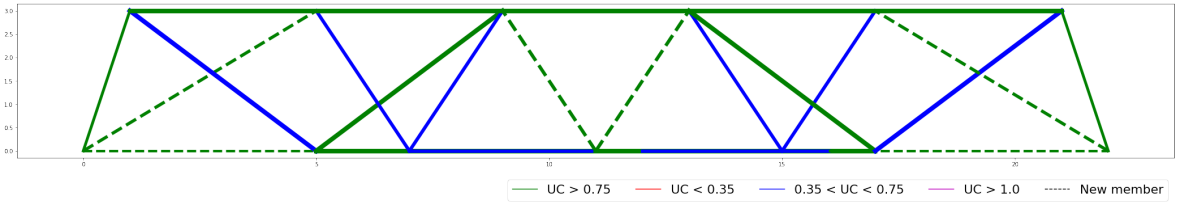
Iteration 7:



Iteration 8:



Iteration 9:



C

Edited databases Provinciehuis Zuid-Holland

6 Databases for the optimization of the steel trusses of Ahoy with reclaimed elements of Provinciehuis Zuid-Holland are provided. Elements in each database are described by a length **L**, cross section **a**, index number **i** and number of availability **n**.

$$\text{Element} = [L, a, i, n] \tag{C.1}$$

C.1. Edited database A and B

```
Stock A:
Number of different elements = 55, Total number of elements = 403
[[3.4, 0.83, 1, 8], [3.8, 0.83, 2, 5], [4.2, 0.83, 3, 6], [4.9, 0.83, 4, 7], [5.4, 0.83, 5, 7], [6.8, 0.83, 6, 8], [7.5, 0.83,
7, 2], [8.3, 0.83, 8, 11], [9.8, 0.83, 9, 3], [2.7, 1.15, 10, 12], [5.1, 1.15, 11, 14], [5.7, 1.15, 12, 4], [7.1, 1.15, 13,
8], [6.9, 1.35, 14, 3], [2.0, 2.01, 15, 8], [2.2, 2.4, 16, 2], [4.2, 2.4, 17, 6], [6.3, 2.4, 18, 1], [1.9, 3.14, 19, 1], [3.4,
3.88, 20, 3], [4.1, 3.88, 21, 6], [5.1, 3.88, 22, 16], [3.4, 3.91, 23, 12], [4.3, 3.91, 24, 48], [5.8, 3.91, 25, 18], [8.7, 3.
91, 26, 8], [17.4, 3.91, 27, 6], [2.6, 4.23, 28, 1], [3.0, 4.23, 29, 2], [3.4, 4.23, 30, 2], [6.3, 4.23, 31, 1], [0.9, 4.53, 3
2, 4], [3.5, 5.38, 33, 2], [4.3, 5.38, 34, 16], [8.6, 5.38, 35, 9], [5.4, 6.43, 36, 8], [10.9, 6.43, 37, 4], [3.3, 7.27, 38, 2
7], [5.1, 7.27, 39, 20], [10.0, 7.27, 40, 6], [5.2, 7.68, 41, 5], [5.9, 7.68, 42, 2], [10.9, 7.68, 43, 4], [4.4, 7.73, 44, 2],
[6.1, 7.73, 45, 8], [6.9, 7.73, 46, 7], [9.9, 7.73, 47, 4], [11.0, 7.73, 48, 8], [13.2, 7.73, 49, 6], [5.3, 8.68, 50, 3], [6.
6, 8.68, 51, 7], [9.8, 8.68, 52, 2], [6.1, 9.1, 53, 2], [8.9, 9.73, 54, 4], [14.3, 13.44, 55, 4]]
```

Figure C.1: Database A

```
Stock B:
Number of different elements = 65, Total number of elements = 373
[[3.4, 0.83, 1, 2], [3.6, 0.83, 2, 3], [3.8, 0.83, 3, 1], [4.9, 0.83, 4, 4], [5.3, 0.83, 5, 8], [5.6, 0.83, 6, 4], [6.8, 0.83,
7, 4], [7.2, 0.83, 8, 6], [8.2, 0.83, 9, 16], [9.8, 0.83, 10, 3], [2.7, 1.15, 11, 10], [2.9, 1.15, 12, 2], [5.1, 1.15, 13, 4],
[5.5, 1.15, 14, 13], [5.8, 1.15, 15, 1], [7.1, 1.15, 16, 6], [7.5, 1.15, 17, 2], [6.9, 1.35, 18, 2], [7.5, 1.35, 19, 1], [2.0,
2.01, 20, 8], [2.2, 2.4, 21, 2], [4.2, 2.4, 22, 6], [6.3, 2.4, 23, 1], [1.9, 3.14, 24, 1], [3.4, 3.88, 25, 3], [4.1, 3.88, 26,
6], [5.1, 3.88, 27, 7], [5.4, 3.88, 28, 9], [3.4, 3.91, 29, 12], [4.3, 3.91, 30, 32], [5.8, 3.91, 31, 18], [8.7, 3.91, 32, 1
6], [17.4, 3.91, 33, 6], [2.6, 4.23, 34, 1], [3.0, 4.23, 35, 1], [3.2, 4.23, 36, 1], [3.4, 4.23, 37, 2], [6.3, 4.23, 38, 1],
[0.9, 4.53, 39, 4], [3.5, 5.38, 40, 2], [4.3, 5.38, 41, 10], [8.6, 5.38, 42, 12], [5.4, 6.43, 43, 8], [11.0, 6.43, 44, 4], [3.
3, 7.27, 45, 21], [5.1, 7.27, 46, 12], [10.0, 7.27, 47, 12], [5.4, 7.68, 48, 4], [5.9, 7.68, 49, 4], [7.8, 7.68, 50, 2], [11.
0, 7.68, 51, 2], [4.4, 7.73, 52, 2], [8.8, 7.73, 53, 5], [9.9, 7.73, 54, 4], [11.0, 7.73, 55, 4], [12.1, 7.73, 56, 8], [13.2,
7.73, 57, 2], [14.1, 7.73, 58, 4], [4.9, 8.68, 59, 4], [6.9, 8.68, 60, 4], [7.9, 8.68, 61, 2], [9.8, 8.68, 62, 2], [6.1, 9.1,
63, 2], [8.9, 9.73, 64, 4], [14.3, 13.44, 65, 4]]
```

Figure C.2: Database B

C.2. Edited databases for optimizing with symmetry

```
Stock A1:
Trusses = 1, Max area = 9, Min area = 1, Min availability = 2
Number of different elements = 27, Total number of elements = 147
[[2.7, 1.15, 1, 6], [5.1, 1.15, 2, 7], [5.7, 1.15, 3, 2], [7.1, 1.15, 4, 4], [2.0, 2.01, 5, 4], [4.2, 2.4, 6, 3], [4.1, 3.88,
7, 3], [5.1, 3.88, 8, 8], [3.4, 3.91, 9, 6], [4.3, 3.91, 10, 24], [5.8, 3.91, 11, 9], [8.7, 3.91, 12, 4], [17.4, 3.91, 13, 3],
[4.3, 5.38, 14, 8], [8.6, 5.38, 15, 4], [5.4, 6.43, 16, 4], [10.9, 6.43, 17, 2], [3.3, 7.27, 18, 13], [5.1, 7.27, 19, 10], [1
0.0, 7.27, 20, 3], [5.2, 7.68, 21, 2], [10.9, 7.68, 22, 2], [6.1, 7.73, 23, 4], [6.9, 7.73, 24, 3], [9.9, 7.73, 25, 2], [11.0,
7.73, 26, 4], [13.2, 7.73, 27, 3]]
```

Figure C.3: Database A1

```
Stock A2:
Trusses = 1, Max area = 15, Min area = 0, Min availability = 2, Min Length = 3
Number of different elements = 38, Total number of elements = 171
[[3.4, 0.83, 1, 4], [3.8, 0.83, 2, 2], [4.2, 0.83, 3, 3], [4.9, 0.83, 4, 3], [5.4, 0.83, 5, 3], [6.8, 0.83, 6, 4], [8.3, 0.83,
7, 5], [5.1, 1.15, 8, 7], [5.7, 1.15, 9, 2], [7.1, 1.15, 10, 4], [4.2, 2.4, 11, 3], [4.1, 3.88, 12, 3], [5.1, 3.88, 13, 8],
[3.4, 3.91, 14, 6], [4.3, 3.91, 15, 24], [5.8, 3.91, 16, 9], [8.7, 3.91, 17, 4], [17.4, 3.91, 18, 3], [4.3, 5.38, 19, 8], [8.
6, 5.38, 20, 4], [5.4, 6.43, 21, 4], [10.9, 6.43, 22, 2], [3.3, 7.27, 23, 13], [5.1, 7.27, 24, 10], [10.0, 7.27, 25, 3], [5.2,
7.68, 26, 2], [10.9, 7.68, 27, 2], [6.1, 7.73, 28, 4], [6.9, 7.73, 29, 3], [9.9, 7.73, 30, 2], [11.0, 7.73, 31, 4], [13.2, 7.7
3, 32, 3], [5.3, 8.68, 33, 1], [6.6, 8.68, 34, 3], [9.8, 8.68, 35, 1], [6.1, 9.1, 36, 1], [8.9, 9.73, 37, 2], [14.3, 13.44, 3
8, 2]]
```

Figure C.4: Database A2

```
Stock A3:
Trusses = 2, Max area = 15, Min area = 1, Min availability = 1
Number of different elements = 30, Total number of elements = 72
[[2.7, 1.15, 1, 3], [5.1, 1.15, 2, 3], [5.7, 1.15, 3, 1], [7.1, 1.15, 4, 2], [2.0, 2.01, 5, 2], [4.2, 2.4, 6, 1], [4.1, 3.88,
7, 1], [5.1, 3.88, 8, 4], [3.4, 3.91, 9, 3], [4.3, 3.91, 10, 12], [5.8, 3.91, 11, 4], [8.7, 3.91, 12, 2], [17.4, 3.91, 13, 1],
[4.3, 5.38, 14, 4], [8.6, 5.38, 15, 2], [5.4, 6.43, 16, 2], [10.9, 6.43, 17, 1], [3.3, 7.27, 18, 6], [5.1, 7.27, 19, 5], [10.
0, 7.27, 20, 1], [5.2, 7.68, 21, 1], [10.9, 7.68, 22, 1], [6.1, 7.73, 23, 2], [6.9, 7.73, 24, 1], [9.9, 7.73, 25, 1], [11.0,
7.73, 26, 2], [13.2, 7.73, 27, 1], [6.6, 8.68, 28, 1], [8.9, 9.73, 29, 1], [14.3, 13.44, 30, 1]]
```

Figure C.5: Database A3

```
Stock A4:
Trusses = 2, Max area = 15, Min area = 0, Min availability = 2, Min Length = 3
Number of different elements = 20, Total number of elements = 62
[[3.4, 0.83, 1, 2], [6.8, 0.83, 2, 2], [8.3, 0.83, 3, 2], [5.1, 1.15, 4, 3], [7.1, 1.15, 5, 2], [5.1, 3.88, 6, 4], [3.4, 3.91,
7, 3], [4.3, 3.91, 8, 12], [5.8, 3.91, 9, 4], [8.7, 3.91, 10, 2], [4.3, 5.38, 11, 4], [8.6, 5.38, 12, 2], [5.4, 6.43, 13, 2],
[3.3, 7.27, 14, 6], [5.1, 7.27, 15, 5], [6.1, 7.73, 16, 2], [11.0, 7.73, 17, 2], [6.6, 8.68, 18, 1], [8.9, 9.73, 19, 1], [14.
3, 13.44, 20, 1]]
```

Figure C.6: Database A4

```
Stock B1:
Trusses = 1, Max area = 10, Min area = 1, Min availability = 2
Number of different elements = 34, Total number of elements = 139
[[2.7, 1.15, 1, 5], [5.1, 1.15, 2, 2], [5.5, 1.15, 3, 6], [7.1, 1.15, 4, 3], [2.0, 2.01, 5, 4], [4.2, 2.4, 6, 3], [4.1, 3.88,
7, 3], [5.1, 3.88, 8, 3], [5.4, 3.88, 9, 4], [3.4, 3.91, 10, 6], [4.3, 3.91, 11, 16], [5.8, 3.91, 12, 9], [8.7, 3.91, 13, 8],
[17.4, 3.91, 14, 3], [4.3, 5.38, 15, 5], [8.6, 5.38, 16, 6], [5.4, 6.43, 17, 4], [11.0, 6.43, 18, 2], [3.3, 7.27, 19, 10], [5.
1, 7.27, 20, 6], [10.0, 7.27, 21, 6], [5.4, 7.68, 22, 2], [5.9, 7.68, 23, 2], [8.8, 7.73, 24, 2], [9.9, 7.73, 25, 2], [11.0,
7.73, 26, 2], [12.1, 7.73, 27, 4], [14.1, 7.73, 28, 2], [4.9, 8.68, 29, 2], [6.9, 8.68, 30, 2], [7.9, 8.68, 31, 1], [9.8, 8.6
8, 32, 1], [6.1, 9.1, 33, 1], [8.9, 9.73, 34, 2]]
```

Figure C.7: Database B1

```
Stock B2:
Trusses = 2, Max area = 10, Min area = 1, Min availability = 2
Number of different elements = 18, Total number of elements = 51
[[2.7, 1.15, 1, 2], [5.5, 1.15, 2, 3], [2.0, 2.01, 3, 2], [5.4, 3.88, 4, 2], [3.4, 3.91, 5, 3], [4.3, 3.91, 6, 8], [5.8, 3.91,
7, 4], [8.7, 3.91, 8, 4], [4.3, 5.38, 9, 2], [8.6, 5.38, 10, 3], [5.4, 6.43, 11, 2], [3.3, 7.27, 12, 5], [5.1, 7.27, 13, 3],
[10.0, 7.27, 14, 3], [12.1, 7.73, 15, 2], [4.9, 8.68, 16, 1], [6.9, 8.68, 17, 1], [8.9, 9.73, 18, 1]]
```

Figure C.8: Database B2

D

Python Script

The optimization tool is scripted in Python, the basis of the optimization script is provided by (He et al., 2019). The script contains 10 functions plus input specifications for the optimization.

Input for the optimization

A stock of reclaimed elements must be defined in matrix form with length, cross section and number of availability for every unique element, the stock is sorted from smallest to largest cross section. Values for penalty's and maximum unity checks are determined after the optimization is performed with all possible combinations of input parameters. The grid for the optimization problem is defined with width, height and scaling factor. Threshold numbers for efficiency and reuse percentage, Lcut, Lshort, maximum tension and compression stresses and maximum number of iterations must be given. The user can determine weather all designs must be plotted or only designs meeting all conditions for efficiency and reuse percentage, intermediate iterations can be plotted as well.

```
#Execution function when called directly by Python
if __name__ =='__main__':
    #           L      A     n
    Members = [[2.0 , 0.80 ,  6],[2.5 , 0.70 ,  6],[1.5 , 0.40 ,  6],[2.8 , 1.70 ,  10],[1.8 , 0.60 ,  8],[3.0 , 1.20 ,  6],
               [1.2 , 0.50 ,  6],[4.2 , 2.00 ,  8]]

    Members.sort(key = lambda element: (element[1], element[0]))
    for i, s in enumerate(Members):
        s.insert(2,1+i)
    print(Members)

    jc = np.array([1.0,3.0,5.0])
    cspf = np.array([1.0,3.0])
    Acspf = np.array([0.25,0.50,1.0])
    Nmp = np.array([1.0,3.0])
    ANmp = np.array([0.25,0.5,1.0])
    maxUC = np.array([1.0,2.0])

    results = len(jc)*len(cspf)*len(maxUC)*len(Acspf)*len(Nmp)*len(ANmp)*len(pmincu)*len(pminfinalcu)
    plot = 0
    print(results, 'possible results')
    print('-----------------------------------------------------------------------------------------------------------------------------------------')

    for m, n in enumerate(pmincu):
        for o, p in enumerate(pminfinalcu):
            for a, b in enumerate(jc):
                for c, d in enumerate(cspf):
                    for e, f in enumerate(maxUC):
                        for g, h in enumerate(Acspf):
                            for i, j in enumerate(Nmp):
                                for k, l in enumerate(ANmp):
                                    plot += 1
                                    trussOpt(width = 12, height = 6, st = 1.0, sc = 0.70, jc = b, cspf = d, minCU = 0.70,
                                             maxUC = f, Lshort = 0.1, Lcut = 0.3, Load = -1.0, Stock = Members, sf = 1.0,
                                             Acspf = h, minfinalCU = 0.30, Nmp = j, ANmp = l, minReuse = 0.80, plot = plot,
                                             max_iterations = 100, plot_all = False, plot_intermediate = False)
```

Figure D.1: Input for the optimization

Main Function

The optimization starts with creating the optimization grid. Next, load and support conditions must be defined by the user. 0 And 1 represent the degree of freedom in x and y direction. The ground structure is created after which the PSML is obtained with the PossibleStockMemberList function.

```
#Main function
def trussOpt(width, height, st, sc, jc, cspf, minCU, maxUC, Lshort, Lcut, Load, Stock, sf, Acspf, minfinalCU, Nmp, ANmp,
             minReuse, plot, max_iterations, plot_all, plot_intermediate):
    width = int(width * sf)
    height = int(height * sf)
    poly = Polygon([(0, 0), (width, 0), (width, height), (0, height)])
    convex = True if poly.convex_hull.area == poly.area else False
    xv, yv = np.meshgrid(range(width+1), range(height+1))
    pts = [Point(xv.flat[i], yv.flat[i]) for i in range(xv.size)]
    Nd = np.array([[pt.x, pt.y] for pt in pts if poly.intersects(pt)])
    dof, f, PML = np.ones((len(Nd),2)), [], []

    #Load and support conditions (roller support)
    for j, nd in enumerate(Nd):
        if (nd == [0,height]).all():
            dof[j,:] = [1,0]
        elif (nd == [width/2, 0]).all():
            dof[j,:] = [0,0]
        else:
            pass
    # Load
    f = np.zeros(len(Nd)*2)
    for i, nd in enumerate(Nd):
        if (nd == [width,height]).all():
            f[i*2+1] += Load
    #Create the 'ground structure' and PSML
    for i, j in itertools.combinations(range(len(Nd)), 2):
        dx, dy = abs(Nd[i][0] - Nd[j][0]), abs(Nd[i][1] - Nd[j][1])
        if gcd(int(dx), int(dy)) == 1 or jc != 0 or jc == 0:
            seg = [] if convex else LineString([Nd[i], Nd[j]])
            if convex or poly.contains(seg) or poly.boundary.contains(seg):
                PML.append( [i, j, np.sqrt(dx**2 + dy**2), False, False] )

    PSML = PossibleStockMemberList(PML,Stock,Lshort,Lcut,sf)
```

Figure D.2: Create optimization grid, ground structure and PSML

Initial connectivity must be defined by the user, members included are 'connected nodes Cn'. If a structural stable initial connectivity is obtained the member adding loop starts. First, the optimization with the dual interior point method is performed, next, the optimization of assignment of available cross sections from stock is executed. Virtual displacements are recalculated and penalty's for inefficient and new members are determined and applied. Intermediate iterations can be plotted if the user desires.

```
    dof = np.array(dof).flatten()
    f = [f[i:i+len(Nd)*2] for i in range(0, len(f), len(Nd)*2)]

    # Initial connectivity
    for pm in [p for p in PSML if p[2] <= 4.5 * sf]:
        pm[3] = True

    maxArea = np.max(np.array(Stock)[:,1])
    #Start the 'member adding' loop
    for itr in range(1, (max_iterations + 1)):
        plotting = False
        iteration = [itr]
        PSMLarray = np.array(PSML)
        Cn = PSMLarray[PSMLarray[:,3] == True]

        # Optimize with dual interior point method
        q, aOpt = solveLP(Stock, Nd, Cn, f, dof, st, sc, jc, iteration, sf, cspf, Acspf)
        # Optimize assignment of available members from stock
        finalArea, finalUC, vol, new_mems, Reuse_percentage, reuse_list = Assigning_Available_Members(aOpt, q, Stock, Cn, q,
                                                                                                       maxUC, st, sc, sf,maxArea)
        # Calculate virtual displacements
        u = np.array([calcu(Nd, Cn, finalArea, f, dof)])
        # Calculate penalty's
        calcPenalties(PSML, aOpt, finalArea, cspf, sf, minCU, Stock, iteration, Acspf, minfinalCU, new_mems, Nmp, ANmp)
        # Plot intermediate iterations
        if plot_intermediate == True:
            final_plot = False
            print("Itr: %d, vol: %f, mems: %d" % (itr, vol, len(Cn)))
            plotTruss(width, height, Nd, Cn, finalArea, q, finalUC, minCU, minfinalCU, final_plot, new_mems, maxArea * 1e-2,
                      ""Itr:" + str(itr)", False)
```

Figure D.3: Optimize, assign cross sections, recalculate displacements and determine and apply penalty's

Violation of virtual displacements is checked and the structure is verified for efficiency, maximum and minimum unity check and reuse percentage. If all requirements are met, 'plotting' is 'True' and the design will be plotted. If the final iteration is reached and all designs must be plotted, 'plotting' will be 'True' as well. Information about the design is obtained and presented to the user.

```
        # Stop optimization if no violation of strain occurs, min average UC, max UC and min Reuse % is achieved
        if stopViolation(Nd, PSML, PSMLarray, dof, st, sc, u, jc):
            UCbig = []
            for a, check in enumerate(finalArea):
                if check > 1e-2*maxArea:
                    UCbig.append(finalUC[a])
            Reuse_mems = Reuse_percentage*len(UCbig)
            UC_Reused = (np.sum(UCbig) - (len(UCbig)-Reuse_mems)*1.0) / Reuse_mems
            #if np.sum(UCbig)/len(UCbig) > minCU:         # average UC
            if UC_Reused >= minCU:                        # average Reused UC
                if np.max(UCbig) <= 1.0:
                    if np.min(UCbig) >= 0.1:
                        if Reuse_percentage >= minReuse: plotting = True
        if plot_all == True and itr == max_iterations:
            plotting = True
        if plotting == True:
        print('Nodes: %d Members: %d plot: %s' % (len(Nd), len(PSML), plot))
        print("jc = %s, cspf = %s, Acspf = %s, Nmp = %s, ANmp = %s, maxUC = %s" % (jc, cspf, Acspf, Nmp, ANmp, maxUC))
        print('.                       .                       .')
        print("Itr: %d, vol: %f, mems: %d" % (iteration[0], vol, len(Cn)))
        print('Plotted members = ', numberbigger)
        print('average UC = ', np.sum(UCbig)/len(UCbig))
        print('average UC reused members = ', UC_Reused)
        print('Reuse percentage = ',Reuse_percentage*100, '%')
        final_plot = True
        plotTruss(width, height, Nd, Cn, finalArea, q, finalUC, minCU, minfinalCU, final_plot, new_mems, maxArea * 1e-2,
                  "Finished", False)
        print('List of reused elements: ', reuse_list)
    else:
        print('No result found for plot ',plot)
```

Figure D.4: Plot designs meeting all requirements for strain, UC, and reuse percentage

Possible Stock Member List

The PSML is created by filtering the PML with lengths available from stock. If an element from stock fits at a position in the PML, the index of this element is added to the member in the PML and the second 'False' statement changes to 'True', implicating it will be included in the PSML. A member in the PSML is described by two nodes, lenght, 2 'False/True' statements, 4 zeros for applying penalty's and element indices from stock. Members included in the initial connectivity and added during the member adding process are connected nodes Cn and have two 'True' statements.

$$PSM = [nd, nd, L, False, True, Acsp, csp, ANmp, Nmp, element\_indices] \tag{D.1}$$

$$Cn = [nd, nd, L, True, True, Acsp, csp, ANmp, Nmp, element\_indices] \tag{D.2}$$

```
#Define possible stock member list PSML
def possibleStockMemberList(PML,Stock,Lshort,Lcut,sf):
    #Add zeros for Acsp, csp, ANmp, Nmp
    for m, pm in enumerate(PML):
        pm.extend(np.zeros(4))
    #Create PSML
    Lpm = []
    for i, pm in enumerate(PML):
        for j, sk in enumerate(Stock):
            if ( pm[2]>=(sk[0]*sf-Lcut*sf) ) & ( pm[2]<=(sk[0]*sf+Lshort*sf) ):
                # If member from stock fits in PML, append index and second statement = True
                pm[4] = True
                pm.append(sk[2])
        Lpm.append(len(pm))
    maxLp = np.max(Lpm)
    # All members in PML should have the same list length
    for i, Lpmi in enumerate(Lpm):
        dif = maxLp - Lpmi
        for j in range(dif):
            PML[i].append(0)
    PSML = []
    # Append all members in PML with True statement to PSML
    for i, pm in enumerate(PML):
        if pm[4] == True:
            PSML.append(pm)
    return PSML
```

Figure D.5: Possible Stock Member List

Equilibrium matrix B

Equilibrium of forces and geometry is calculated with matrix B containing information about nodes, connected nodes, length and degrees of freedom.

```
#Calculate equilibrium matrix B
def calcB(Nd, Cn, dof):
    m, n1, n2 = len(Cn), Cn[:,0].astype(int), Cn[:,1].astype(int)
    l, X, Y = Cn[:,2], Nd[n2,0]-Nd[n1,0], Nd[n2,1]-Nd[n1,1]
    d0, d1, d2, d3 = dof[n1*2], dof[n1*2+1], dof[n2*2], dof[n2*2+1]
    s = np.concatenate((-X/l * d0, -Y/l * d1, X/l * d2, Y/l * d3))
    r = np.concatenate((n1*2, n1*2+1, n2*2, n2*2+1))
    c = np.concatenate((np.arange(m), np.arange(m), np.arange(m), np.arange(m)))
    return sparse.coo_matrix((s, (r, c)), shape = (len(Nd)*2, m))
```

Figure D.6: Equilibrium Matrix B

Solve Linear Programming

The objective of the optimization is to minimize volume. Penalty's are added to the length of a Cn to increase volume share of this element. Constraints are added for a minimum cross section of 0 and a maximum cross section of the maximum cross section available from stock. Optimal geometry is defined with internal loads and cross sections for all Cn.

```
#Solve linear programming problem
def solveLP(Stock, Nd, Cn, f, dof, st, sc, jc, iteration, sf, cspf, Acspf):
    B = calcB(Nd, Cn, dof)

    # Add penalty's to length of a Cn
    # Cn = [Nd, Nd, L, True, True, Acsp, csp, Nmp, index members]
    l = [col[2] + jc + col[5] + col[6] + col[7] + col[8] for col in Cn]

    # Optimize with Dual Interior Point Method
    Stockarray = np.array(Stock)
    a = cvx.Variable(len(Cn))
    obj = cvx.Minimize(np.transpose(l) * a)
    q, eqn, cons= [], [], [a>=0]
    for k, fk in enumerate(f):
        q.append(cvx.Variable(len(Cn)))
        eqn.append(B * q[k] == fk * dof)
        cons.extend([eqn[k], q[k] >= -sc * a, q[k] <= st * a])
        cons.extend([a<=np.max(Stockarray[:,1])])
    prob = cvx.Problem(obj, cons)
    vol = prob.solve()
    q = [np.array(qi.value).flatten() for qi in q]
    aOpt = np.array(a.value).flatten()
    u = [-np.array(eqnk.dual_value).flatten() for eqnk in eqn]

    return q, aOpt
```

Figure D.7: Solve linear programming with dual interior point method

Capacity Utilization

For every element available cross sections from stock are retrieved with the index numbers added to the connected nodes (Cn), threshold number 1e-2*maxArea is added or small internal loads. For every cross section, capacity utilization is calculated. Most efficient cross sections are assigned and if no cross section available result in a unity check equal or lower than 1.0, the element retains it optimized cross section and is a new member. Members to which threshold number 1e-2*maxArea is assigned retain their optimized cross section as well. A memberlist is created with indices of members from stock or indices of -1 and 0, -1 for new members for which no cross sections resulting in UC ≤ 1.0 could be assigned and 0 for new members with cross sections ≤ 1e-2*maxArea.

```
def capacityUtilization(Stock, Cn, qflat, maxUC, st, sc, sf,maxArea):
    memberlist, av_mems, av_mems_CUf = [],[],[]
    assigned_cs = np.zeros(len(qflat))

    for z, qz in enumerate(qflat):
        mems = Cn[z][9:len(Cn[z])]
        av_mems.append(mems)
        aa, UC, CUf, CU = [],[],[],[]
        for i, m in enumerate(mems):
            if m > 0:
                aa.append(Stock[int(m)-1][1])
        aa.append((1e-2)*maxArea)
        for j, area in enumerate(aa):
            UC.append (max( qz/(st*area), -qz/(sc*area)))
            if UC[j] >= 1.0:
                CUf.append(UC[j]/1.0)
            else:
                CUf.append(1.0/UC[j])
            if UC[j] > maxUC:
                CUf[j] = 100 + UC[j]

        # Determine most efficient cross section
        av_mems_CUf.append(CUf)
        minCUf = np.min(CUf)
        min_pos_CUf = [i for i, x in enumerate(CUf) if x == minCUf]
        # Create member list
        mem_pos = min_pos_CUf[0]
        if mem_pos < len(mems):
            assigned_cs[z] = aa[mem_pos]
            if minCUf < 100: # UC < maxUC
                memberlist.append([mems[mem_pos],True])
            else: # new member
                memberlist.append([-1, True])
        else: # new member
            memberlist.append([0,True])

    return av_mems, av_mems_CUf, assigned_cs, memberlist
```

Figure D.8: Assigning most efficient cross sections from stock

Assign Available Cross Sections

Optimal assigned cross sections with the CU function are retrieved to determine cross section penalty's for new members in a later stadium. Capacity utilization factors for all possible available cross sections for each elements are used during the assignment of cross sections taking availability into account

```
def assignAvailableMembers(aOpt, q, Stock, Cn, qflat, maxUC, st, sc, sf,maxArea):

    qflat = np.array(q).flatten()
    # Assign most efficient cross sections
    av_mems, av_mems_CUf, assigned_cs, memberlist = CapacityUtilization(Stock, Cn, qflat, maxUC, st, sc, sf,maxArea)

    Optimal_Assigned = np.zeros(len(aOpt))
    for i, m in enumerate(memberlist):
        if m[0] > 0: # Reused
            Optimal_Assigned[i] = Stock[int(m[0])-1][1]
            m.append(i)
        elif m[0] < 0: # UC > maxUC -> New
            Optimal_Assigned[i] = assigned_cs[i]
            m[0] = 0
        else: # New
            Optimal_Assigned[i] = aOpt[i]
```

Figure D.9: Retrieve optimal assigned cross sections with CU

De function check_Violation_Availability is used to check how many times each element is assigned. First the memberlist is sorted from smallest to largest index number, next the list is iterated and when the index number of the current member is higher than the previous index number, number of reuse can be retrieved with the iteration number. If the number of reuse is higher than the number of availability for a certain index, the violating index is added to the vioAv. Violating indices receive a 'False' statement in the memberlist.

```
def check_Violation_Availability(Stock,memberlist):
    # Check how many times each element is reused
    sorted_memberlist = memberlist[:]
    sorted_memberlist.sort(key = lambda element: (element[0]))
    check = sorted_memberlist[0][0]
    finalcheck = sorted_memberlist[-1:][0][0]
    reuse_list, vioAv = [],[]
    count = 0
    for i, member in enumerate(sorted_memberlist):
        if member[0] > int(check):
            number_of_reuse = i - count
            reuse_list.append([check,number_of_reuse])
            # check if reuse number is higher than available
            if number_of_reuse > Stock[int(check-1)][3]:
                # append index of violating member
                if check > 0: vioAv.append(int(check))
            count = i
            check = member[0]
        if i == (len(sorted_memberlist)-1):
            number_of_reuse = i-count+1
            reuse_list.append([check,number_of_reuse])
            if number_of_reuse > Stock[int(check-1)][3]:
                if int(check) > 0: vioAv.append(int(check))

    # Set violating index in memberlist to false
    for i, member in enumerate(memberlist):
        if member[0] in np.array(vioAv):
            member[1] = False
    return reuse_list
```

Figure D.10: Check violation of availability

The optimization of assigning available cross sections as efficient as possible is an iterative process. The memberlist is iterated from lowest index, with smallest available cross section, to highest index, with largest available cross section.

```
maxitr = int(np.max(np.array(Stock)[:,2]))
check = True
# Assign available cross sections as efficient as possible
for i in range(maxitr):
    if check == True:
        check_Violation_Availability(Stock,memberlist)
    index_Cs_Stock = i + 1
    CUf_vio = []
    # Check violation of all members
    vio_Mems = [d for d in memberlist if d[0]==index_Cs_Stock]
```

Figure D.11: Iterate through elements from stock from smallest to largest cross section

Every iteration i violation of the ith index is checked, if violation occurs only the most efficient members retain the index of their assigned cross section. If available, indices of larger cross sections are assigned to the other less efficient members. If no violation of availability of a certain member occurs, nothing changes thus the check statement changes to 'False' and availability of elements with check_Violation_Availability is not checked during the next iteration.

```
    # Only members present more than once can have violation of availability
    if len(vio_Mems) > 1:
        if np.all(vio_Mems[1]) == False:
            # find index of violating members in memberlist
            index_vio_Mem = np.array(vio_Mems)[:,2]
            # Retrieve CUf for these violating members)
            index_vio_CUf = []
            for i, index in enumerate(index_vio_Mem):
                index_vio_CUf.append([c for c, x in enumerate(av_mems[int(index)]) if x == index_Cs_Stock][0])
            for j, m in enumerate(index_vio_Mem):
                CUf_vio.append([av_mems_CUf[int(m)][index_vio_CUf[j]],int(m)])
            # sort optCU from lowest to highest
            def sort_key2(CUf_vio):
                return CUf_vio[0]
            CUf_vio.sort(key=sort_key2, reverse=False)
            # assign only n most effective ones
            n = Stock[index_Cs_Stock-1][3]
            for o, cufvio in enumerate(CUf_vio):
                index_opt_mem = cufvio[1]
                if o < n:
                    memberlist[index_opt_mem][1] = True
                else:
                    # check if larger cross sections are available
                    if index_Cs_Stock < np.max(av_mems[index_opt_mem]):
                        # Define index of new cross section in available members
                        index_new_mem = [y for y, x in enumerate(av_mems[index_opt_mem]) if x == index_Cs_Stock][0] + 1
                        # Assign new member
                        memberlist[index_opt_mem][0] = av_mems[index_opt_mem][int(index_new_mem)]
                    else: # it is a new member
                        memberlist[index_opt_mem][0] = 0
                        memberlist[index_opt_mem][1] = True
        check = True
    else: check = False #nothing has changed
```

Figure D.12: Assign available cross sections as efficient as possible

When optimization of the memberlist is completed, a list with final assigned cross sections is created. Members with an index > 0 in the memberlist are reused elements with assigned cross sections, members with an index of 0 are new members. New members with a cross section > 1e-2*maxArea are added to the new_mems list with optimized and optimal assigned cross section to determine cross section penalty's for new members. Volume, reuse percentage and final unit check are determined and returned.

```
reuse_list = check_Violation_Availability(Stock,memberlist)
finalArea = np.zeros(len(aOpt))
new_mems = []
Reused = 0
New = 0
for i, m in enumerate(memberlist): # check index
    if m[0] > 0: # reused member
        finalArea[i] = Stock[int(m[0]-1)][1]
        Reused += 1
    else: # new member
        finalArea[i] = aOpt[i]
        if aOpt[i] >= (maxArea*(1e-2)):
            new_mems.append([i,aOpt[i],Optimal_Assigned[i]])
            New += 1
Reuse_percentage = Reused/(Reused+New)
vol = 0
finalUC = []

for i, (area,load) in enumerate((zip(finalArea,qflat))):
    vol += Cn[i][2]/sf*area
    finalUC.append(max( load/(st*area), -load/(sc*area)))

return finalArea, finalUC, vol, new_mems, Reuse_percentage, reuse_list
```

Figure D.13: Obtain the final lists with reused and new element, reuse percentage, volume and unity checks

Determine and Apply penalty's

Cross section, accumulating cross section, new member and accumulating new member penalty's are determined for new members with a cross section > 1e-2*maxArea. The capacity utilization factor (CUf) is determined for each new member based on the optimized and initial assigned cross section from stock with the CU function. Iterations from which penalty's should be applied must be defined by the user.

```
#Assign cross section penalties to PSML
def calcPenalties(PSML, aOpt, aAs, cspf, sf, minCU, Stock, iteration, Acspf, minfinalCU, new_mems, Nmp, ANmp):

    # Cn = [Nd, Nd, L, True, True, Acsp, csp, ANmp, Nmp, member index]

    # New member penalty and cross section penalties for new members
    # new = [index, aOpt, aOpt_assigned]
    new_csp = np.zeros(len(aOpt))
    new_Acsp = np.zeros(len(aOpt))
    new_Nmp = np.zeros(len(aOpt))
    new_ANmp = np.zeros(len(aOpt))

    for i, new in enumerate(new_mems):
        CUf = np.min([ new[1]/new[2] , 1/(new[1]/new[2]) ])
        # Cross section penalty
        if iteration[0] >= 1:
            if CUf <= minCU:
                new_csp[new[0]] = (minCU/CUf)*cspf
        # Accumulating cross section penalty
        if iteration[0] >= 3:
            if CUf <= minfinalCU:
                new_Acsp[new[0]] += (minfinalCU/CUf)*Acspf
        # New member penalty
        if iteration[0] >= 3:
            new_Nmp[new[0]] = Nmp
        # Accumulating new member penalty
        if iteration[0] >= 15:
            new_ANmp[new[0]] = ANmp
```

Cross section and accumulating cross section penalty's are determined for reused elements, penalty's are determined based on the optimized and assigned cross section. The capacity utilization factor (CUf) is calculated, iterations from which penalty's should be applied must be defined by the user.
Penalty's are applied by adding the penalty to the correct location in the PSML, penalty's are only applied to Cn.

```
    # Cross section penalties reused members
    csp = np.zeros(len(aOpt))
    Acsp = np.zeros(len(aOpt))

    for i, (o,a) in enumerate((zip(aOpt,aAs))):
        CUf = np.min([ o/a , 1/(o/a) ])
        # Cross section penalty
        if CUf <= minCU:
            csp[i] = (minCU/CUf)*cspf
        # Accumulating cross section penalty for low CU
        if iteration[0] >= 3:
            if CUf <= minfinalCU:
                Acsp[i] = (minfinalCU/CUf)*Acspf
        # Accumulating cross section penalty for UC > maxUC
        if iteration[0] >= 10:
            if (o > a)&(CUf > minfinalCU):
                Acsp[i] = (o/a-1)*Acspf

    counter = 0
    # Assign cross section and New member penalties
    for i, PSM in enumerate(PSML):
        if PSM[3] == True:
            PSM[5] += Acsp[counter]
            PSM[5] += new_Acsp[counter]
            PSM[6]  = csp[counter]
            PSM[6] += new_csp[counter]
            PSM[7] += new_ANmp[counter]
            PSM[8]  = new_Nmp[counter]
            counter += 1
```

Figure D.14: Determine and apply cross section and new member penalty's

Recalculate virtual displacements

Stiffness matrices are calculated for every element with their assigned or optimized cross sections after which the complete stiffness matrix is assembled. Every node in the design domain has unique degrees of freedom, with these unique degrees of freedom elements are placed at the right location in the complete stiffness matrix. With the support reactions, loads and inverse stiffness matrix virtual displacements are recalculated.

```
#Calculate virtual displacements
def calcu(Nd, Cn, a, f, dof):
    ndof = 2
    u2 = np.zeros(len(Nd)*ndof)
    K = np.array(len(u2)*[np.zeros(len(u2))])
    E = 1
    Kele = []

    # Calculate stiffness matrix for each element
    for z, m in enumerate(Cn):
        xi, xj, yi, yj, lx, ly = [],[],[],[],[],[]
        xi = Nd[ int(m[0]) ][0]
        yi = Nd[ int(m[0]) ][1]
        xj = Nd[ int(m[1]) ][0]
        yj = Nd[ int(m[1]) ][1]
        lx = (xj-xi)/ m[2]
        ly = (yj-yi)/ m[2]
        Kele.append( (a[z]*E/Cn[z][2]) * np.array([[ lx**2,  lx*ly, -lx**2, -lx*ly],
                                                   [ lx*ly,  ly**2, -lx*ly, -ly**2],
                                                   [-lx**2, -lx*ly,  lx**2,  lx*ly],
                                                   [-lx*ly, -ly**2,  lx*ly,  ly**2]]) )
```

```
    # Calculate complete stiffness Matrix
    for i in range(len(Cn)):
        dof1 = Cn[i][0] * ndof
        dof2 = Cn[i][0] * ndof + 1
        dof3 = Cn[i][1] * ndof
        dof4 = Cn[i][1] * ndof + 1
        eldofs = [dof1,dof2,dof3,dof4]
        eldofs1 = np.arange(4)
        for (p, d), (k, v) in zip(itertools.product(eldofs, eldofs), itertools.product(eldofs1, eldofs1)):
            K[int(p)][int(d)] += Kele[i][k][v]
    # Check support reactions
    for i in range(len(dof)):
        if dof[i] == 0:
            for j in range(len(dof)):
                K[i][j] = 0
                K[j][i] = 0
                K[i][i] = 1
    # Calculate displacements
    Kinv = np.linalg.inv(K)
    force = f[0]
    u2 = Kinv.dot(force)
    return(u2)
```

Figure D.15: Recalculationg of virtual displacements with Matrix Method

Check violation of virtual strain

Virtual strain is calculated for members not included in the Cn, strain is calculated with the recalculated virtual displacements. The members with the largest violation of virtual strain are added from the PSML to the Cn by changing the 1st 'False' statement to 'True'.

```
#Check dual violation
def stopViolation(Nd, PSML, PSMLarray, dof, st, sc, u, jc):
    lst = np.where(PSMLarray[:,3]==False)[0]
    Cn = PSMLarray[lst]
    l = Cn[:,2]
    B = calcB(Nd, Cn, dof).tocsc()
    y = np.zeros(len(Cn))
    for uk in u:
        yk = np.multiply(B.transpose().dot(uk) / l, np.array([[st], [-sc]]))
        y += np.amax(yk, axis=0)
    vioCn = np.where(y>1.0001)[0]
    vioSort = np.flipud(np.argsort(y[vioCn]))
    num = ceil(min(len(vioSort), 0.05*max( [len(Cn)*0.05, len(vioSort)])))
    for i in range(num):
        PSML[lst[vioCn[vioSort[i]]]][3] = True
    return num == 0
```

Figure D.16: Add members with the largest violation of virtual strain to the Cn

Plot truss

Size of the plot must be defined by the user. Colour of each elements is determined based on efficiency, new elements are shown as dotted lines. Separate functions are defined to create a legend.

```python
#Visualize truss
def plotTruss(width, height, Nd, Cn, a, q, finalUC, minCU, minfinalCU, final_plot, new_mems, threshold, str, update = True):
    plt.ion() if update else plt.ioff()
    plt.clf(); plt.draw()
    plt.title(str)
    plt.figure(figsize=(width/height*7, 7))
    plt.axis('equal')
    plt.margins(x=0)
    tk = 5 / max(a)

    dashed_plot = np.zeros(len(Cn))
    for i, new in enumerate(new_mems):
        dashed_plot[new[0]] = 1

    for i in [i for i in range(len(a)) if a[i] >= threshold]:
        if all([finalUC[i] <= minfinalCU]): c = 'r'
        elif all([(finalUC[i] > minCU) & (finalUC[i] < 1.01)]): c = 'g'
        elif all([finalUC[i] > 1.01]): c = 'm'
        else: c = 'b'
        pos = Nd[Cn[i, [0, 1]].astype(int), :]
        if dashed_plot[i] == 0:
            plt.plot(pos[:, 0], pos[:, 1], c, linewidth = a[i] * tk)
        else:
            plt.plot(pos[:, 0], pos[:, 1], c, ls=('dashed'), linewidth = a[i] * tk)

    # legend
    y1,y2,y3,y4,y5,x = 0,0,0,0,0,width
    plt.plot(y1, c = 'g', label = ('UC > %s' % (minCU)) )
    plt.plot(y2, c = 'r', label = ('UC < %s' % (minfinalCU)) )
    plt.plot(y3, c = 'b', label = ('%s < UC < %s' % (minfinalCU, minCU)))
    plt.plot(y4, c = 'm', label = ('UC > 1.0'))
    plt.plot(y5, c = 'k', ls=('dashed'), label = ('New member'))
    plt.legend(bbox_to_anchor =(1.0, -0.1), ncol=5, prop={'size': 8})
    plt.close(1)
    plt.pause(0.01) if update else plt.show()
```

Figure D.17: Plot the designs

Required packages

A list of required packages for the optimization script is given below, specific versions of numpy and cvxpy are required.

```python
from math import gcd, ceil
import itertools
from scipy import sparse
import numpy as np
import cvxpy as cvx
import matplotlib.pyplot as plt
from shapely.geometry import Point, LineString, Polygon
from pprint import pprint
import itertools
from pprint import pprint
from scipy.sparse.linalg import inv
from scipy.linalg import solve
import time

# Numpy version 1.19.2
# Cvxpy version 0.4.11
```

Figure D.18: Required packages