Printing reinforcement steel

A study towards optimised, additive manufactured steel for reinforced concrete

In partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE IN CIVIL ENGINEERING

M.P. Drillenburg ook genaamd Lelijveld

to be defended publicly on Friday August 14, 2020 at 14:30

GRADUATION COMMITTEE
Delft University of Technology:
prof. dr. ir. J.G. Rots (chair)
dr. ir. M.A.N. Hendriks
ir. L.P.L van der Linden
van Rossum Raadgevende Ingenieurs B.V.:
ir. N.A.J. Bartels

Faculty of Civil Engineering and Geosciences - Delft University of Technology

Abstract

In light of the global attempts to reduce material use by the construction industry, this research focuses on combining topology optimisation with additive manufacturing of steel. It is investigated whether an automated procedure can be developed to generate reliable strut and tie models for reinforced concrete elements, while satisfying the constraints that apply to 3D-printing using the Wire and Arc Additive Manufacturing(WAAM) technique.

Additive manufacturing offers a fully automated production process where a large freedom in form can be achieved. Topology optimisation concerns with finding a good material distribution within a prescribed domain. A literature review was performed on current developments regarding both subjects. It was found that the WAAM-technique is very suitable for printing reinforcement designs. Sufficiently large models can be printed, and material properties can be achieved that match the properties of traditional reinforcement steel. This manufacturing process is expected to produce functional structures that can readily be used as reinforcement steel in buildings. Two main manufacturing constraints should be accounted for during design of the model. A minimum member inclination and a minimum member diameter are both expected to be necessary to ensure a smooth printing process.

Several different topology optimisation algorithms are discussed in the second part of the literature review, and it is determined which algorithm is most suitable to continue with in the rest of this research. Examples are presented that explain the functionality of three important optimisation schemes: Bi-directional Evolutionary Structural Optimisation(BESO, Solid Isotropic Material with Penalisation(SIMP) and Ground Structure Optimisation(GSO). It was found that all three can be used to analyse reinforced concrete. Each algorithm has advantages and disadvantages, so there is no obvious best choice. However, motivated by the easy access to member forces and availability of a very good Python implementation, it is chosen to use GSO for the remainder of this research.

This Python script was modified to include the constraints that come with an additive manufacturing process. It was found that the minimum member inclination can straightforwardly be included. The new function that was proposed allows the user to specify a minimum inclination, and ensures that no members are generated within the design domain that violate this minimum angle. Experimenting with this new function revealed cases where material use increased significantly when this function was used. This lead to development of an alternative procedure to ensure a printable design. In this alternative procedure, an optimisation without any angle constraint is performed first. Then, in the form of a post-processing script, The complete model is rotated around two separate axes in an attempt to find a suitable printing orientation.

The third and final proposition that was done in this part, consists of a post-processing script for the minimum member diameter. Including this minimum diameter in the optimisation would require rigorous changes to the optimisation script. Therefore it was chosen to investigate the performance of this post-processing script first.

The case study that was performed in the third part of this research, proved that this post-

processing script for member diameter is sufficiently efficient for practical implementation. Together with the ability to slightly suppress the amount of members that are generated in the design domain, the printing constraint for minimum diameter could relatively easily be enforced. A bigger challenge lies within ensuring the minimum member inclination. The 60° minimum that was set, proved to be very harsh on the solution space. In the example from the case study, no printable model could be generated without significantly reducing the material efficiency. However, it is argued that this minimum inclination constraint can possibly be relieved by recent developments in additive manufacturing techniques. An example of this could be a rotating printing surface, that has the potential to remove this angle constraint completely.

Overall, the experience of combining topology optimisation, additive manufacturing and strut and tie modelling has been predominantly positive throughout this research. The combination of a state of the art manufacturing technique and a more performance driven design process with a labour intensive traditional calculation procedure has shown promising first results. In the example in this research, 30% less material was required to accommodate the tensile forces in the concrete.

Contents

Chapter 1

Introduction

Motivated by the large environmental impact [1] and relatively high cost, the construction industry is being forced to limit the use of structural steel. Therefore, clever designs are required that minimise the amount of steel in a structure. However, a higher material efficiency should never go at the expense of structural safety. Traditionally, load bearing or stability elements are designed to be bigger than necessary. Driven by the advance of computation power, the ability to accurately estimate the safety and stability of structures has greatly increased since the turn of the century. Precise knowledge on the internal force distribution allows engineers to design structures with high material efficiency, while maintaining the level of structural safety that is required by the building codes.

Structural components that remain relatively difficult to analyse to date, are reinforced concrete elements. Non-linear stress distributions are observed in the so-called *disturbed regions*, where Bernoulli's hypothesis is not valid. Precise calculation of these stresses can be time-consuming and therefore expensive, so often simplifications are used. The strut-and-tie model [2], prescribed by the Eurocode[1] as a valid method to analyse these regions, is often used as such a simplification. Although this method allows engineers to design reinforcement in a fairly straightforward way, the efficiency of the final design relies heavily on the chosen layout of the struts and ties. It is for this reason that researchers have shown interest in finding automated procedures for compiling strut-and-tie models. It was proven that topology optimisation methods show great potential for this purpose [3;4].

Topology optimisation methods attempt to find the most optimal layout of material in a prescribed domain, subjected to a specific set of boundary conditions [5;6]. Producing fluent, irregular shaped geometries, designs from optimisation methods often pose challenges for traditional manufacturing processes. With the advances in additive manufacturing processes, or 3D-printing, a solution to this issue may be within reach. For instance, the optimised glass swing by Snijder, van der Linden, Goulas et al. [7], seen in Figure 1.1, showcases the promising potential of combining topology optimisation with 3D-printing.

Although additive manufacturing techniques allow larger form freedom than most traditional manufacturing processes, the printing process poses new design challenges. In the example of topology optimisation results, often manual geometry changes have to be made to successfully print the object. This generally has to do with the orientation of the object, and the angle at which the printer can function. This raises questions regarding the efficiency, or optimality, of the final product. It is obvious that adjusting geometry from an optimised result is counterproductive and should therefore be avoided. Therefore it is justifiable to address these concerns through research, to get a better understanding of the possibilities.

[1] NEN-EN 1992-1-1 Section 6.5

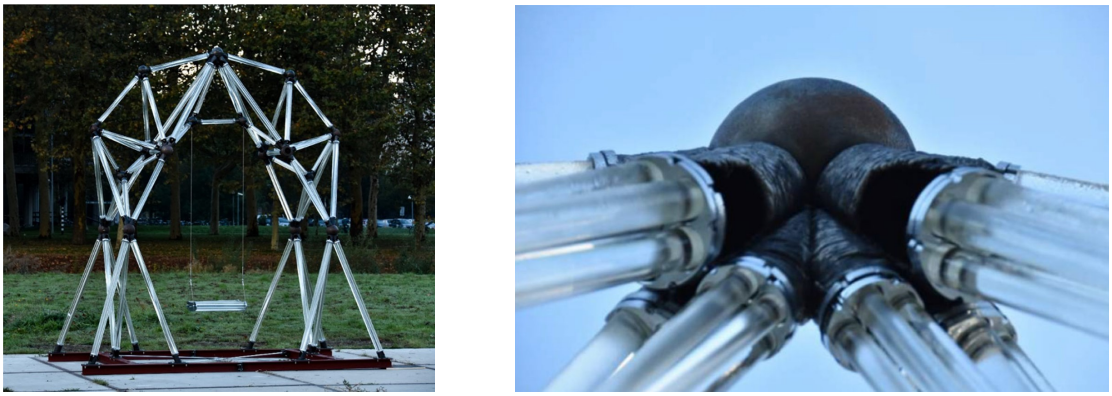


Figure 1.1: Optimised glass swing with 3D-printed steel nodal connections, taken from Snijder, van der Linden, Goulas et al. [7]

1.1 Additive Manufacturing of Steel

Additive manufacturing, also called 3D-printing, is a manufacturing technique that started gaining popularity around 1980 [8]. The idea for this technique had been around for much longer, the first patent dating back to 1926 [9]. Over the years different techniques emerged, but *fused deposition modeling*, or FDM, appeared to be the most feasible for polymer-based commercial 3D-printers. FDM is a relatively simple process, in which printing material is fed through a heated coil. This coil melts the material, and can be moved above a printing surface. By adding material from the top, a 3D-shape can be extruded. Additive manufacturing of metal uses a slightly different approach, labelled *Directed Energy Deposition* or DED [10]. This technique is very similar to ordinary welding. The object is heated by an electric arc, and new material is added in the form of a wire. Together with the material that is already present, this wire forms a pool of molten metal, which solidifies upon cooling down. 3D-printing of metal with this method is commonly referred to as Wire and Arc Additive Manufacturing or WAAM. A schematic representation of the process can be seen in Figure 1.2.

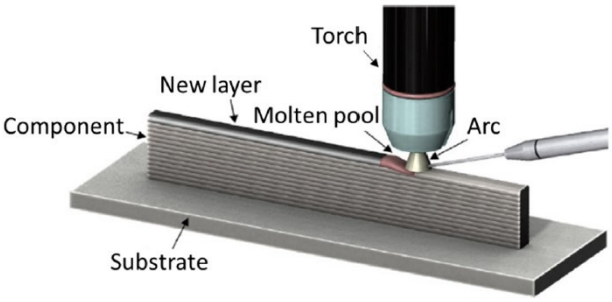


Figure 1.2: Schematic overview of Wire Arc Additive Manufacturing process, taken from Rodrigues [11]

1.2 Structural Optimisation

The quest for structural optimality has already occupied researchers for a long time. Although difficult to define explicitly, an optimal structure is generally seen as a structure with just the

right amount of material while retaining a sufficient level of stiffness and safety. Already in 1904, Michell [12] introduced his problem formulation of exact analytical solutions for truss structures. An example of such a truss can be seen in Figure 1.3. Michell's solutions generate structures with an infinite amount of bars, to transfer a load to a rigid foundation. His research proved that the optimal form of such trusses tends towards a grid in which all members are at a 90°angle to each other. Prager [14] continued with Michell's truss theory, and looked into possibilities to

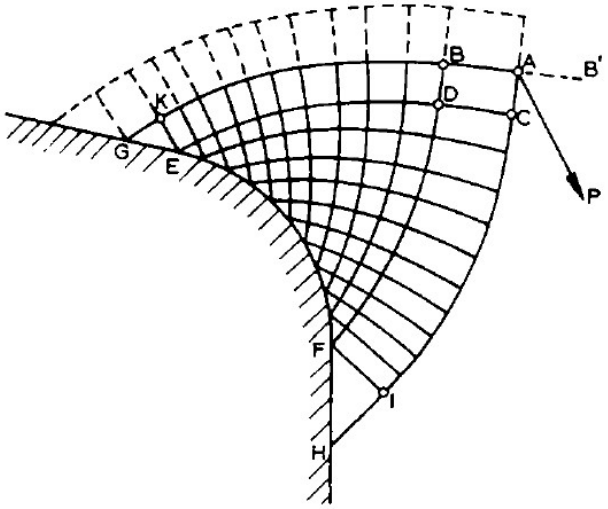


Figure 1.3: A *Michell Truss*, taken from Hegemier and Prager [13]

.

generate more practical structures. Opposed to the infinite formulation of Michell, he formulated methods to limit the amount of nodes that are generated in a design.

1.3 Problem Description

Extensive research is available on topology optimisation, and several different solution algorithms have been developed throughout the years. Multiple FEA-packages have implemented some type of optimisation functionality. In the automotive and aerospace engineering branches, this type of software is used on a regular basis. In design of buildings and other civil structures however, topology optimisation is rarely used. There are some examples where research is conducted on specific subjects, like the study by Fairclough and Gilbert to optimal forms of long-span bridges [15] and the glass swing of Snijder, van der Linden, Goulas et al. [7]. A field where topology optimisation is considered very promising, is automatic generation of strut-and-tie models for reinforced concrete. It has been proven in previous research that using topology optimisation for automatic strut-and-tie modelling shows great potential [3;4].

Recent years has also seen a large increase in additive manufacturing possibilities. 3D-printing is now available for an abundance of materials, including polymers, concrete and steel. Due to the large freedom of form, this manufacturing technique has gained a lot of support in the past decade. Combining this form freedom with the, usually complicated, geometry that results from topology optimisation methods is of significant interest. In general, the design of reinforcement steel is a time consuming process. Not only is the calculation extensive, but the further processing of the design poses various challenges. For instance, due to the often complicated layout of steel bars, the 3D modelling of reinforcement steel takes a lot of time. So even before the technical difficulties of the realisation on the construction site are considered, it is

clear that designing reinforcement steel for concrete is very tedious. In light of the developments in additive manufacturing and optimisation techniques, it would therefore be interesting to investigate the technical feasibility of optimised, additive manufactured, steel reinforcement.

Before, obtaining a valid strut and tie model from a topology optimisation result was seen as a big hurdle that prevented practical implementation. Recently however, Xia, Langelaar and Hendriks [16] were successful in deploying image recognition techniques for this purpose. Now, with additive manufacturing as intended production process, researchers are faced with additional challenges. Processing the specific manufacturing constraints after the optimisation is undesired. Therefore, research is required on how to account for the manufacturing process during the optimisation. It is expected that this order will produce a higher final material efficiency.

1.4 Research Goals

The broader goal of this research is *to contribute to a method for automatic generation and production of an optimised reinforcement layout for reinforced concrete, considering additive manufacturing constraints.*

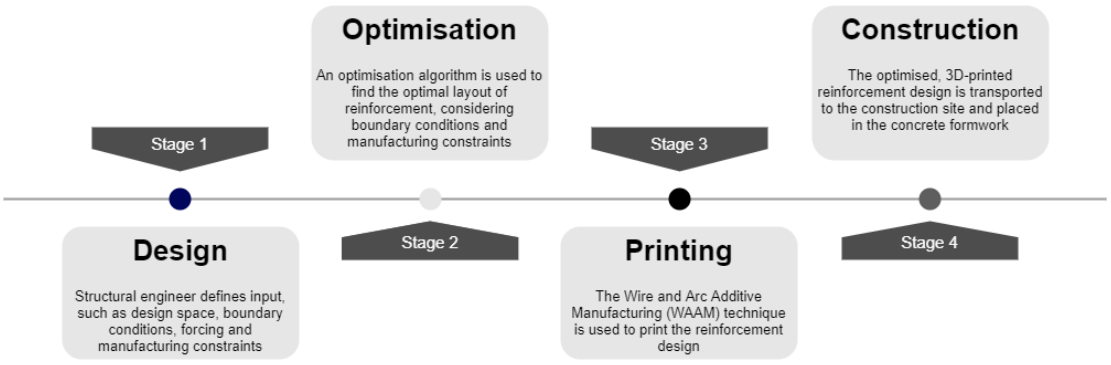


Figure 1.4: Flowchart of intended method

The idea for this intended method as follows. First, the structural engineer defines the input for the concrete element in computer software. Depending on the geometry and material input by the designer, the program determines an efficient layout of reinforcement steel. The output of this program is a digital model, consisting of the steel elements necessary to accommodate the tensile forces in the concrete. This model is then sent to a factory that is able to 3D-print it. The model can also be imported in BIM-software so it can be included in the final design. After the steel structure is printed in the factory, it can be transported to the construction site. It is placed at the correct position in the formwork before the concrete is poured.

Minimising material use plays an important role in the global efforts to limit the environmental impact of the construction sector. The ability to automatically produce highly efficient reinforcement layouts will contribute to these efforts. Next to this environmental argument, the stand-alone nature of the method is also expected to yield a reduction of design time needed. The calculation, modelling and production process will all be largely automated. This in turn contributes to the digitisation of the structural engineering sector as a whole, and allow more modern and performance-driven structures.

Now that the general goal of this research is established, it is filtered to a specific main research goal. This goal is *to extend a topology optimisation scheme in such a way, that it is able to*

generate a strut-and-tie model that is ready to be printed. The research questions that are proposed in the next section are aimed at providing a guideline to achieving this goal.

1.5 Research Questions

The following main research question is proposed.

To which extend can reliable strut-and-tie models be generated by a topology optimisation scheme, which includes additive manufacturing constraints?

Formulating an answer to this main question will be done in three parts. The first part consists of a literature review. This review is divided into two parts. The second part of this research proposes extensions and modifications to an existing topology optimisation scheme. The third part attempts to validate these modifications, by presenting a case study. Each part answers one or more sub-questions, which are explained below.

1.5.1 Part 1a: Additive Manufacturing

First, an in-depth review of additive manufacturing of metallic components is presented. To determine whether this production process fits the requirements of reinforcement steel, the available techniques are discussed. Details about the production process are highlighted, such as material properties of the printed material. Although 3D-printing offers a very large freedom in structural shape, there will still be limitations. The relevant limitations will need to be accounted for in the process of defining a suitable strut and tie model. It is also important to consider the format in which the digital model should be delivered to the printer. The following questions will be answered in this part:

- Which techniques are available to print steel, and which is most suitable for reinforcement?
- How are material parameters influenced by the different manufacturing processes?
- Which manufacturing constraints should be accounted for?
- How should the geometry be prepared?

1.5.2 Part 1b: Topology Optimisation

The second part of this research will focus on automatic generation of a strut and tie model for reinforced concrete. Ongoing and past research in this field will be discussed. Special attention will be given to topology optimisation methods. Several different optimisation techniques are available, which will be compared to each other in the context of additive manufacturing. It will be determined which optimisation algorithm is used in the rest of this research.

- Which optimisation algorithm is suitable for analysing reinforced concrete, and can easily be modified to include manufacturing constraints?

1.5.3 Part 2: Extension of Optimisation Method

The second part of this research focuses on improving the optimisation result for additive manufacturing. The goal is to obtain a printable model, that can be sent to a 3D-printer without any further steps. Although 3D-printing will allow complex shapes, there will still be some practical limitations. Implementing these constraints in the optimisation will most likely lead to a more efficient final structure. The constraints that follow from the first part of the research are therefore implemented in the optimisation algorithm that follows from the second part. The results that are produced by this extended method are then compared to results from literature.

· How can the selected optimisation algorithm be extended for additive manufacturing?

1.5.4 Part 3: Validation

The third and final part of this research consists of a case study. This case study is analysed twice. The first analysis will cover the traditional calculation procedure. This traditional method is explained, and the significant steps are shown on the basis of an example. The second analysis is done by a proposed method. This proposed method follows from the improved optimisation algorithm that is discussed in the previous chapter. This part of the research presents a validation of the proposed method. This method is compared to the traditional analysis as a whole, in the form of a discussion. Specific shortcomings are highlighted, and possible solutions these are presented. This part of the research also includes a show case of the newly developed method. In this section,

· How efficient are the results of the extended optimisation method compared to the traditional calculation methods?

1.6 Scope Limitations

Topology optimisation, strut and tie modelling and additive manufacturing processes are all very complex and extensive concepts. An abundance of research is available on each of these topics, and a lot of information can be extracted from various sources. To keep this research feasible as a graduation assignment, it is subjected to the following scope limitations.

- When design regulations are discussed in this research, the focus is on Dutch building regulations and the Eurocode.
- When this research mentions strut and tie modelling, the procedure from Eurocode NEN-EN 1992-1-1 section 6.5 is meant.
- This research will not go into detail about the mathematical proof of structural optimality, specifically the degree to which a certain algorithm is heuristic.
- The WAAM technique can be used in combination with multiple materials. A lot of research is available on the influence of material properties, but for the case study of this research, a strength of 435 N/mm^2 is assumed. Previous research proves that this strength is achievable.
- Material distortion and warping during printing are not discussed.
- Problems with multiple load cases are beyond the scope of this research.

Part 1

LITERATURE REVIEW

Chapter 2

Additive manufacturing in structural design

2.1 Introduction

This chapter discusses developments regarding additive manufacturing of steel. The following research questions are answered at the end of this chapter:

- Which techniques are available to print steel, and which is most suitable for reinforcement?
- How are material parameters influenced by this manufacturing process?
- Which manufacturing constraints should be accounted for?
- How should the geometry be prepared?

Several different aspects of this upcoming production process are explained, and it is discussed how it suits the requirements for reinforcement designs. Two different processes are commonly used for additive manufacturing of metallic components[17]. Powder Bed Fusion(PBF) and Wire and Arc Additive Manufacturing(WAAM). The first section of this chapter explains the PBF technique in terms of printing setup and functionality. It is also discussed how the material parameters are influenced by this production method. The second section is about WAAM, and is constructed in a similar manner. Finally, in the third section, knowledge that was obtained is put into perspective of the purpose of additive manufacturing of steel reinforcement. The research questions will be answered in the fourth and final section.

2.2 Powder Bed Fusion

The first process that is discussed is Powder Bed Fusion or PBF. The setup is shown in Figure 2.1. This technique is similar to most additive manufacturing processes, as it builds the specimen layer by layer. A powder feed is present, that uses a device to spread powder over a fabrication surface. This powder is melted by a heat source, usually a laser, at specific locations. This process is repeated, and an object forms inside the powder bed. A big advantage of this production process is that the specimen is always supported by the powder that surrounds it. In some instances this eliminates the need for a support structure in the design.

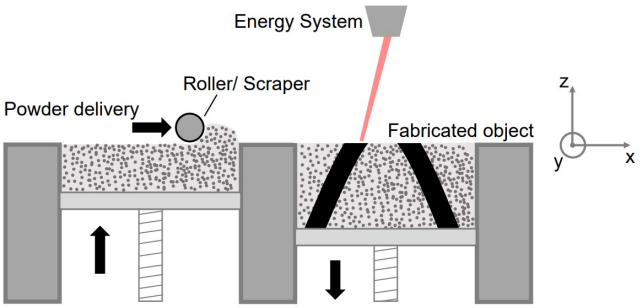


Figure 2.1: Powder bed fusion, taken from Wiberg [10]

The influence of this production process on the material parameters of the final structure is of course very important to consider. Because of the growing demand for additive manufacturing, researchers have shown increased interest in investigating the uncertainty in these material parameters. For instance, Hu and Mahadevan [18] have attempted to estimate the yield strength of printed material. This was done with advanced uncertainty quantification techniques, that also use experience from uncertainty data from traditional manufacturing processes. Opposed to this probabilistic approach, Kelly [19] performed various lab tests on a specific nickel alloy. It was found that the material produced with PBF shows increased strength, ductility and fatigue indicators when compared to traditionally wrought material. However, it was proven that the PBF process introduced some anisotropic effects.

Another two aspects of additive manufacturing processes that require attention are the build volume and build speed. Several manufacturers offer PBF machines, the larger versions reaching a build volume of around 500mm x 500mm x 500mm [20]. The build speed ranges between 5-80 cm^3/hr, but it has been suggested that using multiple lasers can significantly speed up the process [21].

In light of our goal to print reinforcement designs for concrete, it is expected that the PBF technique is not very suitable. The maximum build volume is too small for complete reinforcement designs, especially when elements for larger buildings are required. PBF is very suitable for specialised components with a complex geometry, but not for reinforcement steel.

2.3 Wire and Arc Additive Manufacturing

In this section the Wire and Arc Additive Manufacturing process(WAAM) technique is discussed. The basic printing setup for Wire Arc Additive Manufacturing is pictured in Figure 2.2.

2.3.1 Printing Process

This setup consists of a welding torch, mounted on a robotic arm. A separate wire feeder and power source are necessary to supply the welding arc and the welding material. The welding material is supplied from a coil. The positioning table is the basis on which the specimen is printed. In some applications, it is also possible to rotate this positioning table. This can be done to print under steeper angles, or to reach otherwise inaccessible places. The printing process is very similar to an industrial welding process. These welding processes are very common in, for instance, the automotive industry. The large scale industrial application of these welding robots makes them widely available, and control mechanisms already highly developed.

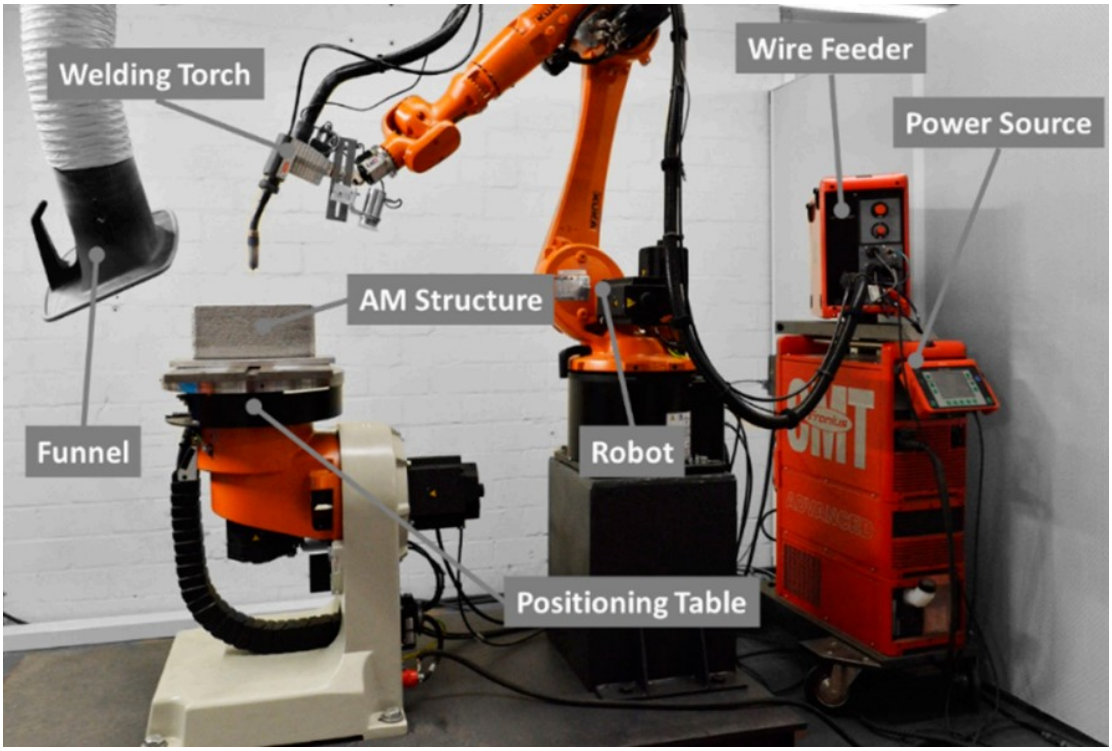


Figure 2.2: WAAM-setup, taken from *Köhler et al.*[22]

The printing process is very similar to ordinary arc welding. Arc welding is a technique widely used in the industry to join metal elements. This technique uses an electric arc to heat up the base material. The heat from the arc melts the metal, and creates a weld pool. In this weld pool, molten material from both elements mixes. To fill the joint, feed material is used. To protect the weld from being influenced by oxygen in the air, a shielding gas is sometimes necessary. This gas is different for each type of material and welding technique, but often contains a mixture of argon and carbon dioxide.

Three main arc welding techniques can be identified; Electrode, MIG/MAG and TIG. WAAM generally uses the Gas Metal Arc Welding(GMAW) technique[11]. This technique is more commonly known as Metal Inert Gas(MIG) or Metal Active Gas(MAG) welding. In this technique, the electric arc is generated between the substrate and a consumable wire. This wire is fed from the welding machine to the torch. The shielding gas is supplied to the torch through the same hose. This hose is flexible, which allows the torch to be moved easily. The welding torch is attached to a robotic arm. This robotic arm is the reason this manufacturing technique offers such high freedom in form, especially when combined with a movable printing surface. In Figure 2.3 it can be seen how a propeller for a large ship is fabricated using the WAAM technique.

Printing Strategies

Two printing strategies can be applied when using the WAAM technique; Continuous or dot-by-dot printing. As the name suggest, continuous printing is done when material is deposited without interruptions. This results in the forming of so-called welding beads. For larger objects, this technique is most popular because it has the highest deposition rate. Another printing strategy is dot-by-dot printing. This technique deposits drops of welding material with pauses in between. This leads to a different finish of the product, with a much more uneven surface pattern. A comparison of the two strategies can be see in figure 2.4.

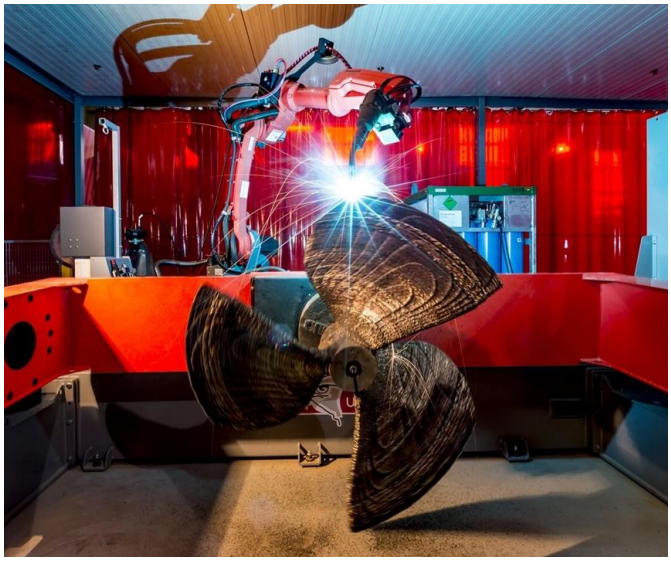


Figure 2.3: Fabrication of a ship's propeller using the WAAM technique

Figure 2.4: Surface finish for continuous(left) compared to dot-by-dot(right) strategy

2.3.2 Material Properties

Due to the significant difference from traditional manufacturing techniques, additive manufactured material shows some differences in material properties. These differences can be observed on both macro and micro scale. Wu, Pan, Ding et al. [23] published an in-depth review of the effect of these differences on engineering parameters such as yield strength and ductility. This review is based on lab tests for a wide variety of materials. For steel and aluminium, it was found that overall performance of the material was slightly lower when compared to traditional manufacturing techniques. Several solutions are discussed, which include post-process heat treatment and interpass cold rolling. Some of these measures proved to increase the performance of the material significantly.

Another issue that was observed in various studies, concerns the anisotropy of the material. It was found that significant difference of properties occur in the direction of printing [11]. It is expected that this is caused by the amount of heat cycles that are experienced by the material. The welding torch gradually builds up the structure from the base plate, which means the first deposited material experiences more heat cycles than the material at the top. This causes

differences in micro structure, leading to varying material properties.

2.3.3 Process Constraints

Although the WAAM technique offers far greater form freedom than any other traditional manufacturing technique, there are still limitations. Two of these limitations are worth mentioning here. The first concerns the minimum printing angle. If we assume that material is added from the top, it was found that a 60°angle from the horizontal[1] is still printable. Any angle smaller than this can become unstable during printing, especially with slender elements. This constraint is sketched in Figure 2.5.

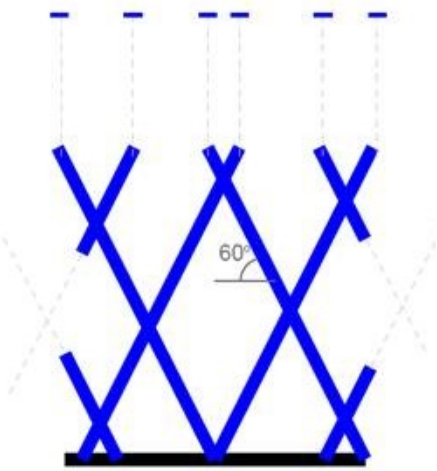


Figure 2.5: Angle constraint for WAAM technique

The other constraint that needs to be considered is a minimum member diameter. Due to the size of the welding wire and the way the torch is set up, only relatively large 'drops' of metal can be deposited at once. This means that only elements with a certain diameter can be produced. This diameter can be reduced by using the dot-by-dot method, but in practice this is rarely done. The dot-by-dot technique is more sensitive to printing errors. Combined with the much smaller deposition rate, often the continuous process is considered superior. Reinforcement designs are large structures. A large deposition rate is therefore expected to be essential for the feasibility of this method. Therefore, the minimum member diameter for the continuous process is used in the remainder of this research. This minimum diameter is found to be 1cm[1].

2.3.4 Printing Software

Preparing a design for a 3D-printer is often called *slicing*. As the name suggests, this process slices the model and generates a path for the print nozzle. This process is similar for all additive manufacturing processes based on the *Directed Energy Deposition* method. Stand-alone software is available for this purpose, for instance Ultimaker Cura. However, in most industrial applications this software is not often used. Most companies that offer WAAM technique have either developed their own software, or modified existing software to their specific needs. An example of this is the software by MX3D, designed specifically for the WAAM process.

[0]This constrained is advised by Vincent Wegener, founder of Rotterdam Additive Manufacturing Lab(RAMLAB)

This path finding for additive manufacturing is still under development, and has created a whole field of research on its own. For our purpose, it is sufficient to know that most printing software offers functionality to import stereolithography files(.stl). This type is widely available for 3D-models, which means large 3D-software like Rhinoceros can export to this format.

2.3.5 Viability for reinforcement printing

Print Size

Opposed to the *Powder Bed Fusion* technique shown in the previous section, WAAM is much more suitable for printing larger structures. All material is deposited in-situ, so no powder beds or other containers are required. In theory, this means that there is no direct limit to the size of the structure other than the reach of the robotic arm. If the robot itself is able to change position, like in the case of the bridge built by MX3D(Figure 2.6), the size limit disappears altogether.

Figure 2.6: Additive manufactured bridge by MX3D

Print Speed

Printing speed or deposition rate of WAAM processes are generally higher than other additive manufacturing techniques. The deposition rate depends heavily on the geometry and material of the object that is printed. The settings of the welding torch, for example voltage and welding current, determine the rate at which material can be added. To avoid overheating of the substrate, it is common practice to alternate printing location. When all of this is considered, deposition rates of WAAM vary between 4-9 kg/h [17].

Surface properties

The surface finish of printed material is generally seen as a challenge of the WAAM technique. For most applications, a smooth surface is required. This means that usually post-processing is necessary to obtain the desired result. However, for reinforcement bars, the rough surface is an advantage. Bonding strength between concrete and steel is very important to activate the hybrid system. In traditional rebar, the surface is purposely profiled for this specific reason.

Additive manufactured material automatically provides this. It was proven by Mechtcherine, Grafe, Nerella et al.[24] that as-built printed material exhibits similar bonding behaviour to traditional steel. If higher bonding is required, additional profiling can be included in the design without much effort(Figure 2.7).

Figure 2.7: Normal(left), improved (middle) and extra profiled (right) additive manufactured steel bars for concrete reinforcement. Taken from Metchcherine, Grafe, Nerella et al.[24].

Costs

In general, WAAM is more expensive than traditional manufacturing[17]. Again this aspect of the process is difficult to quantify. Traditional manufacturing techniques have become very developed over time, and are therefore optimised to a high level. Additive manufacturing techniques are not yet firmly established production methods, which ultimately leads to higher production costs. These costs may reduce over time, but it is expected that additive manufacturing will never replace traditional processes in a structural engineering context. It should be seen as an addition to the possibilities, filling a gap for a very specific type of structures. This manufacturing technique is expected to be feasible only for objects that present big challenges during design and execution phase. In these instances, the higher production costs may be justified by the time savings for engineers and construction workers.

2.4 Answering the research questions

This section presents answers to the four research questions that are stated at the beginning of this chapter. The first research questions is:

Which techniques are available to print steel, and which is most suitable for reinforcement?

Two main types of additive manufacturing are suitable for metallic components. *Powder Bed Fusion* relies on a laser that melts a metallic powder at specific locations. *Wire and Arc Additive Manufacturing* uses conventional arc welding combined with a robotic arm. In light of additive manufacturing of reinforcement steel for concrete, only WAAM is relevant. This is the only technique that is able to print large enough structures.

How are material parameters influenced by this manufacturing process?

Several studies have been done to establish the strength and ductility of additive manufactured steel. Wu, Pan, Ding et al.[23] have published extensive information on this subject. From their work it was concluded that a slight reduction in material performance can be observed in 3d-printing of steel and aluminium.

More elaborate research on a specific material was done by Metcherine[24]. Here, a S235JR base plate was used combined with a solid welding wire complying with ISO 14341 - A - G 46 5 M G3Si1. The following conclusions were found when compared to conventional steel reinforcement:

- 28% lower yield stress
- 16% lower tensile strength
- 250% higher strain capacity

This indicates that a reinforcement design that is prepared for the WAAM technique will require other material parameters than if it was made from conventional reinforcement steel. It is therefore of importance to ensure that these parameters can be adapted accordingly.

Which manufacturing constraints should be accounted for?

Two main constraints can be recognised. A minimum printing angle of 60°from the horizontal is required to ensure enough stability during printing. Also, a minimum bar diameter of 1cm should be ensured. Smaller diameters can not be printed.

Another important challenge of the WAAM technique is limiting the distortion of printed material. The printing strategy should be chosen such that no excessive amount of heat is added to the already printed structure. This is especially harmful when printing at the same location for an extended period of time. This is a big challenge currently faced in this field, but it is beyond the scope of this research.

How should the geometry be prepared?

Any 3D-model can be printed, as long as it can be exported to a stereolithography file(.stl). Most software packages, including Rhinoceros and Grasshopper, are able to export to this file format. It is worth mentioning here that the .stl file can only be sliced if it is a closed solid. This can be described by water tightness. If the shape would be filled with water and it stays in, the model is valid. If any holes or openings are present in the design, the slicing software will generate an error. This is also beyond the scope of this research.

Chapter 3

Structural Optimisation

This chapter focuses on structural optimisation. Several different types of optimisation strategies exist in literature. To establish an insight in current developments in this field of research, three algorithms are explained here. An introduction to the mathematical basis is given, as well as some examples that clarify their application. The research question that is answered in the final section of this chapter is:

- *Which optimisation algorithm is best suitable for analysing reinforced concrete?*

This chapter consists of four parts. The first section gives a brief historic overview of topology optimisation. The research efforts of the past century resulted in three main branches of topology optimisation. In the next three sections, these three main branches are discussed. For each method, the theory is explained first. An example is then presented that showcases the functionality of the method. The suitability for additive manufacturing process is an important factor in this research. Therefore, a section is presented that compares the optimisation algorithms in this context. The applicability of each method to automated strut and tie modelling is discussed, as well as possible challenges and shortcomings. In the final section, an answer is formulated to the research question.

3.1 Historical Perspective

The nature of the structural design process has always been, and to some degree still is to this date, size oriented. Layout of structural elements is chosen from practical considerations such as interior space or architectural forms. When this layout is established, each individual member is analysed and validated for strength and deflections. Structural optimisation attempts to use a different approach. Here, the layout of structural elements is not known beforehand. The design space in which material may be placed is defined, and an algorithm searches for the optimal configuration of this material. According to Sigmund and Maute [6], topology optimisation aims to answer the question: *How to place material within a prescribed design domain in order to obtain the best structural performance?*.

Generally seen by fellow researchers as the founder of modern optimisation techniques, Michell [12] published his work on layout optimisation of trusses in 1904. He formulated analytical methods to find the optimal layout of bars, given a set of boundary conditions. His formulation was not bounded, which meant an infinite amount of bars could be generated to transfer the loads to a rigid foundation. This was deemed impractical, so several researchers attempted to improve his

methods. An example of this is the work done by Prager [14], who tried to reduce the complexity by limiting the amount of nodes that were generated.

Probably due to the relatively tedious calculations that were involved, the layout optimisation techniques of Michell were not widely used up until the 1980's. It was around this time that computers became available, that could perform large amounts of calculations in succession. The tedious calculations could now be automated, which opened doors to new possibilities. Bendsøe and Kikuchi [25] where among the first researchers to start exploring the concept of computational optimisation. Their proposed method assumes a material model with tiny voids. An optimisation loop searched for the optimal configuration of these voids within the design space. In Figure 3.1, some intermediate results of their work can be seen.

Figure 3.1: Optimisation of a rectangular modelling space. Black areas indicate material, white areas are voids. Taken from Bendsøe and Kikuchi [25]

A year later, Bendsoe [26] presented his research that included a penalisation factor for intermediate-density elements. This forced the result to be more black-and-white, meaning a more pronounced division between material and voids. Xie and Steven [27] continued with this concept, extending it to a so-called evolutionary procedure. A finite element analysis is performed on a given shape, and the elements with the lowest stress are discarded. It was proven that this method produces results similar to the analytical solutions presented by Michell [12].

3.2 ESO

ESO, or Evolutionary Structural Optimisation, was proposed by Xie and Steven in 1993 [28]. The method is inspired by natural processes. Bone erosion around metal implants is given as an example. It is found, that when metal implants are used to reinforce fractured bone, local bone erosion occurs in areas that are minimally loaded. This example is pictured in Figure 3.2. If only the material that is loaded remains, this process can be seen as very structurally

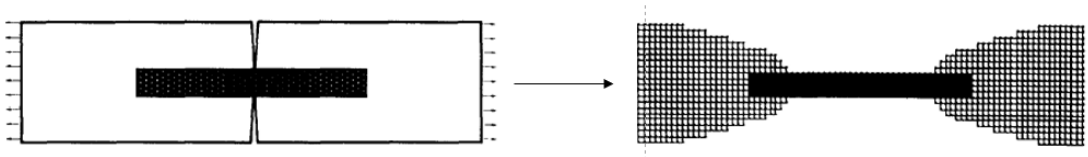


Figure 3.2: Bone erosion around metal implant. Taken from Xie and Steven [28]

efficient. Xie and Steven therefore developed an algorithm that uses this idea to find efficient load bearing structures. In this algorithm, the stress distribution is calculated in a Finite Element Analysis. A rejection criterion is used to eliminate elements. This rejection criterion is defined as a rejection ratio (RR) times the maximum von Mises stress. A loop is initiated, repeatedly eliminating elements and re-evaluating the stress distribution. A steady state is found when no more elements satisfy the rejection criterion. When this steady state is reached, an evolution rate (ER) is added to the rejection ratio and the elimination loop is re-initiated. This process is

repeated until the predetermined optimality condition is met. This condition is usually expressed as a performance index. The performance index is defined as the non-dimensional number:

$$PI = \frac{\sum \sigma_{vM} V_e}{FL} \tag{3.1}$$

In which σ_{vM} is the von Mises stress, V_e is the element volume, F is a representational force and L is a reference length. This performance index compares the performance of the structure against a fully stressed design in which all material is loaded to the yield stress.

3.2.1 BESO

With the method that is described above, it is very difficult to mathematically prove that the result is not just a local optimum. When elements are discarded, there is no functionality to reuse this material at a later stage. To overcome this problem, the Bi-directional Evolutionary Structural Optimisation (BESO) algorithm is proposed by Xie, Steven and Querin in 1998 [29]. The biggest difference with the previous method is the functionality to add elements. This adding is done, similar to the elimination scheme, by comparing the stress in an element to the maximum von Mises stress that occurs. If this occurring stress is larger than a certain inclusion ration (IR) times the maximum von Mises stress, elements are added to all free edges of this element. This allows addition of material in highly-stressed areas. The process of this algorithm is visualised in the flowchart in Figure 3.3.

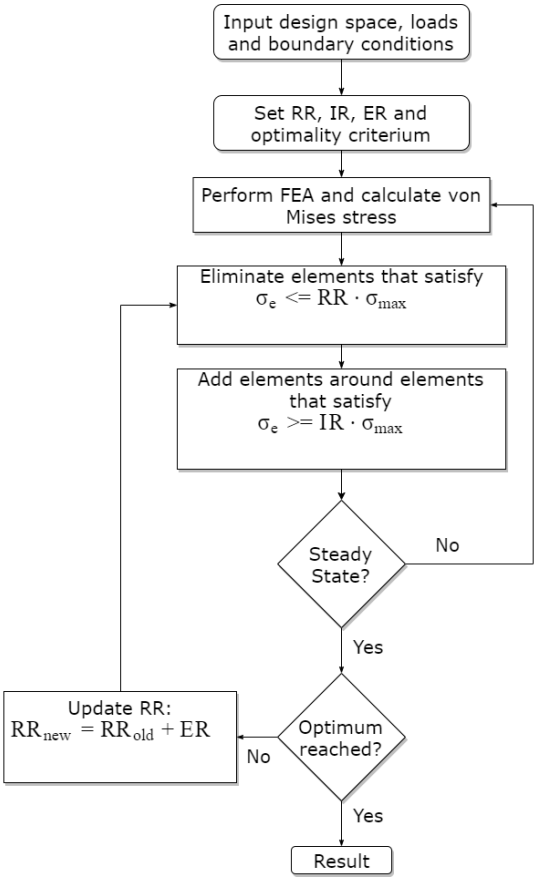


Figure 3.3: Flowchart of BESO-algorithm

3.2.2 Mathematical Representation

Including the bi-directional ability, the BESO method takes the mathematical form[1]:

$$\begin{aligned} \min_{\mathbf{x}} : f(x) = \mathrm{PI} &= \frac{\sum_{e=1}^{N} \sigma_{\mathrm{VM}_e} \mathrm{V}_e}{\mathrm{FL}} \\ \text{subject to} : \sum_{e=1}^{N} [[\mathrm{K}]^e \{u\}^e - F^e] - \{P\} &= 0 \\ : x_e \left(x_e \cdot \sigma_{\mathrm{VM}_e} - RR \cdot \sigma_{\mathrm{VM}_{Max}}\right) &\geq 0 \\ : \eta_e \left(IR \cdot \sigma_{Max} - \eta_e \cdot \sigma_e\right) &\geq 0 \\ : x_e \in X_e = \{0,1\}, e \in E & \\ : \eta_e \in H_e = \{1,t\} & \end{aligned} \tag{3.2}$$

where

- σ_{VM_e} is the element Von Mises stress;
- V_e is the element volume;
- F is a representational force;
- L is a reference length;
- RR is a rejection ratio, range $0 \leq RR \leq 1$;
- IR is the inclusion ration, range $0 \leq IR \leq 1$;
- E is the set of discrete elements within maximum possible design domain;
- e is element e of the set E;
- X_e is a set of allowable discrete values;
- N is the number of elements in the structure;
- η_e is the addition multiplier for element e. Range $0 \leq \eta_e \leq 1$;
- $t = 1 - \frac{q_a}{q_p + q_a}$;
- q_p is the sum of discrete values of elements present around and including element **e** before checking inequality equation;
- q_a is the sum of discrete values of elements added around element **e** if the inequality constraint was violated;

3.2.3 Example

To demonstrate the functionality of the BESO algorithm, an example is presented here. We will consider the fixed cantilever in Figure 3.4. The width is chosen to be double the height, with the complete left edge fixed. A force of magnitude F is applied to the middle of the right edge. When this design space is optimised with the BESO method, an initial rejection rate(RR) is required as input. The result presented below is taken from Xie and Steven [28]. The three steps, or evolutions, are the steady state solutions with a rejection rate of 11%, 15% and 18% respectively.

[1] Taken from Xie, Steven and Querin [29]

Figure 3.4: Cantilever example

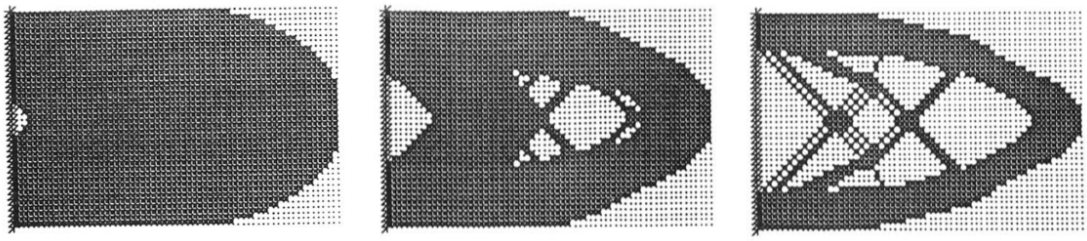


Figure 3.5: BESO result of fixed cantilever, taken from Xie and Steven [28]

3.3 SIMP

The next solution strategy that will be discussed here is Solid Isotropic Material with Penalisation. According to Rozvany [5], SIMP is the most popular topological optimisation algorithm. This algorithm is also based on a finite element analysis. The main component of the objective function of a SIMP algorithm is the compliance of a structure. Compliance is defined as force times displacement, which is also known as the strain energy. In finite element notation, this can be represented as:

$$c = \mathbf{U}^T \mathbf{K} \mathbf{U} \tag{3.3}$$

where $\mathbf{U}$ and $\mathbf{K}$ are the global displacement and stiffness matrix respectively. Similar to the BESO-method, the SIMP-scheme redistributes material over the design domain in each iteration. However, this is not done by discarding elements, but rather by adjusting the density of each element. This makes it a continuous scheme, in which elements can have an intermediate density. For simplicity reasons, we will limit our scope to plane stress problems. For this case, it can be proven that the stiffness of an element is directly proportional to its density. The topology optimisation problem, with the aim to minimise compliance, therefore takes the form:

$$\begin{aligned} \underset{\mathbf{x}}{min} : c(x) = \mathbf{U}^T \mathbf{K} \mathbf{U} &= \sum_{e=1}^{N} (x_e)^p u_e^T k_0 u_e \\ \text{subject to} : \frac{V(x)}{V_0} &= f \\ : \mathbf{KU} &= \mathbf{F} \\ : 0 < x_{min} &\leq x \leq 1 \end{aligned} \tag{3.4}$$

where x_e is the scaling factor that is applied to the density of each element. p is the penalisation factor, that is introduced to suppress elements with an intermediate densities. For manufacturing considerations it is desired to have a discrete domain, in which elements are either present or not present. The factor p penalises the stiffness of intermediate densities and contributes to a

discrete domain. The minimal compliance problem is subjected to three constraints. The first constraint is on the volume. At each iteration, the volume ratio has to be equal to a predefined volume fraction. The second constraint is the well known constitutional relation that relates force to displacement. The third and last constraint is on the design variables x_i. These variables are allowed to take values between zero and one, but not equal to zero. Zeros in the stiffness matrix cause singularities in the solution and are undesired from a numerical point of view.

3.3.1 Penalisation factor

Already in the 1970's, Rossow and Taylor [30] proposed a method that implemented a density-adjusting technique for finding optimal structural forms. However, the result of their application returned a lot of so-called 'grey elements'. Grey elements are finite elements with intermediate densities. For practical reasons, grey elements are undesirable. Imagine a plate with thickness t and density ρ that is modelled in two dimensions. This plate is optimised and a certain element has a final density of 0.5ρ. If this plate was made from a single material, the only way to manufacture this element would be to have a thickness of $0.5t$ in this location. This would lead to an expensive machining process which makes it unpractical for implementation. A more 'black and white' solution is required. For this reason, Bendsoe [26] proposed a penalisation factor. As mentioned, the density of an element is directly proportional to the stiffness for plane stress problems. Rozvany [5] explains this by relating the normalised plate thickness ρ(actual thickness over maximum thickness) to the normalised stiffness s. The power law then takes the form:

$$\rho = s^{\frac{1}{p}} \tag{3.5}$$

In words; the stiffness of elements with intermediate densities is penalised. A commonly used penalisation factor p is 3. The graph in Figure 3.3.1 presents this relation compared to the situation where no penalisation factor is used.

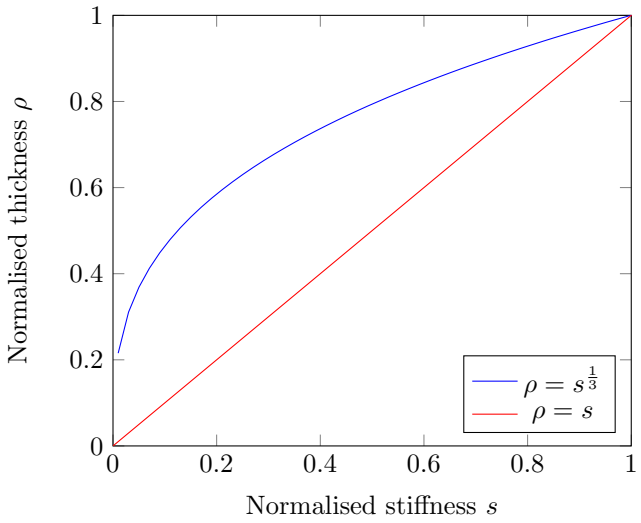


Figure 3.6: Relation between density and stiffness

3.3.2 Numerical Implementation

The basis of the SIMP method has now been established. Several different numerical implementation methods are available. In-depth explanation of these methods is beyond the scope of

this research, so instead an example will be presented. The example that will be shown here is generated with *A 99 line topology optimization code written in Matlab* by Ole Sigmund [31]. This code is based on a *Optimality Criteria* method. The flowchart that describes the process in this script can be found in Figure 3.7. The

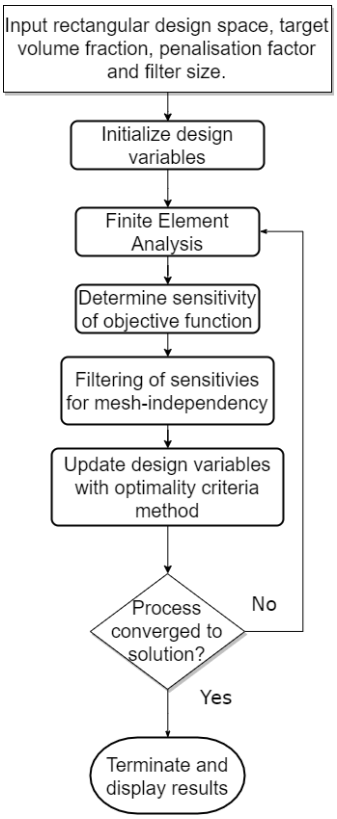


Figure 3.7: Flowchart of SIMP-algorithm

3.3.3 Example

For illustration purposes, we will consider the same structure as in the example from the previous section. This cantilever structure can be found in Figure 3.4. To demonstrate the workings of the SIMP algorithm, we will make use of *A 99 line topology optimization code written in Matlab*, by Ole Sigmund [31]. This code implements the SIMP method. We choose a volume fraction of 0.3, a penalty factor of 3.0 and a relative filter size of 1.2.

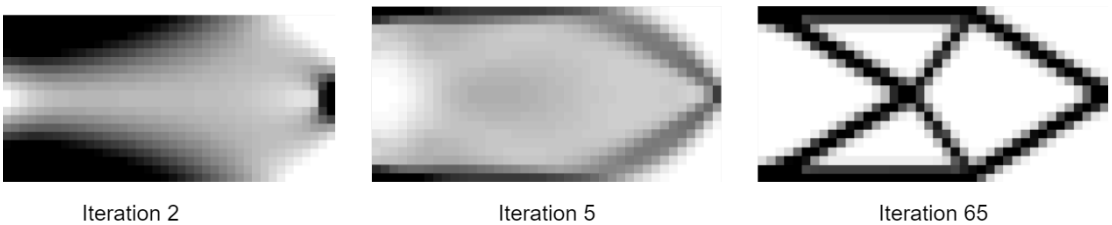


Figure 3.8: SIMP result of fixed cantilever example

As can be seen in the leftmost picture of Figure 3.8, a lot of material has already been adjusted in

the second iteration. In the fifth iteration, the final outline of the structure can be distinguished. The third image shows the structure after 65 iterations. The influence of the penalty factor is clearly visible. Almost all elements are either completely contributing, or not present in the solution.

3.4 Ground Structure Method

The third and final type of structural topology optimisation algorithm that will be discussed in this chapter is the ground structure method. Opposed to the (B)ESO and SIMP method described previously, this third solution strategy has a discrete nature. In both (B)ESO and SIMP algorithms, the design space is modelled and analysed as a continuum. The ground structure method uses an initial truss layout for internal force analysis. When a design space is given as input, the first step is to generate node locations within this plane. As an initial truss layout, or *Ground Structure*, usually all possible connections are made. An example of such a ground structure is given in Figure 3.9. This image is taken from He, Linwei and Gilbert [32]. These

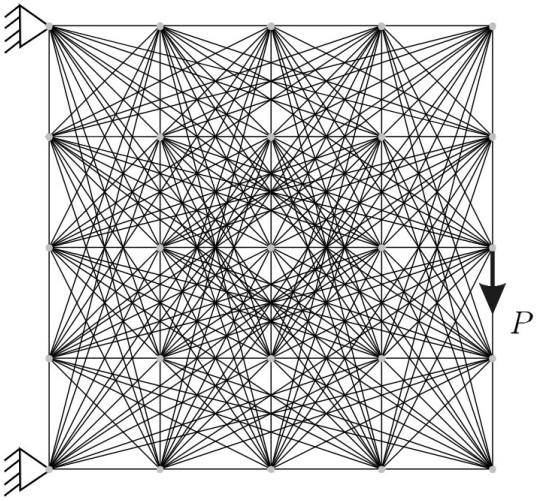


Figure 3.9: Ground Structure, taken from He, Linwei and Gilbert [32]

researchers developed a python script that implements the ground structure method. This python script, together with its mathematical basis, will be used here to explain the method. In this case, the problem is defined as a minimum volume problem. The problem is described by:

$$\begin{gathered} \min V = \mathbf{I}^T \mathbf{a} \\ \text{Subject To:} \\ \mathbf{Bq{=}f} \\ \mathbf{q} \geq -\sigma^- \mathbf{a} \\ \mathbf{q} \leq \sigma^+ \mathbf{a} \\ \mathbf{a} \geq \mathbf{0} \end{gathered} \tag{3.6}$$

Where V denotes the volume of the complete structure. $\mathbf{I}$ is a vector containing all the member lengths, $\mathbf{a}$ is a vector with the corresponding areas. $\mathbf{B}$ is the stiffness matrix and $\mathbf{q}$ are the internal forces. $\mathbf{f}$ are the external forces. This python implementation allows users to specify different limiting tensile and compression stresses, which are denoted by σ^+ and σ^- respectively.

3.4.1 Example

Again we consider the fixed cantilever from Figure 3.4. We use the python implementation by [32] to illustrate the ground structure method. The limiting tensile stress is set equal to the limiting compression stress to unit value 1. The joint cost is taken as 1.0. Figure 3.10 shows two intermediate plots and the final result.

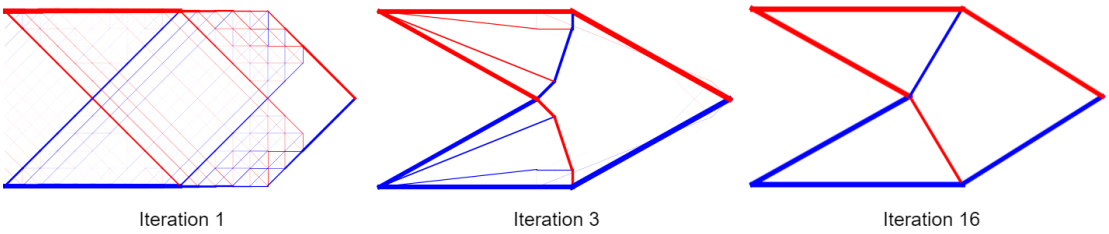


Figure 3.10: GSO result of fixed cantilever example

The first image is a representation of the structure after the first iteration. A lot of small truss elements are visible. It can already be observed that the top and bottom left corners will be the important supports. After the third iteration, pictured in the middle, most small elements have disappeared. The final shape has become more pronounced, and the elements that have a large contribution to the load bearing have increased in size. In the final iteration, pictured on the right, the connection in the middle has changed shape. The meeting point of these four connections has shifted slightly to the left.

3.5 Suitability for additive manufacturing of reinforcement

Now that three optimisation algorithms are explained, the question raises which of these is most suitable for implementing in an additive manufacturing process for reinforcement steel. Different aspects of the result can be identified that are important to consider in this context.

3.5.1 Complexity

First, we compare the complexity of the result. For traditional manufacturing processes like casting or milling, it is unwanted to have a high complexity. Fabricating voids and oddly shaped geometry will increase production times and therefore costs. It is for this reason that most implementations of topology optimisation schemes offer some kind of method to force simplicity of the solution. In SIMP for instance, the penalisation factor is introduced to force a more 'black and white' solution. This is good for traditional manufacturing, because then it is difficult to fabricate elements with intermediate densities. In 3D-printing, this may not be a problem. Let's consider the example of the plate with varying thickness that is described in the SIMP-section above. To cast this element, complex formwork or a mould would be required. If the same element would be manufactured by 3D-printing, it would be possible to print the varying thickness. The python implementation of the ground structure method by Xie and Steven [32] also includes a function that decreases the complexity. This is done by introducing a joint cost, which suppresses designs that generate more connections and thus more members. These measures are effective in light of traditional manufacturing, but all have a negative effect on the overall performance of the structure. This is proven in Appendix A. It is therefore questionable whether these complexity-suppressing measures should be activated when designing for additive manufacturing. The effectiveness of these measures is therefore not of large importance when choosing a suitable algorithm, and should not be decisive for the choice.

3.5.2 Stress Insight

Another aspect to consider is stress in the individual members. A steel reinforcement design is required, but the optimised topology includes the areas that are under compressive stress. Here

lies a problem for the (B)ESO and SIMP methods, since it is not immediately clear what the stress distribution within the structure is. To identify which elements need to be included in the reinforcement design, it is important to know which areas are under tensile and compressive stress. The printed reinforcement should only include the members that are under tensile stress, because the compression will be taken by the concrete. To obtain this stress distribution from the finite element analysis, an extra step is necessary. The displacement field needs to be assimilated in a process of back-substitution, where the internal stress is calculated with the element stiffness matrices. In the ground structure method, this information is already available. The member forces are calculated and in each iteration to check of the maximum stress is not violated. It is therefore expected that it is easier to extract details about the stress distribution from the result of a GSO optimisation.

3.5.3 Design Readiness

The third item that will be discussed here is design readiness. 3D-printing relies heavily on the software that controls the printers. This software is developed to automatically generate print paths and print nozzle movements. It is important to consider the input these programs require, because this is the next step in the realisation process. The result of the topology optimisation eventually needs to be the input for such a program. As we have seen in the previous chapter, this input is usually a 3D-model. Generating such a model is generally done in modelling software like Rhinoceros. It is therefore important to discuss the possibility to import an optimisation result in such modelling software. Both SIMP and BESO methods rely on a continuum formulation, where the design domain is modelled with plain stress elements. After removing or adjusting the elements within this domain, the solution consists of some sort of matrix representation of the design domain. In this matrix, each entry represents an element. The elements will have either density 0 or 1 in BESO, or any value between 0 and 1 in SIMP. To obtain a 3D-model from this result, some sort of post processing is required. The result of the ground structure method is different in this way. The result consists of a list of members or bars with their corresponding cross sectional area. Importing this result in Rhinoceros can be a relatively straightforward process if this list is exported as, for instance, a CSV-file.

3.5.4 Software Availability

For the purpose of this research, it is important to find topology optimisation software. Commercial options are available, such as Tosca Structure and Autodesk Nastran. However, these will not allow changes to the code. Options with open-source code should therefore be considered.

For the BESO algorithm, Zuo and Xie [33] developed a Python script that is able to perform topology optimisation on 3D structures. The script adopts the Abaqus Scripting Interface. The finite element calculation is preformed in Abaqus, and python is used to determine which elements need to be discarded or added.

An open-source implementation of the SIMP algorithm was made by Sigmund [31]. This Matlab script offers a very elegant and compact method to perform topology optimisation. This script is widely used in other research, wich may be attributed to its simplicity. This simplicity results in very high analysis speeds, as proven in Appendix A.

The ground structure method, or layout optimisation technique, is somewhat less pronounced in recent literature. The exception to this is the Python script by He, Gilbert and Song [32]. This script provides a stand-alone optimisation algorithm for layout of trusses. It is based on the member adding technique, as explained in Appendix B.

3.5.5 Code Flexibility

Since all implementation that were mentioned above are made in either Python or Matlab, it is expected that all three can be adjusted to the purpose of this research. The object oriented nature of Python is a big advantage in this light, since external libraries can easily be imported.

The use of the Abaqus Scripting Interface in case of the BESO method can be seen as both an advantage or disadvantage. Abaqus is a robust FEA package, which will yield very precise and fast results. However, performing the calculation in different software may cause issues regarding availability of certain parameters. Therefore, stand alone software is preferred.

3.5.6 Previous Research on Strut and Tie Modelling

It is also interesting to look at previous research in this field. Several examples exist that implement topology optimisation schemes for strut and tie modelling. For instance, the same researchers that worked on developing the (B)ESO method published a paper on using the algorithm for this purpose [3]. The SIMP method was also put forward in other research, and it was found that a strut and tie model could easily be developed by making use of this algorithm [4]. One issue remained, which was discretising the continuum result to individual struts and ties. In both before-mentioned papers, this was done manually. Very recently, Xia, Langelaar and Hendriks [16] published a breakthrough paper, incorporating image recognition techniques to automate this process. Visualised in Figure 3.11, software was presented that is able to identify the individual members from the continuum result.

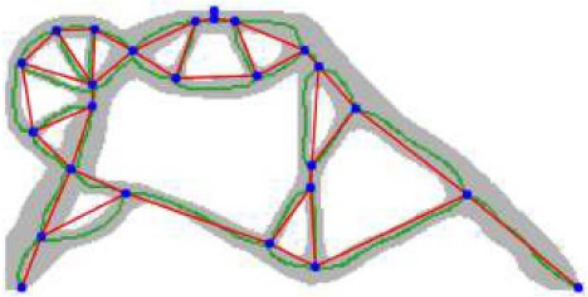


Figure 3.11: Automatic discretisation of topology optimisation result for strut and tie model, by Xia, Langelaar and Hendriks [16]

The ground structure method was also used to generate strut and tie models by Bolbotowski, Knauff and Sokol [34;35]. The use of GSO meant that post-processing was not necessary to obtain a valid strut and tie model. It was proven in their papers that this method is very versatile, and can be used in a multitude of plane stress problems to automatically generate efficient strut and tie models. Some examples of their work can be seen in Figure 3.12.

3.6 Comparison of methods

To give insight in the differences between the algorithms, Table 3.1 is presented. The entries in this table are based on the considerations from the previous section, combined with the results of the parameter study that can be found in Appendix A. Each comparison aspect is assigned a weight factor η. This weight factor is determined by establishing the importance of that specific

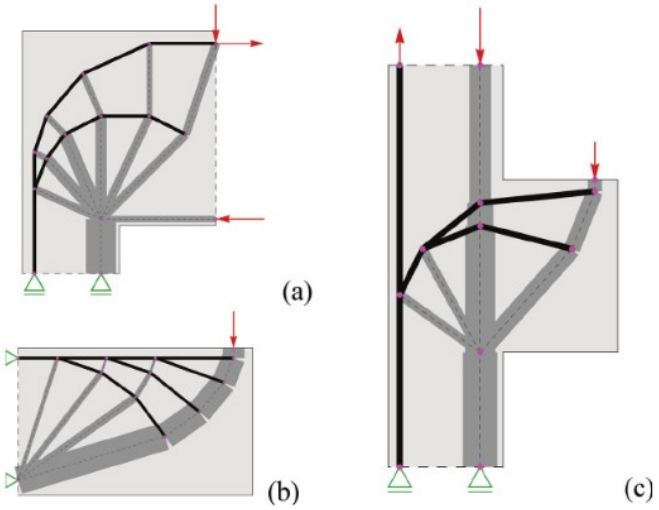


Figure 3.12: Strut and tie models for a corner (a), cantilever (b) and corbel (c) obtained with the Ground Structure Method, taken from Bobotowski and Sokol [35].

aspect in the context of this research. The scores are presented in - -/-/+-/+/++ format, and are assigned score values 1/2/3/4/5 respectively. This allows for calculation of a final score.

	η	BESO	SIMP	GSO
Speed	1	+	++	-
Complexity	0	+	+	++
Stress Insight	2	-	-	++
Design Readiness	2	-	-	+
Software Availability	2	+	++	++
Code Flexibility	2	++	+	++
Innovation Potential	1	-	-	+
Score		32	33	44

Table 3.1: Comparison of TO algorithms

3.7 Answering the research question

The research question that is answered here is:

- *Which optimisation algorithm is suitable for analysing reinforced concrete, and can easily be modified to include manufacturing constraints?*

All variants of topology optimisation that were discussed in this chapter, have shown promising results when applied to strut and tie modelling for reinforced concrete. However, *Ground Structure Optimisation* proved to be the best choice considering the requirements that are set for this research. As can be seen in table 3.1, a multi criteria analysis was performed to find the best alternative. GSO obtained the highest score, which justifies the choice to use this algorithm in the remainder of this research. This choice is strengthened by the availability of a very user-friendly and robust open-source Python script by He, Gilbert and Son [32] that incorporates this algorithm.

In light of previous research on this subject, the choice for GSO as a starting point is expected to be a good addition to this research field as a whole. A lot of examples exist from the past decades where the continuum formulation of topology optimisation methods are used to find efficient strut and tie models for concrete. The use of the ground structure method tends to be less pronounced in literature, with the exception of the work done by Bobotowski, Knauff and Sokol [34;35]. Putting this method in perspective of additive manufacturing is therefore expected to be of significant interest.

Part 2

ADAPTING THE OPTIMISATION TO THE MANUFACTURING PROCESS

Chapter 4

Printing Constraints

The previous chapter has explained the choice for an optimisation algorithm. The goal of this research is to extend this optimisation method to suit the additive manufacturing process. It was shown in Chapter 2 that there are two relevant geometrical constraints related to this production process.

- Minimum angle of 60° from horizontal
- Minimum member diameter of 1.0 cm

It is important to consider these constrains throughout the process of optimising a structure. Where possible, printing constraints should be accounted for during the optimisation. Adding or removing members from the result is expected to be counterproductive and will likely have a negative effect on the material efficiency. This chapter therefore aims to answer the research question:

How can the printing constraints be included in the optimisation algorithm?

For both constraints it is investigated whether they can be included in the optimisation. As a starting point, the python implementation of the Ground Structure Method by He, Gilbert and Song [32] is used. This python script is freely available for the purpose of further development. The methods described in this section form additions or adjustments to this script. In pursue of transparency, these changes are described in detail. The code is made available either in the text or in the appendices.

The first section of this chapter discusses the angular constraint. A solution is proposed to avoid generation of members at a specified angle during the optimisation. The following section proposes a second solution for dealing with the angle constraint, by adjusting the orientation of the complete structure. The third section of this chapter discusses the minimum member diameter, and proposes a method for adjusting the diameter. In the fourth and final section, the knowledge gained in this chapter is collected and an answer to the research question is formulated.

4.1 Angle Constraints

The first manufacturing constraint that is discussed here is the minimum printing angle. As pointed out in Chapter 2, members can only be printed that have a minimum inclination of 60° from the horizontal plane. A smaller angle will result in instability during the printing process, or significant reduction of strength parameters[1]. The first solution that is proposed to this issue, is reducing the set of possible members.

4.1.1 Generating a reduced ground structure

The original script by He, Gilbert and Song is explained in detail in Appendix B. To retain a certain level of readability, some processes in the script are explained in a simplified manner here. When questions arise regarding the functionality, the reader is referred to Appendix B.

Original Approach

In the original code by He, Gilbert and Song[32], a *Possible Member List*(PML) is assembled before the optimisation loop. This is done by dividing the design domain in a grid of evenly spaced nodes. A loop is then initiated, generating all possible members connecting these nodes. A schematic overview of this process can be found in Figure 4.1. Overlapping and duplicate

Figure 4.1: Forming a PML and an initial structure with two loops

elements are not allowed. During the process of the optimisation, a structure is generated from this PML. Members that are excluded from this list can never be present in the final structure. Therefore, it is chosen to implement the desired changes at this stage in the process.

In the original script, a second loop is initiated after assembling the PML. This loop activates an initial set of members before the first iteration of the optimisation. This is done by specifying a maximum length of $\sqrt{2}$. This allows only the horizontal, vertical and most simple diagonals. Such an initial structure for a design space of 20 x 10 nodes is illustrated in Figure 4.2.

Proposed Approach

To exclude members of a certain inclination in the design, the function *CreateGroundStructure* is proposed. This function is available in the *trussopttools* library(see Appendix E.3). This function is created as a substitute for the process as described in Figure 4.1. The function takes three inputs; design domain, minimum angle and maximum initial length. It performs the two operations necessary to obtain a PML and initial set in one loop. A flowchart representing these operations can be found in Figure 4.3.

Combining the two operations is necessary when a minimum inclination of 45° or more is required. In this case, the diagonals from Figure 4.2 are not in the PML. When a maximum

[1]As stated by Vincent Wegener, founder of Rotterdam Additive Manufacturing Lab (RAMLAB)

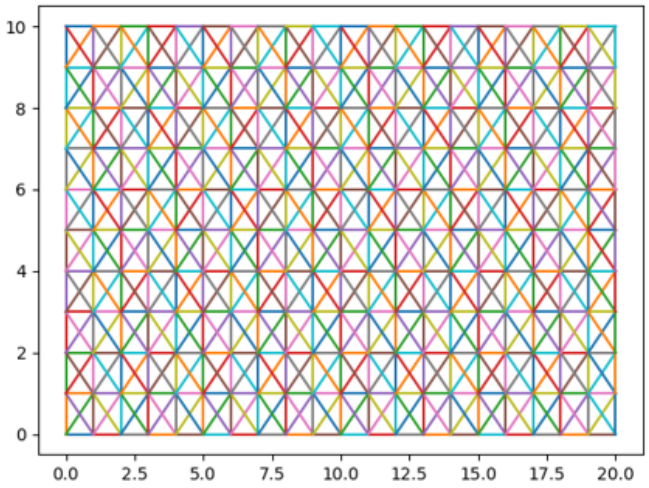


Figure 4.2: Initial set of members for a design space of 20x10 nodes

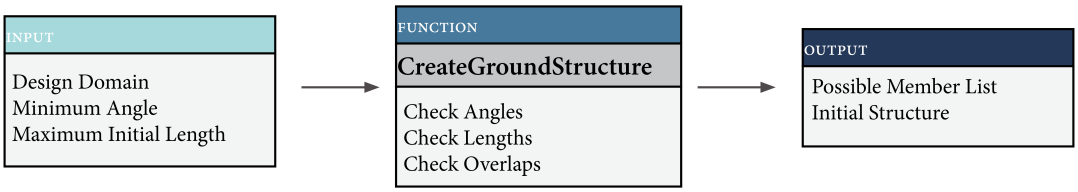


Figure 4.3: Proposed function for creating a PML and initial structure

length of $\sqrt{2}$ is used, only the vertical connections will remain in the initial set. This leads to a structure that is not able to transfer any realistic load. In these cases, the maximum initial length should be adjusted to a value which does allow diagonal elements. The same design space of 20 x 10 nodes, but now including an angle constraint of 50° and a maximum initial length of $\sqrt{5}$ can be seen in Figure 4.4.

4.1.2 Example with angle constraint

To highlight the functionality that was added in this section, an example is presented. The model space that will be analysed is presented in Figure 4.5. The results for two analyses are given below in Figure 4.6. On the left, the model space is optimised without any angle constraints. Like before, red elements are under tension and blue elements are under compression. On the right, members with an inclination lower than 25° are not allowed. It is clear that the horizontal members have now vanished. However, as can be seen on the bottom of the structure between the loads, the result shows some sort of horizontal connection with triangular elements. It can also be seen that the resulting volume of both models differs slightly. Including the angle constraint appears to require more material. This issue is addressed in more detail in Appendix C.

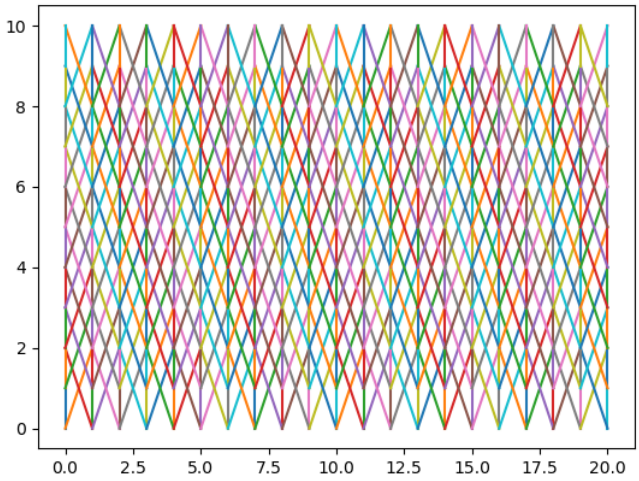


Figure 4.4: Initial set for a design space of 20x10, with a minimum angle of 50°and a maximum initial length of $\sqrt{5}$

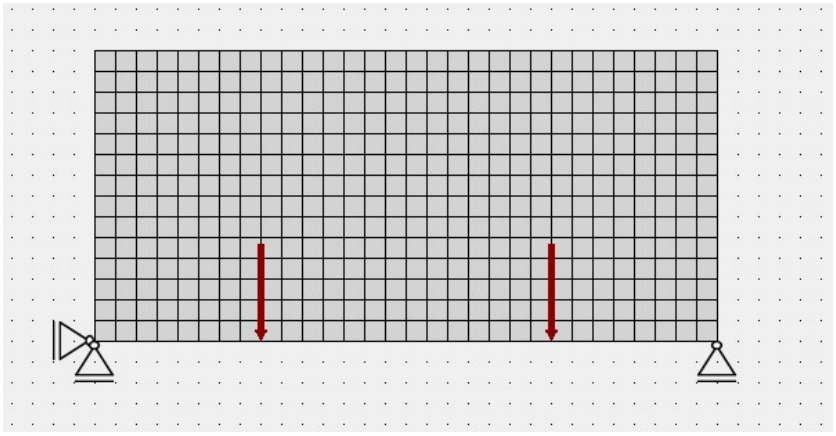


Figure 4.5: Model space example with two loads

4.2 Printing Orientation

As indicated in Appendix C, applying an angle constraint can significantly increase the amount of required material. In the example from the Appendix, this increase was found to be 36%. A comparison can be seen in Figure 4.7. Since the aim is to reduce material use, this is undesired. In the example from the Appendix, a better orientation was relatively easy to identify without any calculations. However, with increasing complexity of the model, a good orientation may not always be obvious. Therefore, another possible solution is presented here. This solution consists of an automated process to find a suitable printing orientation.

Starting point for this method is an undisturbed optimisation process. In other words, the optimisation can be performed without any limitations on member inclination. When the structure is optimised, the result is post-processed by a separate python script. The complete script can be found in Appendix E.2. The suggested approach is to rotate the entire structure around a point. This rotation is performed in small increments, using the procedure from Appendix D. In each increment, a *Performance Index*(PI) is calculated. This PI is defined as the ratio of

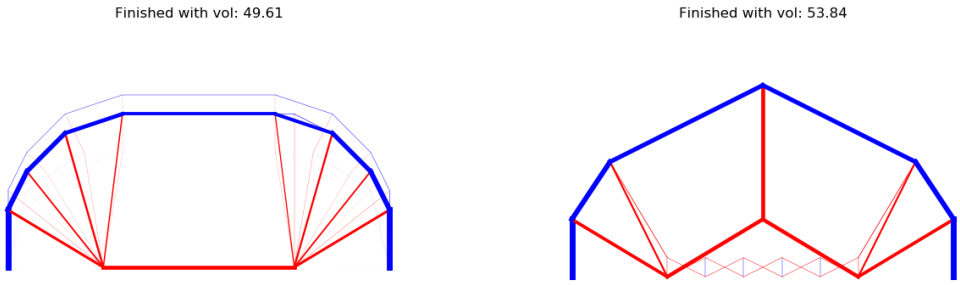


Figure 4.6: Conventional result(left) compared to an optimisation with an angle constraint of 25°(right)

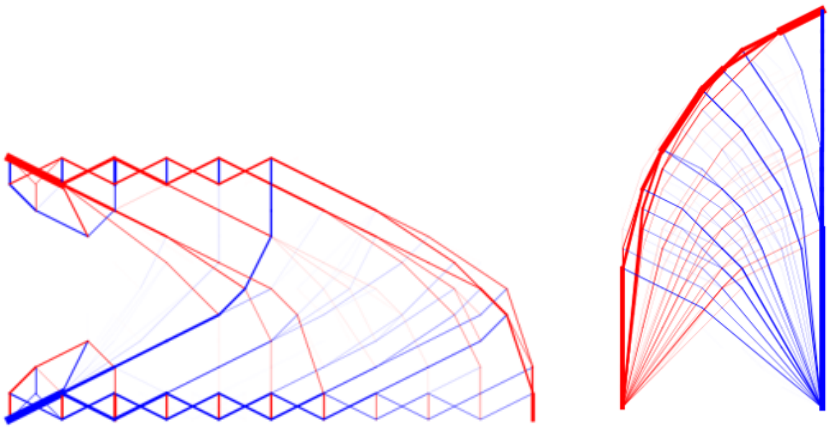


Figure 4.7: Two identical structures with an angle constraint to the horizontal of 25°, only rotated by 90°counter clockwise. Respective volumes are **99.98**(left) and **73.81**(right)

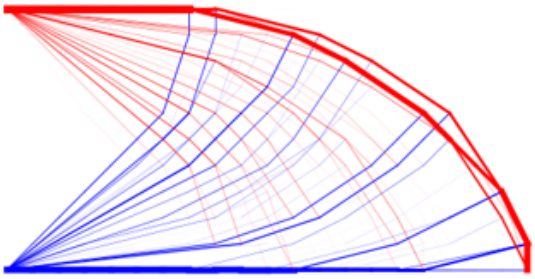
printable material, or:

$$PI = \frac{V_{printable}}{V_{total}} \cdot 100\% \tag{4.1}$$

The printable material is determined by evaluating the inclination of every member. In two-dimensional structures, it is assumed that the printing process is in positive y-direction. For each member, the angle between the x-axis and the member direction is calculated. If this angle is larger than the minimum printing angle, the volume of that member is added to $V_{printable}$. When all members are evaluated, this printable volume is compared to the total volume of the structure. The PI is then reported as a percentage of the total volume. This process is repeated for every rotation increment. The functionality of the script will be highlighted by two examples. The first example is a two-dimensional structure. The second example is a three-dimensional structure.

4.2.1 Example of a simple truss structure in two dimensions

To illustrate the functionality of the script in 2D, we consider the example from Appendix C. The result from the undisturbed optimisation process can be found in Figure 4.8. The output from this optimisation is automatically exported to a csv-file. This csv-file contains information



Property	Value
Width	20
Height	10
σ_t	1
σ_c	1
Joint Cost	0
Minimum Angle	0
Minimum Initial Length	$\sqrt{2}$
Steel Volume	38.96

Figure 4.8: Optimisation result for simple cantilever

on member location, area and internal force. We are only interested in printing the members under tension. Therefore, the blue members will be omitted for the next step. As previously established, the minimum printing angle is 60°. The incremental rotation for this analysis is taken as 5°. In theory, only rotations between 0° and 180 ° have to be analysed. However, in some cases it may be useful to have the full spectrum available. Members indicated in blue are printable. Members with an inclination smaller than the minimum angle are indicated in red.

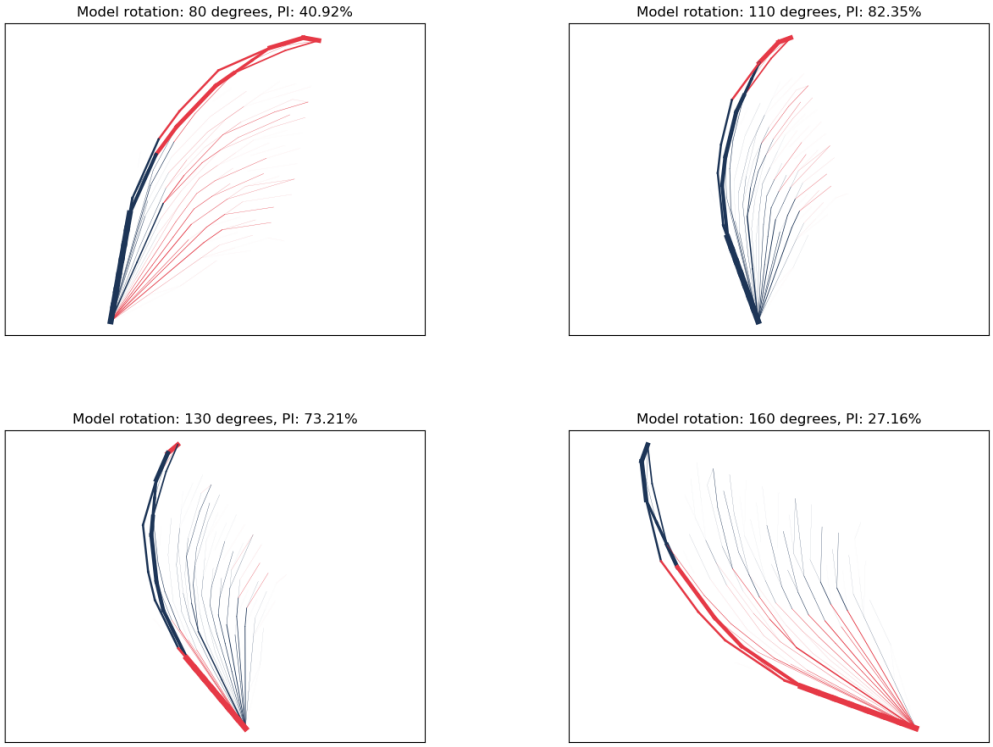


Figure 4.9: Rotating an optimisation result around the z-axis

Some intermediate results can be seen in Figure 4.9. The title of the plot displays the rotation angle and the *Performance Index*. When the analysis is complete, a plot is generated that relates all angles to their respective PI. The plot for this analysis can be seen in Figure 4.10.

The user is then prompted for a desired rotation. When a rotation is selected, an updated csv-file *updatedstruct.csv* is generated. This updated csv-file contains the new locations of the members, considering the given orientation around the z-axis. It is saved in the same folder as

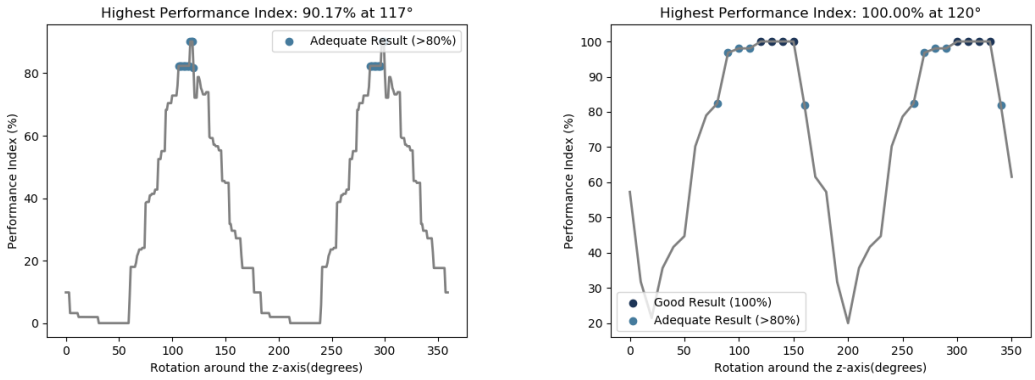


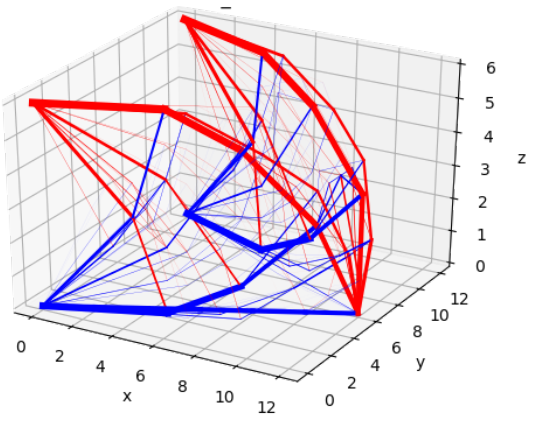
Figure 4.10: Analysis of 2D structure at given rotation around z-axis, for a minimum angle of 60°(left) and 25°(right)

the original *struct.csv*

It becomes clear that when a minimum inclination of 60° is required, there is no orientation in which all members satisfy the constraint. The highest PI is 90.17% at 117°. For illustration purposes, a report is also generated for a minimum inclination of 25°. This can be seen on the right side of Figure 4.10. Several orientations of the model now yield a PI of 100%, which are indicated in dark blue.

4.2.2 Example of a three dimensional cantilever

The functionality of the orientation finding script is also extended for 3D-applications. There is a single script for 2D and 3D cases. It is determined automatically whether the model in question contains 2D or 3D members. If a 3D model is recognised, a similar analysis to the previous section is performed. A model directory can be passed to the script, along with a minimum printing angle and a step size. This step size determines the angular increment at each step. For this example, the optimisation result of Figure 4.11 is used.



Property	Value
Width	12
Height	12
Depth	6
σ_t	1
σ_c	1
Joint Cost	0
Minimum Angle	0
Minimum Initial Length	$\sqrt{3}$

Figure 4.11: Optimisation result for a simple 3D cantilever

To print the model, no members should exceed the previously established inclination of 60°. This inclination is with respect to the horizontal plane. We assume the z-direction to be the printing direction, which makes the xy-plane horizontal. Now, to analyse the best orientation,

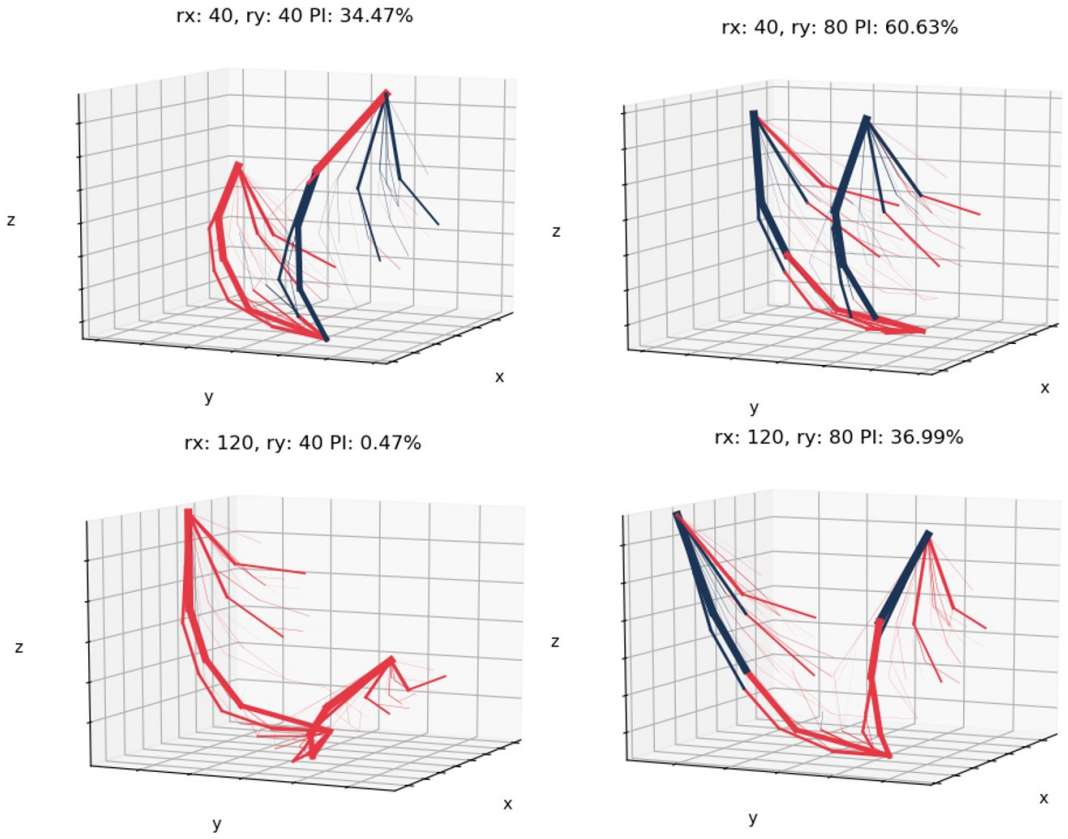


Figure 4.12: 3D model rotated around x- and y-axis, for a minimum printing angle of 60°

the model is rotated around the x- and y-axis. The process is the same as in 2D, but now needs to be rotated around two separate axes. This process is significantly more tedious than the 2D-example, since a lot more orientations have to be analysed.

Again, we only consider the members under tension. For the structure described above, some intermediate plots are given in Figure 4.12. The title of the graphs shows the rotation around the x- and y-axis, as well as the *Performance Index* of this orientation. Again, printable members are assigned a dark blue colour. Red elements have an inclination smaller than the specified minimum. When the analysis is finished, a report similar to the one in the previous chapter is generated. For a step size of 1°, this report is given in Figure 4.13 for two minimum inclinations. The report shows good performing areas in the rx, ry domain in darker shades. It is clear from the result that it was not necessary to analyse both rx and ry between 0° and 360°. However, analysing the complete spectrum is very convenient for the next sept in the process. A final model orientation has to be chosen. When the analysis is finished, the user is given a list of rotation pairs that have a high *Performance Index*. The user is prompted for an x-rotation and a y-rotation, upon which the structure is plotted in this orientation. This makes it possible to adjust the orientation, in case there are multiple rotation pairs that have the same PI.

Considering Figure 4.14 for instance, a structure that is flipped around the y-axis by 180° can be seen. This means the PI is identical, which can also be seen in the result of the analysis in Figure 4.13. Although the PI is unchanged, the structure at y-rotation 250° will be much easier to print. Members that are hanging downward are unprintable, because the printing is done from the ground up. This highlights a shortcoming of this analysis method, since this issue is not accounted for in the calculation of the PI. Allowing the user to adjust the printing orientation after the analysis partially solves this issue.

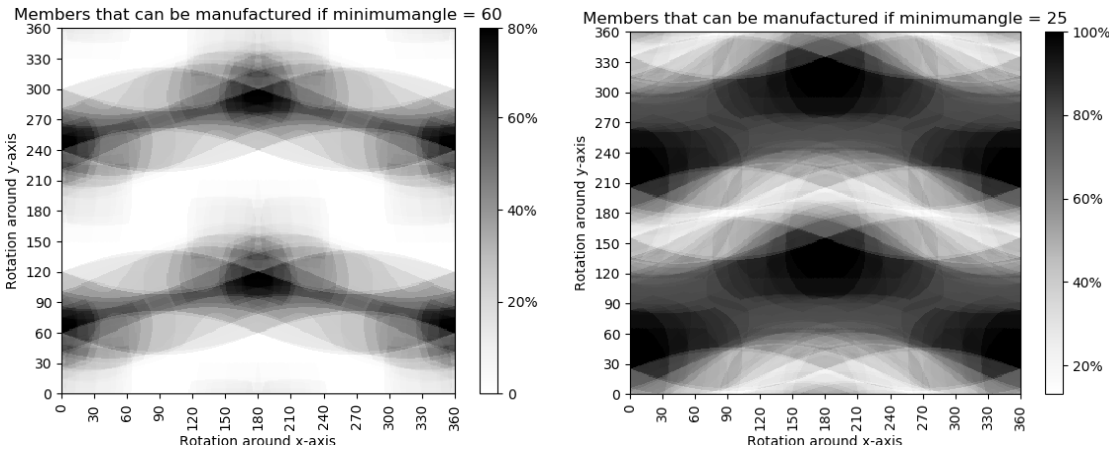


Figure 4.13: Analysis of 3D structure at given rotations around x- and y-axes, report for a minimum angle of 60°(left) and for a minimum angle of 25°(right)

Another observation that can be made is about the largest value of the PI. Similar to the previous section, a minimum inclination of 60° proves to be a relatively harsh constraint. When this constraint is applied, it leads to a structure that is not completely printable in any orientation. The highest PI in this example is only 78.73%.

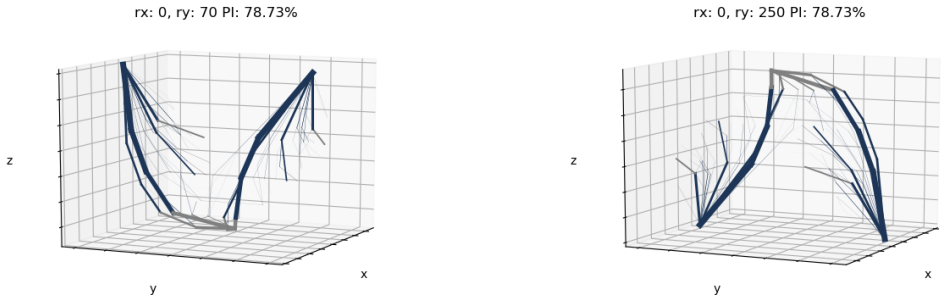


Figure 4.14: Comparison of two orientations with the same PI

4.3 Member Diameter

The other printing constraint that requires attention is the minimum member diameter. As mentioned previously, it was found that a minimum diameter of 1cm is necessary to ensure a successful printing process. This section discusses the possibility to include this manufacturing constraint in the optimisation process. To understand how this can be done, some more knowledge on the functionality of the script is required. This is explained in the first part of this section. The second part of this section uses this knowledge to obtain a method to adjust the minimum member diameter of the model.

4.3.1 The Member Adding Scheme

An in-depth explanation of the *member adding scheme* by He, Gilbert and Song[32] can be found in Appendix B, but the relevant steps are also explained here. As mentioned previously, the optimisation script assembles an initial structure before the optimisation loop is started. This initial structure is the starting point for the optimisation. When the optimisation loop is initiated, three operations are performed in each iteration. The first operation is to construct an equilibrium matrix, relating the externally applied load to the internal force distribution. This is done in matrix-vector form, with the forces in vector notation and the equilibrium relations in matrix notation. This notation is necessary for the next step, where the cross sectional areas are adjusted.

The external Python library CVXPY[36;37] is used for this next step. This convex optimisation tool is developed to solve a broad spectrum of optimisation problems. For the purpose of this research, it is sufficient to know that this library is able to optimise a list of variables, given a set of boundary conditions. In the case of the member adding scheme, this list of variables contains the cross sectional areas of the individual members. The boundary conditions are a limiting tensile and compressive stress. Together with the equilibrium matrix and the external force vector, CVXPY optimises the cross sectional areas. This yields a structure that uses the minimum amount of material to transfer the external load to the supports.

The third and final step of the iteration is evaluating the virtual strain of every member in the *Possible Member List*. This means that each member in the domain is evaluated, not only the ones present in the current structure. A specific strategy is then used to determine which members are added to the solution, this is explained in more detail in Appendix B. The result of this step is an augmented structure, where additional members are added to highly stressed areas. If no members need to be added to the domain, the process is terminated. Otherwise, the next iteration is started.

4.3.2 Implications for a minimum diameter

Due to the nature of the method that is describe above, demanding a minimum member diameter is not as straightforward as expected. In the member adding scheme, elements that are activated will permanently remain in the solution. If a certain member is no longer necessary for the force transfer, its cross sectional area is reduced to 0 by the CVXPY optimisation. However, it will remain activated in the solution. This means that if a minimum diameter would be imposed on every active member, a very inefficient structure can be expected.

Therefore, another solution is proposed. A post-processing script is presented here, that adjusts the member diameters when the optimisation is completed. It was expressed previously that this order is undesired, since it will most likely reduce the efficiency of the structure. However, due to the reasons described in the previous paragraph, it is considered to be justified in this scenario to investigate the possibility. Other solutions would implicate rigorous changes to the optimisation script.

4.3.3 A post-processing method for minimum member diameter

The method that is chosen here consists of a separate Python script. This script reads the contents of an optimisation result. This is done in similar fashion as before, by reading the output csv-file *struct.csv*. This csv-file contains all the information on member location, cross sectional area and internal force. The exact code can be found in Appendix E.4.

The script allows the user to specify a minimum member diameter. The minimum diameter is converted into a minimum area. A loop is then performed over all elements, and the area is adjusted where necessary. At the end of the loop, it is reported how much material was added. The updated model is saved in the same location as the input, with the filename changed to *originalname_bd**xx***, where ***xx*** is the chosen minimum bar diameter in millimetres.

No example of the script is given here, but readers interested in the functionality are referred to section 5.3.1.

4.4 Summary of the Methods

4.4.1 Angular Constraint

Two options were explored in light of the angular constraint of the additive manufacturing process. The first option presented a new method for forming the *Possible Member List*. The optimisation script works with a predefined set of members, assembled from all possible connections within the nodal domain of the model. By including a check that limits the minimum inclination of each member, a reduced set can be formed. This was done by introducing a new function for assembling the PML. This function is also able to activate the initial set of the optimisation in the same loop, by setting a user-specified maximum initial member length. This is required when the minimum member inclination exceeds 45°, since then the inter-nodal diagonals are no longer valid.

It is clear that this method is able to generate geometries where no elements exceed the minimum printing angle. However, applying this constraint has a significant negative effect on the material use in the model. As pointed out in Appendix C, up to 36% more material is needed to transfer the loads to the foundation. Therefore it was investigated whether this issue can be addressed by changing the orientation of the undisturbed optimisation result.

4.4.2 Printing Orientation

Deviating from the initial strategy to include the manufacturing constraints in the optimisation, a method was developed to evaluate the performance of the structure after the optimisation is finished. This was done by rotating the entire model around the origin and calculating a *performance index*. This index compared the volume of material that satisfies the angular printing constraint to the total volume. The proposed script analyses the complete rotational spectrum at user-specified intervals. This can be done for both 2D and 3D models. When the analysis is complete, a report is generated that shows the performance of the model at each interval. The user is able to choose a suitable orientation based on these results. The model is then rotated to this orientation and saved in a new csv-file.

This rotation analysis method allows for an undisturbed optimisation. Members with any inclination can be present in the solution, which enlarges the solution space of the optimisation. More force paths are allowed, which leads to more direct force flow to the foundation. This requires less material, so more efficient structures can be obtained. As was established above(see Figure 4.10 and Figure 4.13), this method is able to reliably report suitable printing orientations.

However, when the limiting minimum inclination is set to 60°, both examples that were discussed in this chapter were not completely printable in any orientation. The maximum *Performance Indices* were 90.2% and 78.7% for the 2D and 3D example respectively. This is expected to be insufficient for most practical applications. When 10-25% of calculated material is not manufac-

tured, the structure will likely not be able to carry the loads. Since the structure is optimised to a high degree, there is no redundancy of material. It is therefore not considered an option to omit a certain percentage of material from the result. Adjusting the model orientation alone will therefore not be sufficient for complex models, such as the results from this type of optimisation.

4.4.3 Member Diameter

The nature of the optimisation presents a challenge for including a minimum diameter. Elements that are not needed for force transfer are never removed from the solution. The *cvxpy* optimisation only reduces their area to 0. Imposing a minimum diameter is therefore not possible, since that would lead to a structure in which every member in the PML is always present in the solution. Changing this would require rigorous changes to the optimisation script. To overcome this issue, a different solution is proposed. This solution consists of a post-processing script, that can be used on any optimisation result. The script adjusts the diameters of every member that violates the user-specified minimum.

4.5 Answering the research question

This chapter has discussed several options to implement the manufacturing constraints in the optimisation process. Accounting for these specific constraints during the optimisation was expected to yield a high material efficiency of the structure. The research question that was considered is:

How can the printing constraints be included in the optimisation algorithm?

For the angle constraint it was found that it can be included, at the expense of higher material use. To a large degree this increase in material use depends on model specific parameters. In the example from Appendix C for instance, only a minimal amount of material had to be added when a suitable model orientation was chosen.

In some cases, a better alternative may be to use the method to find a good printing orientation. Using the report that is generated by this method, printing orientations can be found where a lot of members can be printed. However, the printing constraint of 60° was found difficult to overcome. For both the 2D and the 3D example, a significant amount of material proved impossible to be printed in any orientation.

The minimum diameter constraint is difficult to include in the member adding script by He, Gilbert and Song [32]. This optimisation algorithm does not allow for implementation of a minimum diameter of each element. However, post-processing of the results can be a suitable solution for designs that already use relatively large members. When a lot of smaller elements are generated in a model, adjusting the *joint cost* may offer a solution. This forces the result to be more simple, and small elements are suppressed.

4.5.1 Comments on the angular constraint

It was found in this chapter that the minimum inclination of 60° is a very harsh constraint. For most models that were investigated in this chapter, this constraint caused a large amount of members to be unprintable. This issue may raise questions regarding the feasibility of the proposed method. If this printing constraint causes issues for a lot of models, it can be argued

that combining the ground structure method with additive manufacturing processes is not very efficient. However, there are some arguments that can be raised that address this issue.

A lower member inclination

Printing slender, small diameter round elements is not common practice in 3D-printing. Therefore, it is not clear whether the 60° mentioned here should also be applied to this type of structure. It is expected that this inclination can be further decreased, possibly slightly decreasing the material properties of the printed material. The exact influence of this inclination on the printed result should be investigated further, considering for instance non-solid cross sections. Hollow tubes could change the characteristics of the material, possibly improving the strength when printed at an angle.

Rotating platform

Another solution for the angular constraint could be to print on a movable surface. Several examples exist where the base for a 3D-printer is non-stationary. Combined with the flexibility of the welding torch on the robotic arm, this may cause the angular constraint to disappear completely. When the printed model can be rotated around multiple axes, the member can always be placed in a suitable orientation. Of course, this would require advanced tool-path finding software. Advances in this field develop rapidly, and it is expected that methods like this are within reach.

Part 3

VALIDATION

Chapter 5

Case Study

This chapter presents a validation for the proposed methods from the previous chapter of this research. The research question that was proposed is:

- How efficient are the results of the extended optimisation method compared to the traditional calculation methods?

To answer this question this chapter is divided in five sections. The first section presents an introduction to the case with some background information. The second section contains an explanation of the traditional analysis procedure for a pile cap. The third section showcases the proposed method. In the fourth section, both methods are compared, both quantitatively and qualitatively. The fifth section presents an answer to the research question.

Figure 5.1: Architectural render (left) and photo (right) of RAI-hotel in Amsterdam, the Netherlands[2]

[2]Both images taken from *https://oma.eu/projects/rai-nhow-hotel*

5.1 Case Introduction

This chapter focuses on one of the pile caps under the recently finished RAI-hotel in Amsterdam. The RAI-hotel is situated on the southern main access road in Amsterdam, the Netherlands. The hotel is in close proximity to Amsterdams business district *de Zuidas*. The building is designed by architectural firm *OMA*, and realised by contracting combination of *Pleijsier Bouw* and *G&S Bouw B.V.*. *Van Rossum Raadgevende Ingenieurs B.V.* was responsible for the structural design. The structural model uses a concrete core, and steel truss structures for the cantilevering parts of the triangles. Realising a large building in such a dense location posed challenges for all parties involved. What made this project especially difficult, was the close proximity to the new metro line of Amsterdam. This required ground settlements to be kept to an absolute minimum, and allowed very limited space for construction of the foundation. The foundation is the subject of this case study. The layout of the foundation is pictured in Figure 5.2. This image shows the location of the concrete core of the structure, as well as the location of all the foundation piles. The pile caps used to spread the loads to the piles are also pictured. As an example, we will analyse one of the outer blocks that use 3 piles.

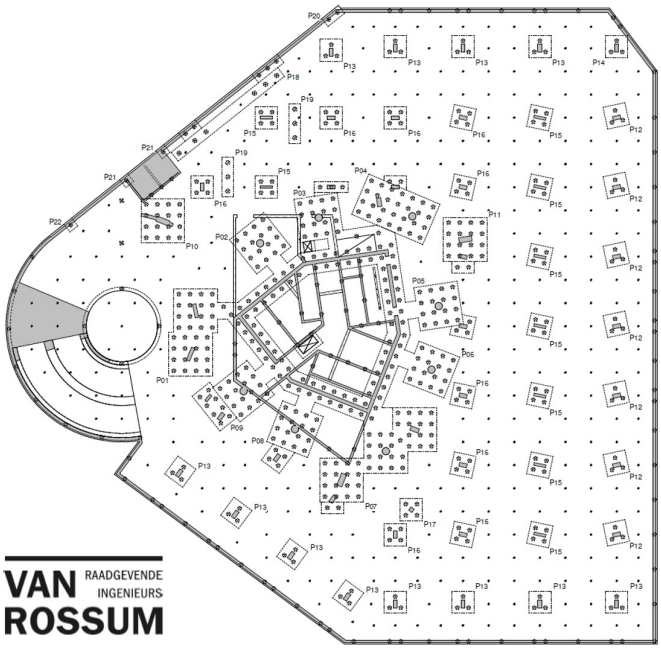


Figure 5.2: Foundation layout of the RAI-hotel

This pile cap is used to demonstrate the proposed method from previous chapter. The three-pole variant is chosen because it will contain a non-symmetrical stress distribution. The calculation method is involved, but not overly complicated, so it is a good guide to show the process. A third reason is that this type of pile cap is very common. As can be seen in Figure 5.2, the RAI-hotel also uses less common 16-pole and even a 32-pole block. Since this validation is a proof of concept, it is chosen to analyse the more commonly applied 3-pole block. However, it would be very interesting to see the same method applied to the larger, more complex blocks in the future.

The ability to use topology optimisation methods to obtain a reinforcement design for this 3-pole block is explained step by step. The result will be a model that is ready to be printed. This proposed analysis method will be compared to the traditional analysis method. Figure 5.3 shows the dimensions of a three-pole block. The three foundation piles are indicated with circles, and the column with a rectangle. The column is not placed in the centre of the block,

but in the centre of gravity of the triangle spanned by the three foundation piles. The outer dimensions of the block are 2600 mm x 2600 mm x 1600 mm.

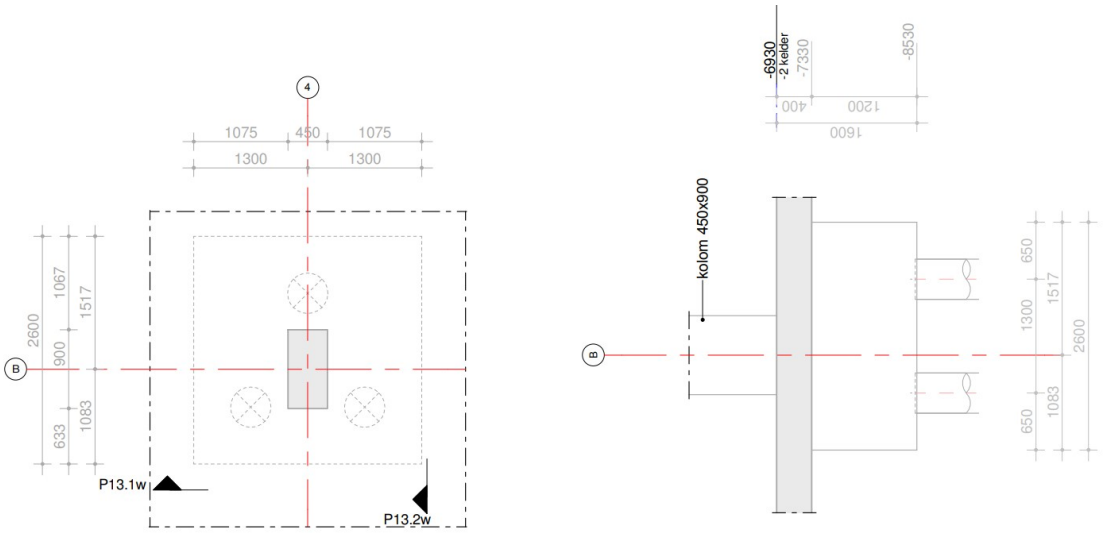


Figure 5.3: Top and sideview of pile cap

5.2 Traditional Analysis Method

First, we consider the traditional analysis method of such a pile cap. This traditional method follows the guidelines from Eurocode NEN-EN 1992-1-1 section 6.5. This section describes analysis methods for concrete elements with non-linear strain distribution. The calculation process for the pile cap that is considered in this section can be found in Appendix F. The analysis is split up in two parts; The strength verification (ULS) and crack width control (SLS). This case study is meant as a proof of concept. It is therefore beyond the scope of this research to consider the crack width control, an the focus will be on the strength verification of the member. The steps that are taken to obtain a valid reinforcement design can be seen in the flowchart in Figure 5.4. Each of these steps will be highlighted briefly in this section. As can be seen in the flowchart, the first step is constructing a valid Strut-and-Tie model. According to the aforementioned Eurocode, development of such a model may be done by "the adoption of stress trajectories and distributions from linear-elastic theory or the load path method". This leaves considerable freedom for the designer. Any model is valid, as long as equilibrium is ensured. In practice, often simple layouts are chosen to limit the complexity of the calculations. In this case, the STM is chosen as in Figure 5.5. In the left image, a side view is drawn. The right image shows a top view. The internal lever arm is indicated with z. This lever arm should be chosen such that enough concrete coverage can be ensured. It is also important that there is enough space around the nodes so that all forces can fully develop. Now the geometry of the model is know, the force distribution can be determined. This force distribution follows from equilibrium conditions. The member forces are then used to calculate the forces in nodes. This nodal check makes sure the assumed lever arm is valid. If more material above or below the nodes is necessary to transmit the forces, the lever arm should be adjusted. This is an iterative process, which is also indicated in the flowchart. When a valid internal lever arm is found, the angles between the struts and the ties are determined. If all angles between struts and ties are 55°or larger, the design compressive stress may be increased by 10% according to the Eurocode. Now all geometry and strength variables are determined, the individual struts and ties are analysed.

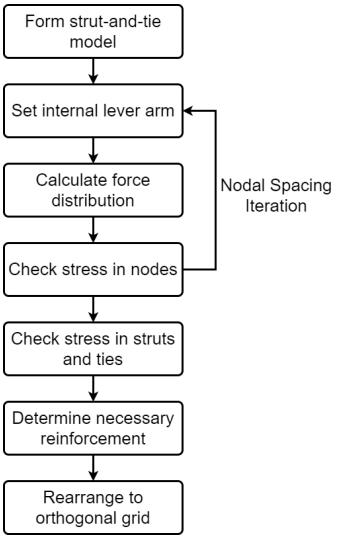


Figure 5.4: Flowchart traditional analysis method without crack width control

First we consider the struts. The concrete is verified in compression strength in a straightforward manner. However, The compressive forces in the struts may lead to significant splitting forces. The Eurocode presents the images in Figure 5.6 to clarify this behaviour. In this case full discontinuity is assumed. The acting design force is $F_{Ed} = 7650$ kN. The formulas presented by the Eurocode give us a splitting force, for which reinforcement should be designed. Because the strut is at an angle, reinforcement needs to be present in horizontal and vertical direction. As worked out in Appendix F, this needs to be:

$$\begin{aligned} \text{horizontal} : \varnothing 16 - 120 = 1676 \text{ mm}^2/\text{m} = 4356 \text{ mm}^2 \\ \text{vertical} : \varnothing 16 - 175 = 1149 \text{ mm}^2/\text{m} = 1494 \text{ mm}^2 \end{aligned} \tag{5.1}$$

The only ties are in the horizontal plane. This will require reinforcement in x- and y-direction. Following the calculation in the Appendix we can calculate the necessary volume of reinforcement to accommodate the tensile force in the tie. From the right image in Figure 5.5 it becomes clear that the force between B and C has an x- and y-component. The tie B-B only has a y-component.

$$\begin{aligned} \mathrm{A_x} &= 2023 \text{ mm}^2 \\ \mathrm{A_y} &= 2034 + 1032 = 3066 \text{ mm}^2 \\ \mathrm{V_{total}} &= \mathrm{A_x} \cdot \mathrm{l_y} + \mathrm{A_y} \cdot \mathrm{l_x} = 2023 \cdot 2600 + 3066 \cdot 2600 = 13231400 \text{ mm}^3 \end{aligned} \tag{5.2}$$

In the calculation in Appendix F, this amount is increased due to requirements from the crack width calculation. As of yet, the proposed method is not yet able to perform this calculation. There is also no functionality for calculating splitting reinforcement for the struts. As mentioned before, this case study is meant as a proof of concept. For this reason, any reinforcement present in the traditional method that is necessary for crack width control or splitting forces is omitted from the result. This is done to ensure fair comparison. The validity of this comparison is further discussed at the end of this chapter. For now, the results of the proposed method will be compared to the values from equation 5.2.

5.3 Proposed Method

The first step in the proposed method is determining the input parameters. It is chosen to model the area between the piles only. Because no forces will be present outside this area, it is

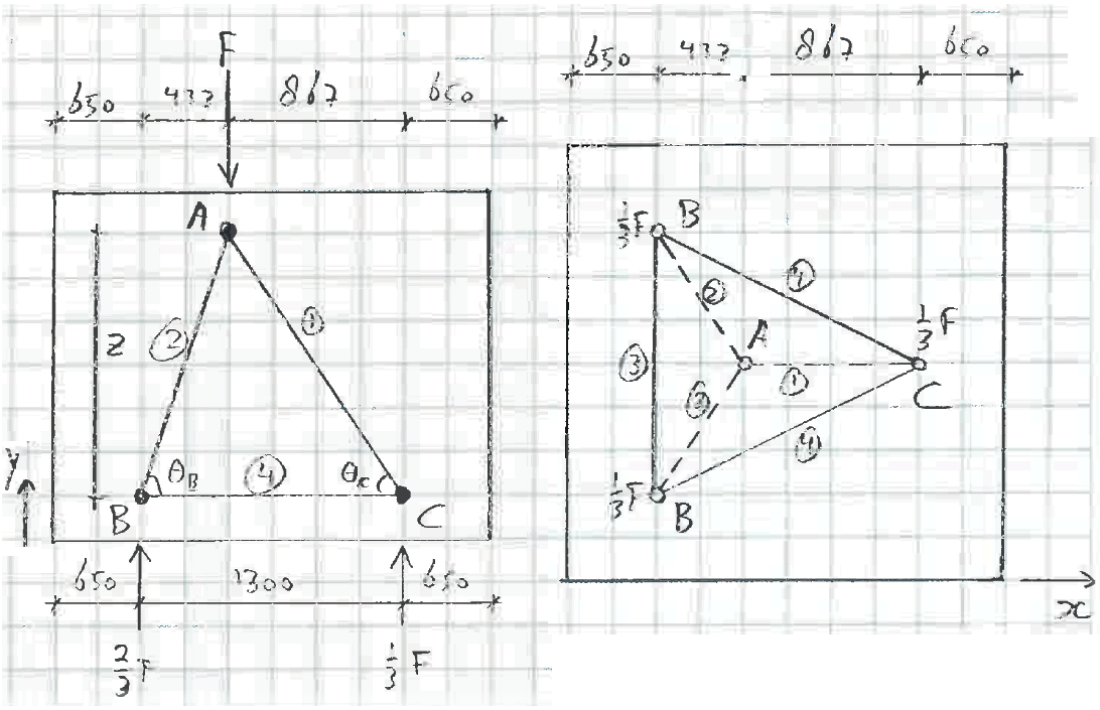


Figure 5.5: Strut-and-Tie model for pile cap

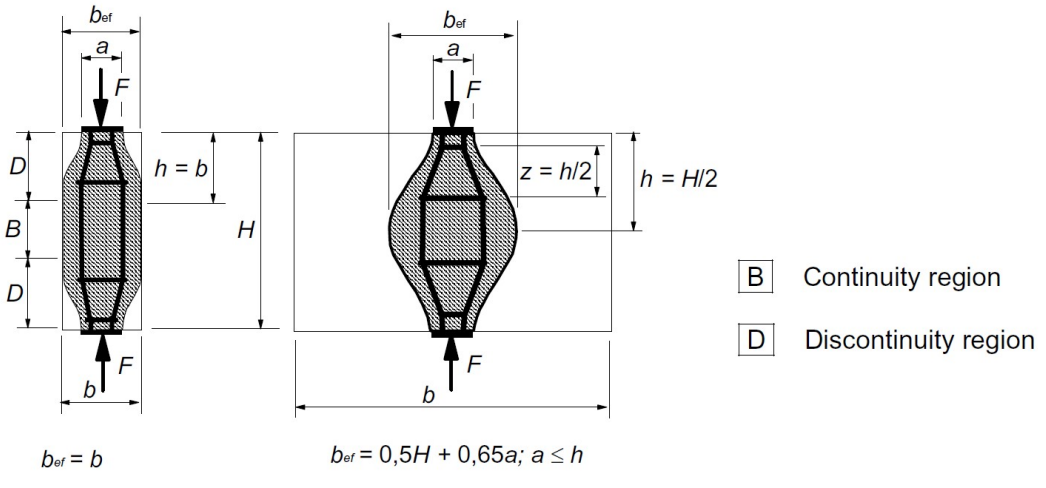


Figure 5.6: Splitting forces in struts, taken from Eurocode 1992-1-1 section 6.5.3

Parameter	Value
Width	12
Height	12
Depth	12
Nodal spacing	108.3 mm
F_{Ed}	7650 kN
$\sigma_{Rd,\ tension}$	0.435 kN/mm^2
$\sigma_{Rd,\ compression}$	0.435 kN/mm^2
Joint Cost	0

Table 5.1: Input parameters for Optimisation Run 1

unnecessary to model it. This square area enclosed by the piles measures 1300 mm x 1300 mm. The height of the block is 1600 mm. However, as we have seen in the traditional calculation from Appendix F, some of this height is needed to ensure sufficient concrete covering. The other thing that requires attention is the placement of forces and boundary conditions. These can only be prescribed at the nodes. As described in Appendix B, the optimisation script runs from a grid of evenly spaced nodes. To limit calculation time, the amount of nodes should be kept to a minimum. However, a certain level of accuracy has to be maintained. Using 10-20 nodes per axis will generally be sufficient. In Chapter 3 it can be seen that quite complex structures can be obtained in this way. If we were to use a design space of 1300 x 1300 x 1300 nodes for this scenario, the amount of possible members would be enormous(see Appendix A). Therefore we need some kind of conversion. The grid that we choose should also accommodate the placement boundary conditions and loads at the correct location. Looking at Figure 5.3, we we observe that the foundation piles are placed at 1/4 and 3/4 times the size in x-direction. The column is placed at 5/12 times the size in x-direction. If we choose a grid of 12 x 12 x 12 nodes, all forces and boundary conditions can be placed at exactly the correct location. The nodal spacing then becomes 1300/12=108.3 mm. All coordinates are multiplied by this conversion, to obtain the correct volumes.

The internal lever arm for the Strut-and-Tie model is taken as 1250 mm in the traditional calculation. This leaves enough room for covering, and also ensures there is enough material present around the nodes. This material is necessary to accommodate the high stresses in these areas. To model this in the proposed method, we have to choose between a height of either 11 nodes (11*108.3=1192 mm) or 12 nodes (12*108.3=1300 mm). For now, we use a height of 12 nodes (1300 mm) as a starting point for our calculation. The influence of this decision is determined later. This choice creates a cubic modelling space. We use the design force of F_{Ed} = 7650 kN because this is a strength calculation. The design strength of the reinforcement steel is taken as 435 N/mm^2.

The force is added as 7650 kN and the strength of the material in tension is set to 0.435 kN/mm^2. A joint cost of 0 is chosen, since all possible solutions are of interest. The input parameters are summarised in Table 5.3. Some intermediate plots, as well as the final result of the optimisation can be found in Figure 5.7. The result, on the bottom right in this Figure, is comparable to the STM that was used in the manual calculation. The main compressive struts, indicated in blue, are in roughly the same locations. However, three smaller compressive struts can also be identified. The grid of ties is fairly complex in the middle. This forms some kind of triangle. We see that the amount of steel that is necessary to accommodate the forces is 8781797.3 mm^3. This is without the splitting reinforcement needed for the struts, because this is not yet calculated. As of yet, the script doesn't offer any functionality to do this automatically. Also the manufacturing constraints are not yet accounted for. These constraints should be considered so a fair comparison can be made. This will be done in the following section.

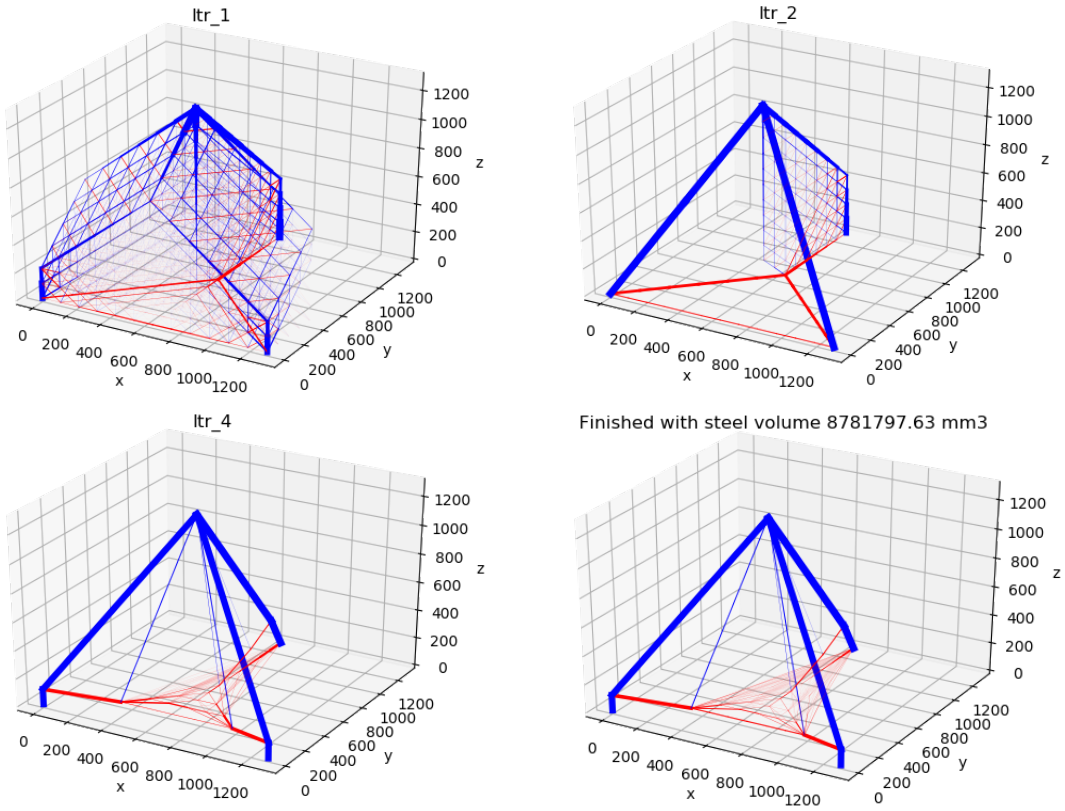


Figure 5.7: Result of Optimisation Run 1

5.3.1 Accounting for the manufacturing process

We know the minimum bar diameter needs to be around 10 mm to ensure a successful printing process. The structure should therefore be analysed so that the smallest bars can be enlarged. For this, the function *AdjustBarDiameters* from the *trussopttools* library is used. This function reads the output csv-file generated by the optimisation program. Each bar that has a diameter smaller than the given minimum is enlarged. An updated csv-file is saved in the same directory. The function also reports the original volumes, the adjusted volumes and the difference between the two. In Table 5.2 the output for this case is visualised.

	Original Volume	Adjusted Volume	Difference
Tension elements	$8.78 \cdot 10^6$ mm^3	$11.11 \cdot 10^6$mm^3	26.5%
Compression elements	$31.57 \cdot 10^6$ mm^3	$31.57 \cdot 10^6$ mm^3	0 mm^3

Table 5.2: Volume adjustments due to minimum bar diameter

We observe a significant increase of required material. This indicates that a lot of the elements that were present in the initial optimisation result had a diameter smaller than 10 mm. The fact that this diameter constraint increases the necessary material by a significant amount, raises questions about the input. This is a problem, considering the goal of achieving the best possible material efficiency. This should therefore be solved, which is attempted in the following section.

5.3.2 Further increasing material efficiency

We observe that the amount of steel necessary to print the structure greatly increases when the manufacturing process is taken into account. This is counterproductive, so options should be explored. As we have seen in the parameter study from Appendix A, almost all optimisation algorithms offer functionality to enforce simplicity. In the case of the *Member Adding Scheme* that is used here, it is possible to include a *joint cost*. This feature will suppress smaller members. This *joint cost* is now slightly increased to 0.1 and the optimisation is repeated. The result can be seen in Figure 5.8. This result is significantly more simple than the previous.

Parameter	Value
Width	12
Height	12
Depth	12
Nodal spacing	108.3 mm
F_{Ed}	7650 kN
$\sigma_{Rd,\ tension}$	0.435 kN/mm^2
$\sigma_{Rd,\ compression}$	0.435 kN/mm^2
Joint Cost	0.1

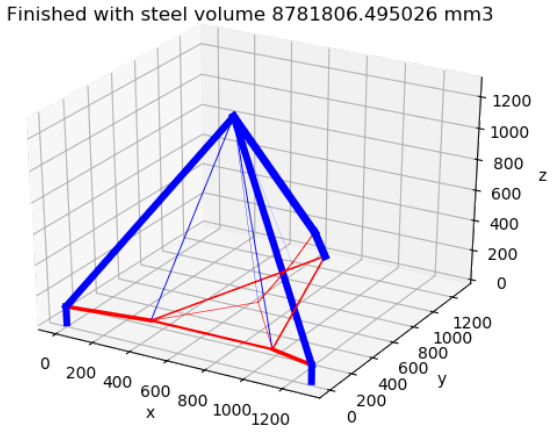


Figure 5.8: Input parameters and result of Optimisation Run 2 with *joint cost* = 0.1

The net of small tensile members inside the triangle has disappeared, and the diameters of the bars seem to have increased. The increase in steel volume is negligible, only 9 mm^3. When the *AdjustBarDiameters* function is run again, we see that all members satisfy the minimum bar diameter. Therefore no material has to be added to make the model printable.

5.3.3 Adjusting for the correct height

As mentioned earlier in this section, the internal lever arm of the strut and tie model was incorrectly set to 1300mm. The actual height that was used in the traditional design method is 1250 mm. Due to the setup of the optimisation script, it is not possible to choose this height for the optimisation. However, the goal is to make a fair comparison. Increasing the height will lead to lower tensile forces, which in turn will require less steel. It is therefore of interest to see if the amount of necessary reinforcement steel can be adjusted for this increased height. To this end, the graph in Figure 5.9 is presented. This is the result of Optimisation Run 3. The input for this run can be found in Table 5.3. In this graph, the solid circles indicate optimisation

Parameter	Value
Width	12
Height	[10, 11, 12, 13, 14]
Depth	12
Nodal spacing	108.3 mm
F_{Ed}	7650 kN
$\sigma_{Rd,\ tension}$	0.435 kN/mm^2
$\sigma_{Rd,\ compression}$	0.435 kN/mm^2
Joint Cost	0.1

Table 5.3: Input parameters for Optimisation Run 3

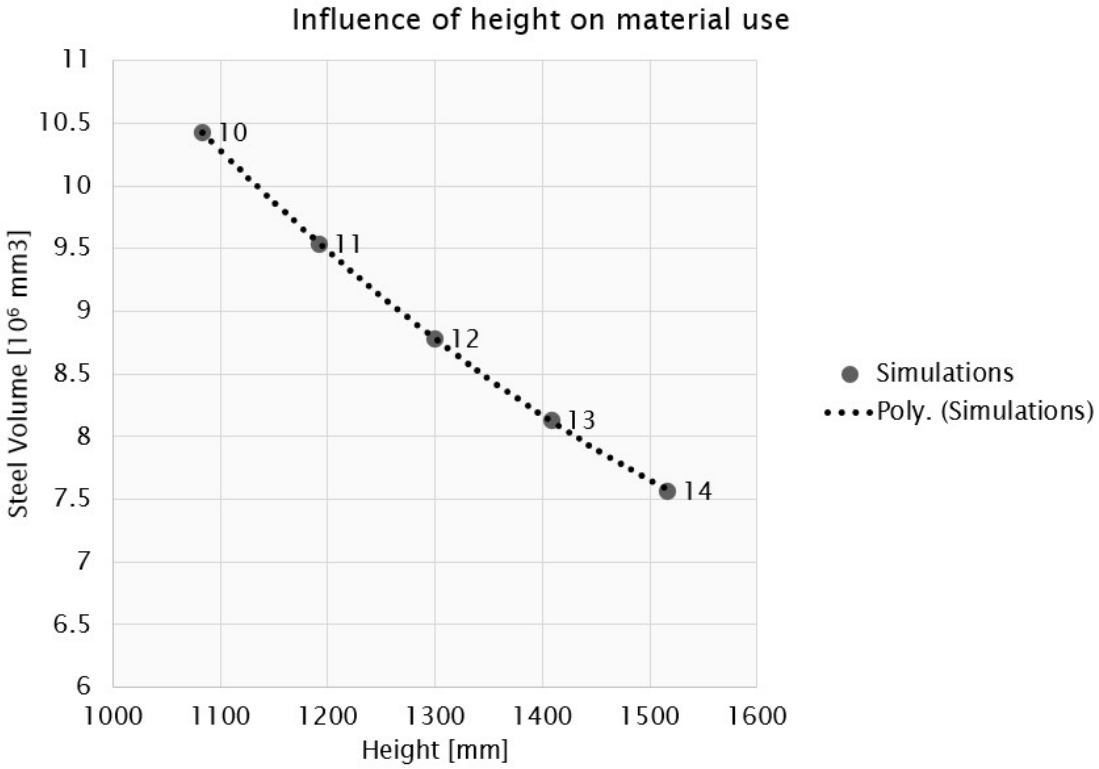


Figure 5.9: Influence of height on case study model with a nodal spacing of 108.3 millimeters

runs. 5 runs are performed, where the label indicates the amount of nodes in z-direction. The run with 12 nodes, and thus a height of 1300mm, is taken as the middle one. Two runs with increased height, and two runs with decreased height are performed to visualise the influence of changing this parameter. A second order polynomial, plotted as a dotted line, appears to fit very well. This graph now allows us to interpolate the necessary reinforcement in the proposed method for a height of 1250 mm. Reading the graph, this amounts to 9.2 * 10^6 mm^3.

5.3.4 The Angle Constraint

To investigate the effect of the angular constraint, two methods are used. The first optimises the domain, now including a minimum member inclination of 60°. The result of this analysis, which uses the same input parameters as in Figure 5.8, can be seen in Figure 5.10.

As can be seen in this figure, a very inefficient structure is generated. The amount of material needed to accommodate the tensile forces is found to be 35.9 * 10^6 mm^3, which is about four times the material that is needed without the angle constraint.

The second method that will be shown here, is automatic determination of a good printing orientation. The script from section 4.2 is used for this procedure. Analysing the result from Figure 5.8, the report is shown in Figure 5.11.

The highest *Performance Index* that was found is only 49.5%. In other words, in the best performing orientation of the model, only half of the elements can be printed. This is unacceptable, and would lead to a structure that is not valid.

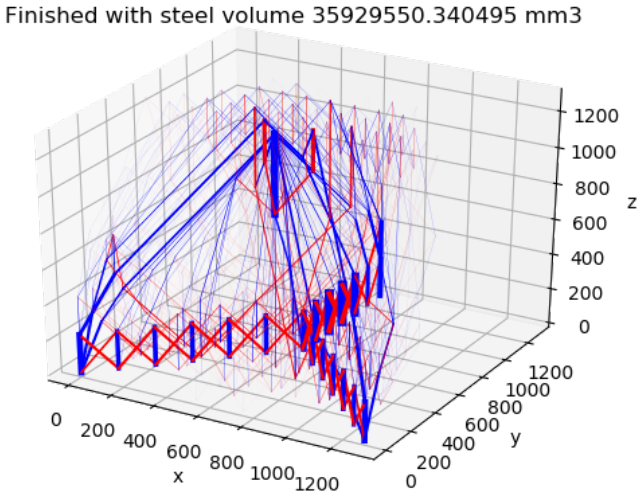


Figure 5.10: Optimisation result including an angle constraint of 60°

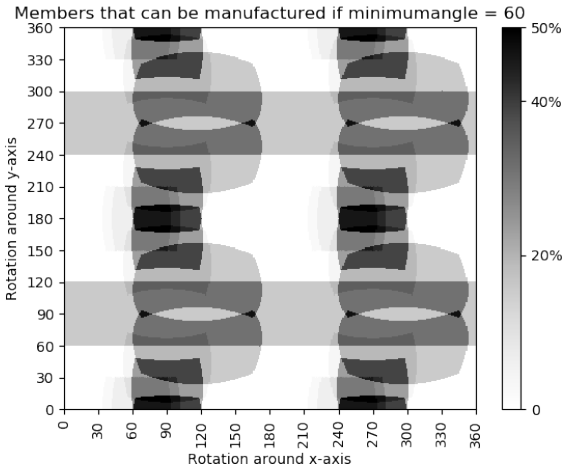


Figure 5.11: Rotation analysis of case study result

5.4 Comparison and discussion of methods

For both methods, the amount of steel necessary to obtain a valid design is now known. These are summarised in Table 5.4. Using the proposed method, a material reduction of 30% can

	Traditional Method	Proposed Method	Difference
Amount [mm^3]	13.2·10^6 mm^3	9.2 ·10^6 mm^3	
Amount [m^3]	0.0132 m^3	0.0092 m^3	
Weight[3] [kg]	103.9 kg	72.2 kg	-30%

Table 5.4: Comparison of necessary reinforcement between methods

be achieved. Of course, several important aspects of a strut-and-tie model calculation are omitted to obtain this comparison. As mentioned before, the proposed method does not account for splitting force reinforcement or crack width control. To be able to make a comparison

[3]With $\rho_{steel} = 7850$ kg/m^3

between the original and the proposed method, this part of the reinforcement is omitted from the traditional calculation. Although a separate investigation is necessary for a complete and inclusive comparison, several notes can be made about this beforehand. Let us consider the reinforcement necessary for accommodating the splitting forces. When we compare Figure 5.5 and 5.8, we observe that the general shape of the strut and tie model is similar. This similarity becomes more apparent when we only consider the struts. This indicates that the vertical force is transmitted to the supports in roughly the same manner. Although no conclusive statements can be made about this, it is expected that the amount of necessary splitting reinforcement will be similar. Thought even further, it would be interesting to explore possibilities to include the splitting reinforcement in the printable design. Printing the splitting reinforcement would allow for exact placement, without the need to rearrange to an orthogonal grid.

Another note that can be made about this comparison considers the crack width control. Crack width calculations ensure that the steel reinforcement is not affected by environmental influences such as corrosion. First and foremost, the concrete covering following from the environmental class of the concrete is there for this reason. However, when concrete is loaded in tension, minor cracks develop that may penetrate towards the reinforcement. When this happens, the concrete covering may not be sufficient anymore to prevent corrosion of the steel. The Eurocode[4] prescribes methods to check whether a structure is sensitive to this behaviour. From these sections it becomes clear that two aspects have a large impact on the crack that occurs; bar diameter and bar spacing. In general, a larger bar diameter and a larger bar spacing lead to a larger crack. If we project this on this case study, we observe that both parameters are larger in the original method. Opposed to the evenly spaced, larger diameter reinforcement bars that are used in this method, the proposed method shows a finely woven net of reinforcement ties between the foundation piles initially. In the example in this case study, this changes when the joint cost is adjusted. The bars are then spaced further apart, and the diameters increase. However, if we compare this to the traditional Strut-And-Tie model, we still see a higher number of members and therefore a smaller average bar diameter. Again, no definitive statements can be made about this difference and further analysis is necessary to calculate the crack width. However, considering the knowledge about bar diameter and bar spacing, we expect to see a smaller crack when the proposed method is used.

The third and final subject that is interesting to discuss in this context, is calculation time. As becomes clear from the calculations in Appendix F, the original method is very tedious. If we exclude the images and pages about input parameters, nine pages are required to calculate the necessary reinforcement. In the case of this pile cap, these calculations are performed manually. Several iterative processes are performed, leading to even longer calculation times. It is difficult to quantify the exact amount of time that was required to perform these calculations, but it is estimated that at least three working hours were spend on the manual calculation alone. If we compare this to the proposed method we see a large difference. Running the optimisation loop alone takes around 90 seconds[5]. Of course time is needed to prepare the input and process the output, but it is clear that large time savings are possible. This is before we have discussed the time necessary to model the pile cap. In the original method, a modeller converts the technical drawings from the engineer into 3D-models. At *Van Rossum Raadgevende Ingenieurs B.V.*, this is done in *Autodesk Revit*. These 3D-models are in turn used to generate drawings for the construction crew. If we extend the proposed method to this step in the design process, we observe good possibilities to integrate the two. The result of the proposed method is a structure that is recorded in a csv-file. We have seen in Chapter 4 that this structure can easily be imported in *Rhinoceros*. Therefore, no problems are expected for importing the result in *Revit* in a similar manner.

[4]See NEN-EN 1992-1-1 sections 7.3.3 and 7.3.4

[5]Using an Intel Core i7-8550U @1.8GHz

5.5 Answering the research question

The research question that was proposed is:

- How efficient are the results of the extended optimisation method compared to the traditional calculation methods?

It was proven that, using the proposed method, up to 30% less material is needed to obtain a valid strut-and-tie model. Although several simplifications were used to make this a feasible case study, it is proven in the previous section that the proposed method is a valid alternative to the traditional calculation procedure. In terms of material use and calculation time, large savings can be achieved. A valid strut and tie model can automatically be determined using the proposed method.

However, the printing constraint regarding the minimum member inclination is reason for concern. Strictly adhering to this requirement, the strut and tie model from this case study proved to be unprintable. The angles between the members of the strut and tie model are generated in such a way, that no valid printable orientation can be found. As argued in the previous chapter however, this is not necessarily a reason to condemn this proposed method. Developments in the field of additive manufacturing show promise for printing at a wider range of inclinations. For instance, a rotating printing surface may offer a solution.

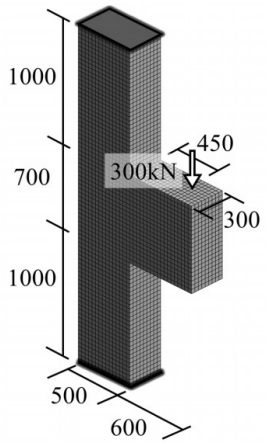
Chapter 6

Showcase

Opposed to the thorough comparison of the proposed method to the traditional analysis procedure in the previous chapter, this chapter aims to give a more general perspective on the proposed design strategy. Several examples are presented here, that show the functionality and possible uses of the method. The level of in-depth discussion is purposely kept to a minimum, to provide a better understanding of the possibilities that this ground structure approach brings. The first example will consist of a three dimensional corbel, based on the design of research by Xia, Langelaar and Hendriks [38]. The second example is a three dimensional cantilever, that is analysed both with and without an angle constraint on the individual members. The third and final example is a two dimensional wall with openings.

6.1 Three dimensional corbel

A frequently occurring design problem where strut-and-tie models are used, is the corbel layout. Corbel layouts are generally used in concrete construction, where a horizontal beam needs to be connected to a vertical column. For this example, we make use of the simplified corbel design as mentioned by Xia, Langelaar and Hendriks [38]. The finite element model from their research can be seen on the left in Figure 6.1. The table in Figure6.1 shows the input values that are used in Python for the extended optimisation. No physical meaning is given to the material strength, length scale and magnitude of the force, but only the aspect ratios and the location of the force are used.



Property	Value
Width	11
Height	27
Depth	3
σ_t	1
σ_c	1
Joint Cost	0
Minimum Angle	0
Minimum Initial Length	$\sqrt{3}$
Results	
Iterations	8
Time	92 seconds

Figure 6.1: Corbel, taken from Xia, Langelaar and Hendriks [38]

Figure 6.3 compares the optimised strut-and-tie model obtained by Xia, Langelaar and Hendriks to the result obtained by the method developed previously in this research.

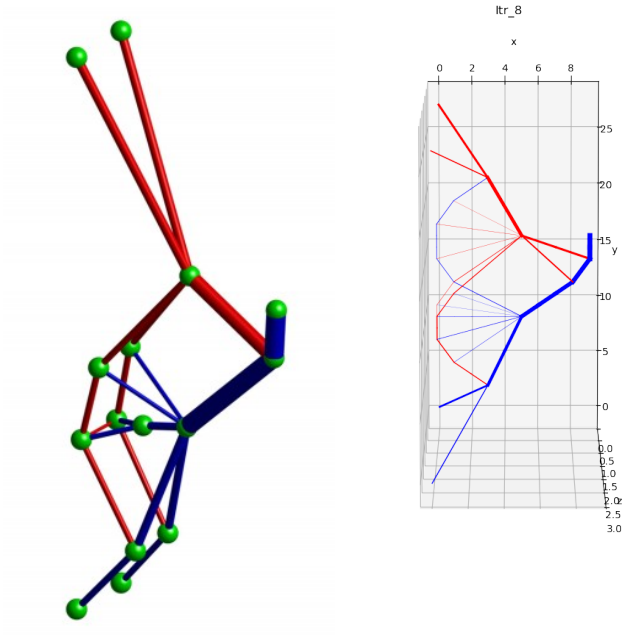


Figure 6.2: Result from Xia, Langelaar and Hendriks [38] (left) compared to the results from this research(right)

6.1.1 More complex loading scenario

Next to this relatively straightforward load case, it is interesting to see how the model performs under more complex forces. To this end, a lateral load is applied at the same position as the vertical one. This scenario is similar to the *complex load combination*, taken from Xia, Langelaar and Hendriks [38]. The comparison can be seen in Figure 6.3.

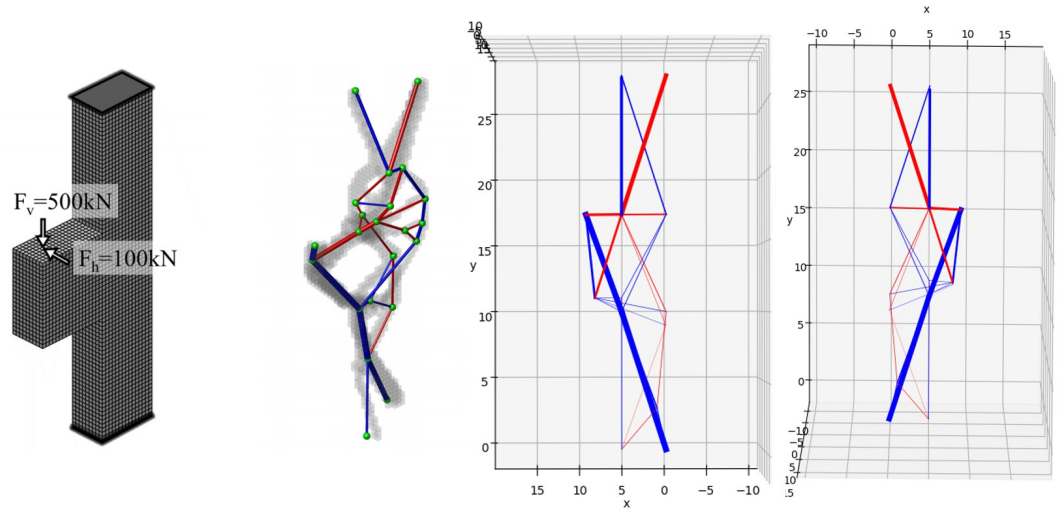


Figure 6.3: Comparison of result from Xia, Langelaar and Hendriks [38](left) to the newly developed method(right)

6.2 Cantilever with and without angle constraint

In this example a three dimensional cantilever is analysed. The first analysis is without any angle constraint. In the second analysis, the minimum angle is set to 25°. The quantitative input and results can be seen in Table 6.1. The cantilever has dimensions 12x12x6 and a point load in negative z-direction is applied to point (12,6,0).

Property	Value
Width	12
Height	12
Depth	6
σ_t	1
σ_c	1
Joint Cost	2
Minimum Angle	0
Minimum Initial Length	$\sqrt{3}$
Results	
Volume	682
Iterations	12
Time	42 seconds

Property	Value
Width	12
Height	12
Depth	6
σ_t	1
σ_c	1
Joint Cost	2
Minimum Angle	25°
Minimum Initial Length	$\sqrt{3}$
Results	
Volume	1090
Iterations	18
Time	73 seconds

Table 6.1: Input and results without angle constraint(left) and with an angle constraint of 25°(right)

Some intermediate plots and the final result for the analysis without angle constraint can be seen in Figure 6.4. The same can be seen for the analysis with angle constraint in Figure 6.5.

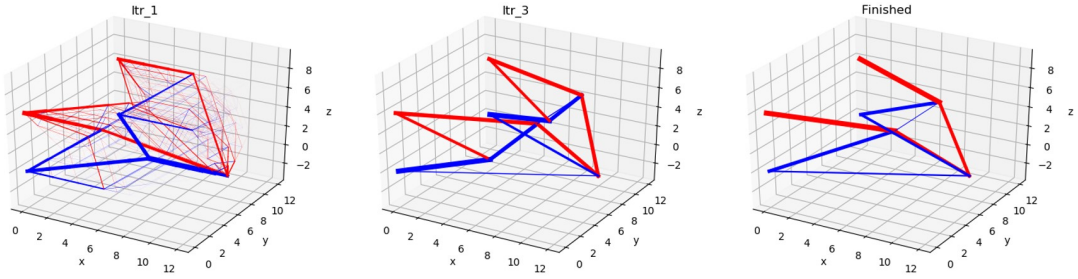


Figure 6.4: Intermediate plots and result for analysis without angle constraint

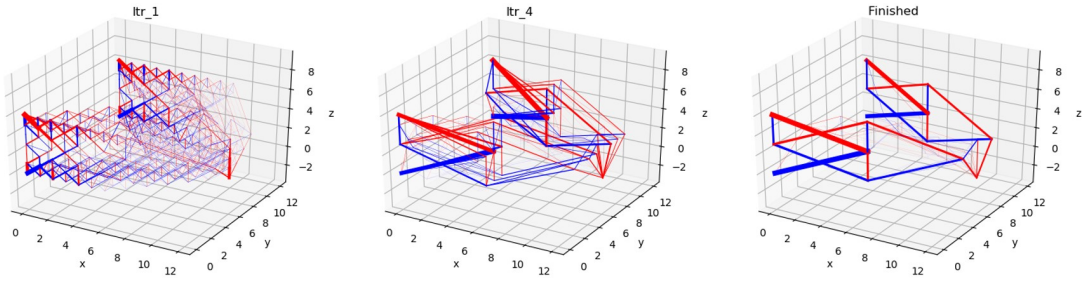
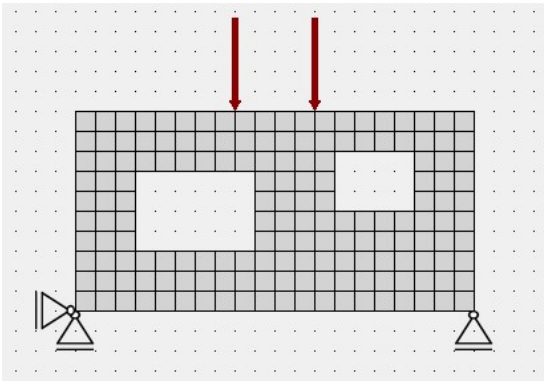


Figure 6.5: Intermediate plots and result for analysis with an angle constraint of 25°

6.3 A wall with openings

Another functionality that will contribute towards practical implementation of the new method, is the ability to assign passive areas in the design. These passive areas can be seen as voids or openings. An example of designs in which this is required are walls with openings for windows or doors. These are commonly analysed with the strut-and-tie method, due to the highly non-linear distribution of the internal forces. The structure that will be analysed in this example is pictured in Figure 6.6.



Property	Value
Width	20
Height	10
σ_t	1
σ_c	1
Joint Cost	3
Minimum Angle	0
Minimum Initial Length	$\sqrt{2}$
Results	
Iterations	22
Time	76 seconds

Figure 6.6: Input structure and parameters for a Wall with openings

The result of the optimisation can be seen in Figure 6.7 and Figure 6.8. This result shows a high level of complexity, especially in the small area between the two openings. It is evident that analysing this strut and tie model manually would be very time consuming. Also, determination of this bar layout without any computational aid would be virtually impossible. This is a good example of why the ground structure method could increase design efficiency significantly.

Automatic generation of complex strut-and-tie models is now possible, and is only minor steps away from practical implementation.

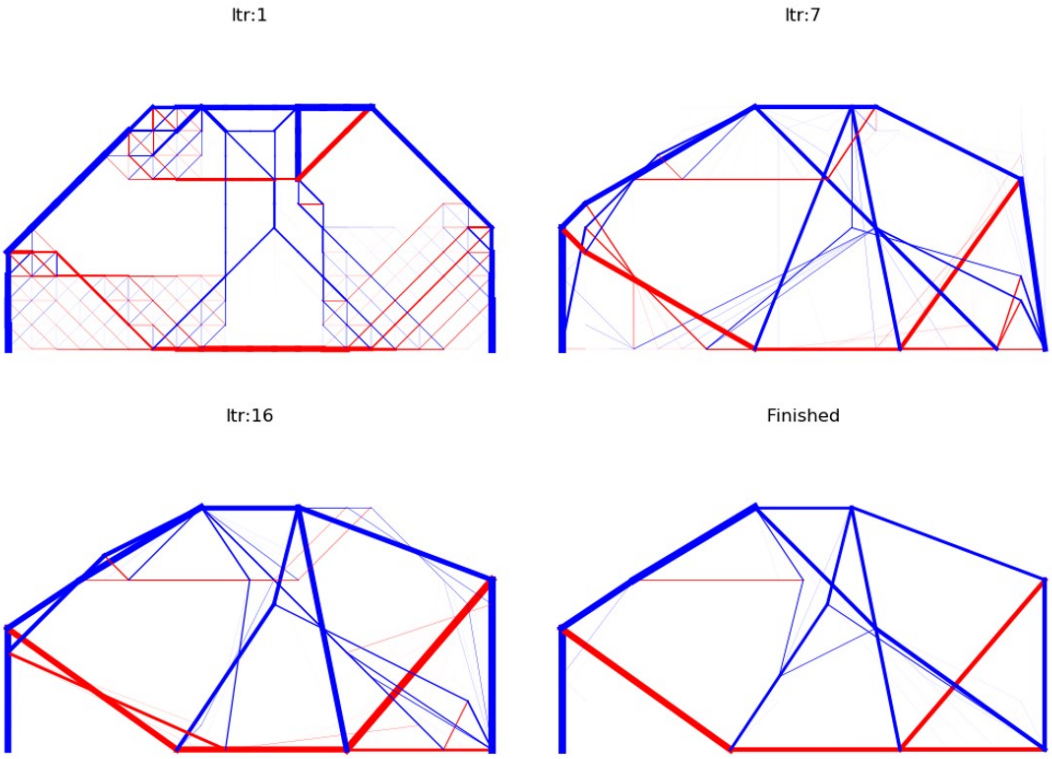


Figure 6.7: Optimisation result of wall with openings

Figure 6.8 shows a comparison between the analysis without and with the openings.

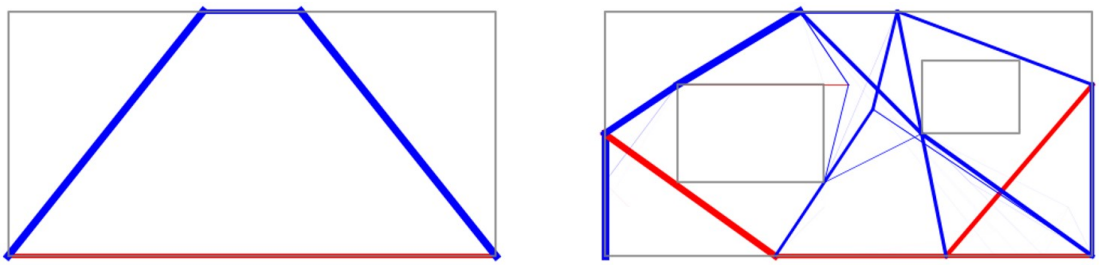


Figure 6.8: Comparison of optimisation result without openings(left) to result with openings(right)

Chapter 7

Conclusions

This chapter contains an overview of the findings in this research. All the research questions that were proposed in the first chapter are discussed here. The aim of this research was to investigate possibilities to integrate structural optimisation and additive manufacturing processes. Previously considered to be unfeasible due to the high level of model complexity, structural topology optimisation was rarely used in practice. Advances in additive manufacturing techniques have now opened new doors, which may pave the way for optimised, 3D-printed structural components. Specifically interesting in this light is automatic generation of strut and tie models for reinforced concrete. This method uses a substantially tedious calculation procedure to obtain a reinforcement design. The efficiency of the method is also very dependent on a good initial guess of force flow within a concrete element. Topology optimisation methods may offer a solution for these issues. To investigate the possibilities, several research questions were proposed in the first chapter of this research.

For each question, the investigation that was performed in the main part of this research is briefly summarised. The process is described that was used to arrive at a suitable answer to the question. This chapter uses the same layout as the entirety of this report, which consisted of three parts. The first part, the literature review, was discussed in Chapter 2 and Chapter 3. These chapters covered the theoretical background of both additive manufacturing and structural optimisation. A foundation was laid down on which the next part of the research could be performed.

This next part considered the extension of an optimisation algorithm, to accommodate the manufacturing constraints of an additive manufacturing environment. Several changes and additions were proposed to an optimisation algorithm that uses the *Ground Structure Method*. These modifications included methods to deal with process-specific constraints, limiting the solution space of the model. This is done in Chapter 4, where the thought processes and considerations needed to obtain suitable solutions are explained in detail.

The third part of this research consists of a validation of the modified optimisation method, to investigate the practical feasibility in a structural design context. To do this, a case study was performed that compared the proposed method to a traditional design process. This gave insight in the possibilities that the proposed method offers, as well as some issues that still require attention and further development.

This chapter is divided in four sections, following the layout that is explained above. The first three sections discuss the research questions related to the specific part of this report. In the fourth and final section, an answer to the main research question is formulated.

7.1 Part 1: Literature Review

Two subjects were investigated in the literature review. Both additive manufacturing and topology optimisation methods were discussed, and relevant previous research was highlighted.

7.1.1 Part 1a: Additive Manufacturing

The first research question that was proposed is:

Which techniques are available to print steel, and which is most suitable for reinforcement?

Two main techniques for industrial-scale printing of metallic components can be identified. *Powder Bed Fusion*(PBF) and *Wire and Arc Additive Manufacturing*(WAAM). PBF is a submerged method, where a powder bed is continuously fed in which the model is built. The powder is molten in specific locations by a laser, which creates solid material upon cooling down. The WAAM technique uses a robotic arm with a welding torch attachment. Conventional arc welding, usually MIG/MAG welding, is used to build a model from base plate. Only WAAM is considered suitable for the purpose of printing reinforcement layouts, since the powder bed method is not capable of producing large enough elements.

How are material parameters influenced by the different manufacturing processes?

Both micro- and macro-scale differences can be observed when 3D-printed material is compared to material from traditional manufacturing techniques. For the powder bed technique, some increases in yield strength and ductility were observed. However, this came at the cost of anisotropic effects. For the WAAM technique, several studies revealed a slight reduction of yield strength and ultimate strength compared to cast or wrought material. Also in this technique anisotropic effects were observed. Several solutions have been proposed to increase the performance of printed material, including post-process heat treatment.

Which manufacturing constraints should be accounted for?

Two constraints should be taken into account. To prevent the printed elements from tipping over, a minimum angle of 60° from the horizontal is advised. This angle can also be chosen to be slightly smaller, but this is expected to cause a significant reduction of strength. Therefore, the angle of 60° is maintained in the rest of the research. The other printing constraint that should be kept is a minimum element diameter. When considering circular cross sections, the diameter should always be 1.0 cm. This ensures a smooth printing process, and retains the expected material parameters.

How should the geometry be prepared?

In theory, any 3D model can be processed to a format that is suitable for the printing software. The conventional file formats used by 3D software such as Rhinoceros or Grasshopper can be used. An example of such a format is a stereo-lithography(.stl) file.

7.1.2 Part 1b: Topology Optimisation

The second subject that was discussed in the literature review is topology optimisation. Topology optimisation concerns with finding the most efficient material distribution within a pre-

scribed design domain. In this research, several algorithms were analysed in light of the automatic generation of strut and tie models for reinforced concrete. The research question was:

Which optimisation algorithm is suitable for analysing reinforced concrete, and can easily be modified to include manufacturing constraints?

Three different solution schemes were discussed; *Bi-directional Evolutionary Structural Optimisation*(BESO), *Solid Isotropic Material with Penalisation*(SIMP) and *Ground Structure Optimisation*(GSO). For all three, the theoretical background was summarised and the functionality explained on the basis of an example. Their suitability for implementation in strut and tie modelling was discussed, and previous research on this subject was reviewed. It was found that for all three algorithms, possibilities exist to utilise the theory for this purpose. In fact, previous research on this topic can be found for each method. All three show great promise for analysing reinforced concrete.

For the purpose of this research, it was chosen to continue with the GSO-method. An excellent Python script, developed by He, Gilbert and Song [32], is available that implements this method. This script is freely available for further development, and good documentation is provided that explains the theory behind it. This choice was strengthened by the relatively low occurrence of this algorithm in the field of topology optimisation research.

7.2 Part 2: Adapting the optimisation to the manufacturing process

Now a lot of knowledge is gathered on both additive manufacturing techniques and topology optimisation, this part of the research aims to combine this knowledge. Several modifications and extensions to the previously mentioned Python script are proposed, that take into account the process constraints of the WAAM technique. The research question for this part is:

How can the optimisation algorithm be extended for additive manufacturing?

Three different suggestions were made to deal with the manufacturing constraints. The first suggestion concerns with forming a reduced set of possible members. The *member adding* technique that is used in the Python script is based on a list of all possible members that can be present in the solution space. The original script allowed all elements that connect two individual nodes in the domain, given that the member does not overlap with an existing member. A new function was proposed as a substitute for the original loop that creates the *Possible Member List*(PML). This function takes as input the design domain, a minimum printing angle and a maximum initial length. Due to reasons described in the appropriate section, the generation of the PML and the forming of an initial structure have to be combined when the minimum angle exceeds 45°. Therefore, also a maximum initial member length has to be provided to the function. Using the new function, it was found that an optimisation could be performed on a domain that is subjected to a minimum angle constraint. A comparison between a conventional analysis and an analysis subjected to a minimum angle of 25° can be seen in Figure 7.1.

As described in Appendix C, including this angle constraint can lead to large volume increases in specific examples. Especially when the force is perpendicular to the rigid foundation, the final efficiency of the model can be considered very poor. In simple cases, a manual adjustment of printing orientation is an evident solution. However, when the complexity of the model is higher, a good printing orientation may not always be easy to differentiate. Therefore, a method was proposed to automatically find a good printing orientation. This method works for both 2D

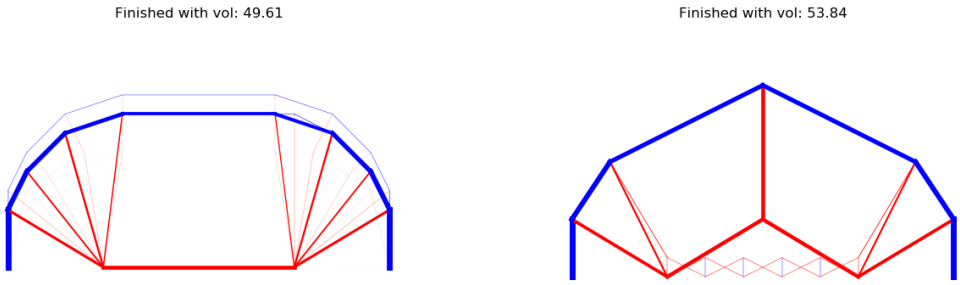


Figure 7.1: Conventional result(left) compared to an optimisation with an angle constraint of 25°(right)

and 3D models, and can be seen as a post-processing step. First, an optimisation is performed without any angle constraints. Then, the model is rotated around the origin in user-specified angular increments. At each rotation, a *Performance Index*(PI) is calculated. This PI compares the volume of printable material to the total volume of material. When the analysis is finished, a report is generated. An example of such a report for a 3D example is given in Figure 7.2.

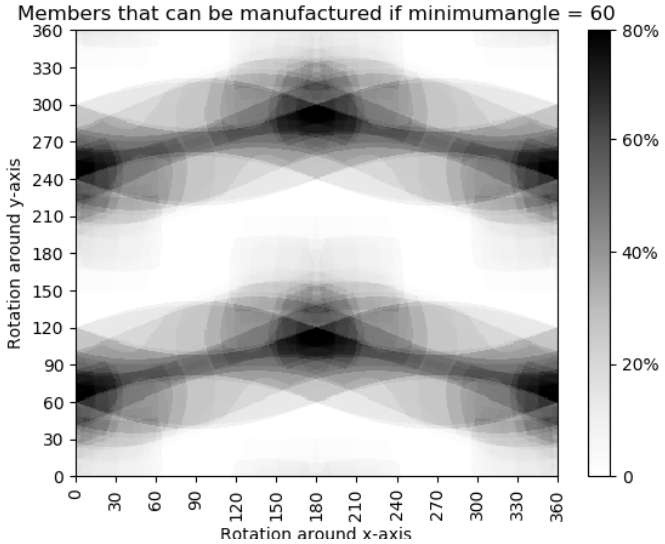


Figure 7.2: Rotation analysis for a model in 3D

The third and final aspect that has been investigated, concerns with the minimum member diameter. Following the findings from the literature review, a minimum diameter of 1.0 cm should always be ensured. An in-depth review of the functionality of the python script was performed, to investigate the possibility to include this constraint in the optimisation. However, it was found that rigorous changes to the script were necessary to achieve this. The *member adding* technique that is used never deactivates members, but only reduces their area to 0. This means a minimum area can not straightforwardly be imposed, because then every member in the domain would always be present. Therefore, a post-processing method is proposed. This analyses the result from the optimisation, and adjusts the areas where necessary.

7.3 Part 3: Validation

The research question that regards the validation of this research is:

- How efficient are the results of the extended optimisation method compared to the traditional calculation methods?

In this third part of the research, a case study was presented to illustrate the functionality of the proposed methods. This case study consists of a pile cap that was designed for a hotel in Amsterdam, which was considered to be a good example to showcase the new methods. It was found that, using the automated process, a reliable strut and tie model could be generated. In the example from this study, 30% less material was required to accommodate the forces compared to the traditional analysis method. It was also argued that the proposed method can yield large savings in computation time.

A large challenge for the proposed method can be distinguished when the minimum inclination is considered. It was found that the optimised strut and tie model could not be printed in any orientation, if a minimum inclination of 60° is required. The highest *Performance Index* of the analysis was 49.5%, which is regarded unacceptable. This issue raises questions regarding the feasibility of the method. However, developments in additive manufacturing techniques may offer a solution, which allows printing at steeper angles. Rotating printing surfaces, that can be adjusted over multiple axes, could potentially allow designs without any angle constraint.

7.4 Answering the main research question

The main research question that was proposed in the introduction is:

- Can reliable strut and tie models be generated by an extended topology optimisation scheme, which includes additive manufacturing constraints?

As proven in the case study in Chapter 5, a reliable way to include the manufacturing constraints of a 3D-printing environment in the calculation process of reinforced concrete elements was developed. The minimum member inclination can be included, although significantly increasing material use. For the minimum member diameter, a post-processing script was proposed. The topology optimisation method proved to be capable of finding an efficient force distribution, and thus an efficient strut and tie model. The minimum member diameter could straightforwardly be applied, when the amount of members in the domain was suppressed slightly.

The minimum member inclination proved to be the biggest hurdle in obtaining a printable design. A minimum inclination of 60° was found to be a very harsh constraint, significantly limiting the form freedom of the model. Even when this constraint was relieved, and the post-processing method for finding a suitable printing orientation was used, no printable design could be identified. For the feasibility of this method, it is therefore of importance that this minimum inclination is reduced. Developments in the field of additive manufacturing show promise for achieving this, for instance possibilities to print on a rotating surface.

Chapter 8

Recommendations

This chapter contains the recommendations based on the results that were found in this research. These consist of suggested future research, as well as more general ideas on how to further develop the proposed method.

1. This research has explored possibilities to extend a Python implementation of the ground structure method with additive manufacturing constraints. For the limitation on member inclination, two separate options were considered; demanding a minimum angle during the optimisation, and a post-processing tool that suggests a suitable printing orientation. Although these two methods were explored separately in this research, it would be of significant interest to look for ways to combine the concepts. An optimisation loop that is capable of both adjusting model orientation while maintaining a minimum member inclination from the horizontal is expected to be a very efficient and elegant next step.

2. Although not discussed in this research, it is also necessary to evaluate the concrete struts to obtain a valid strut-and-tie model. A logical next step in the development of the method would include a closer look into these compressive forces. An extension to the method is necessary to automatically validate the stresses in these elements.

3. This research has focused on the possibility to print the steel reinforcement that is necessary to accommodate the tensile forces in concrete elements. Advances in additive manufacturing techniques, including but not limited to the printing of steel, have made significant steps forward in the past decade. Several examples exist in literature where concrete was printed successfully. An inclusive method, that combines optimised, additive manufactured elements in both steel and concrete is within reach. Constructing a truss-like structure from both printed steel and concrete could in theory lead to large material savings.

4. Traditional strut-and-tie modelling requires designers to add reinforcement to accommodate the splitting forces that appear in the concrete struts. In the case study from Chapter 5, the reinforcement that results from this calculation was omitted to allow for a fair comparison. However, in theory it is possible to include this calculation in the Python script. The first method that comes to mind is to evaluate all compression struts after the optimisation is finished. However, adding the material required for the reinforcement after the optimisation will raise questions regarding the optimality of the result. Another, slightly more elaborate, path could be to calculate the required splitting reinforcement in each iteration of the optimisation loop.

5. Another aspect that was deliberately omitted from the comparison in the case study, was the calculation for crack width control. This procedure is part of the serviceability limit

state(SLS) calculation. In the traditional evaluation procedure, this part of the calculation consumes a large part of the total time required. In light of the more general goal to obtain an automated and performance driven design procedure, it is therefore of interest to look into methods to automate this calculation also. Due to the object oriented approach that is used in the Python programming language, it is possible to create a special library of functions for this. This library can then easily be included in other Python scripts, for instance the one used in this research.

6. The Python implementation of the ground structure method, as used extensively throughout this research, provides a relatively simple way to evaluate small design domains. As proven in Appendix A, increasing the amount of nodes exponentially increases the calculation time. The mathematical basis of the script, the so-called *member adding scheme*, proved to be capable of generating very complex and effective force paths from load to supports. However, generating the *Possible Member List*(PML) is a tedious job that significantly slows the process. A method to increase the speed of this procedure is desired. A possible solution could be to replace the loop structures in the function with matrix-vector multiplications. Known as vectorisation in programming, this procedure has the possibility to greatly improve calculation speeds.

Bibliography

[1] Ali Hasanbeigi, Marlene Arens, Jose Carlos Rojas Cardenas, Lynn Price, and Ryan Triolo. Comparison of carbon dioxide emissions intensity of steel production in China, Germany, Mexico, and the United States. *Resources, Conservation and Recycling*, 2016.

[2] Joerg Schlaich, Kurt Schaefer, and Mattias Jennewein. TOWARD A CONSISTENT DESIGN OF STRUCTURAL CONCRETE. *PCI Journal*, 32(3):74–150, 1987.

[3] Qing Quan Liang and Yi Min Xie. Topology Optimization of Strut-and-Tie Models in Non-Flexural Reinforced Concrete Members. 1999.

[4] Praveen Nagarajan and T M Madhavan Pillai. Development of strut and tie models for simply supported deep beams using topology optimization. Technical Report 5.

[5] G. I.N. Rozvany. A critical review of established methods of structural topology optimization. *Structural and Multidisciplinary Optimization*, 37(3):217–237, 1 2009.

[6] Ole Sigmund and Kurt Maute. Topology optimization approaches: A comparative review, 12 2013.

[7] A. H. Snijder, L. P.L. van der Linden, C. Goulas, C. Louter, and R. Nijsse. The glass swing: a vector active structure made of glass struts and 3D-printed steel nodes. *Glass Structures and Engineering*, 5(1):99–116, 3 2020.

[8] Flaviana Calignano, Diego Manfredi, Elisa Paola Ambrosio, Sara Biamino, Mariangela Lombardi, Eleonora Atzeni, Alessandro Salmi, Paolo Minetola, Luca Iuliano, and Paolo Fino. Overview on additive manufacturing technologies. *Proceedings of the IEEE*, 105(4), 2017.

[9] Baker. The use of an electric arc as a heat source to generate 3D objects depositing molten metal in superimposed layers, 1926.

[10] Anton Wiberg. *Towards Design Automation for Additive Manufacturing A Multidisciplinary Optimization approach Anton Wiberg FACULTY OF SCIENCE AND ENGINEERING*. 2019.

[11] Tiago A. Rodrigues, V. Duarte, R. M. Miranda, Telmo G. Santos, and J. P. Oliveira. Current status and perspectives on wire and arc additive manufacturing (WAAM). *Materials*, 12(7), 2019.

[12] A G M Michell M.C.E. LVIII. The limits of economy of material in frame-structures. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 8(47):589–597, 1904.

[13] G.A. Hegemier and W. Prager. On Michell trusses. *International Journal of Mechanical Sciences*, 11(2):209–215, 2 1969.

[14] William Prager and R T Shield. A General Theory of Optimal Plastic Design. *Journal of Applied Mechanics*, 34(1):184–186, 12 1967.

[15] Helen E. Fairclough, Matthew Gilbert, Aleksey V. Pichugin, Andy Tyas, and Ian Firth. Theoretically optimal forms for very long-span bridges under gravity loading. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 474(2217), 9 2018.

[16] Yi Xia, Matthijs Langelaar, and Max A N Hendriks. A critical evaluation of topology optimization results for Strut-and-Tie modelling of reinforced concrete.

[17] C. Buchanan and L. Gardner. Metal 3D printing in construction: A review of methods, research, applications, opportunities and challenges, 2 2019.

[18] Zhen Hu and Sankaran Mahadevan. Uncertainty quantification in prediction of material properties during additive manufacturing. *Scripta Materialia*, 135:135–140, 7 2017.

[19] Shawn M Kelly. Volume 1: Development and Measurement Analysis of Design Data for Laser Powder Bed Fusion Additive Manufacturing of Nickel Alloy 625 Final Technical Report Deliverable under Cooperative Agreement No. 70NANB12H264 Volume 1: Development of Mechanical Property Data for Laser Powder Bed Fusion Additive Manufacturing of Nickel Alloy 625 Deliverable under Cooperative Agreement No. 70NANB12H264. Technical report, 2014.

[20] Valmik Bhavar, Prakash Kattire, Vinaykumar Patil, Shreyans Khot, Kiran Gujar, and Rajkumar Singh. A review on powder bed fusion technology of metal additive manufacturing. Technical report, 2014.

[21] Mohammad Masoomi, Scott M. Thompson, and Nima Shamsaei. Laser powder bed fusion of Ti-6Al-4V parts: Thermal modeling and mechanical implications. *International Journal of Machine Tools and Manufacture*, 118-119:73–90, 8 2017.

[22] Markus Köhler, Sierk Fiebig, Jonas Hensel, and Klaus Dilger. Wire and arc additive manufacturing of aluminum components. *Metals*, 9(5), 5 2019.

[23] Bintao Wu, Zengxi Pan, Donghong Ding, Dominic Cuiuri, Huijun Li, Jing Xu, and John Norrish. A review of the wire arc additive manufacturing of metals: properties, defects and quality improvement, 10 2018.

[24] Viktor Mechtcherine, Jasmin Grafe, Venkatesh N. Nerella, Erik Spaniol, Martin Hertel, and Uwe Füssel. 3D-printed steel reinforcement for digital concrete construction – Manufacture, mechanical properties and bond behaviour. *Construction and Building Materials*, 179:125–137, 8 2018.

[25] Martin Philip Bends˜e and Noboru Kikuchi. GENERATING OPTIMAL TOPOLOGIES IN STRUCTURAL DESIGN USING A HOMOGENIZATION METHOD. Technical report, 1988.

[26] M P Bendsoe. Structural Optimization Optimal shape design as a material distribution problem. Technical report, 1989.

[27] Grant Steven, Osvaldo Querin, and Mike Xie. Evolutionary structural optimisation (ESO) for combined topology and size optimisation of discrete structures. Technical report.

[28] Y M Xie and G P Steven. A SIMPLE EVOLUTIONARY PROCEDURE FOR STRUCTURAL OPTIMIZATION. Technical Report 5, 1993.

[29] O. M. Querin, G. P. Steven, and Y. M. Xie. Evolutionary structural optimisation (ESO) using a bidirectional algorithm. *Engineering Computations (Swansea, Wales)*, 15(8):1031–1048, 1998.

[30] M P ROSSOW and J E TAYLOR. A Finite Element Method for the Optimal Design of Variable Thickness Sheets. *AIAA Journal*, 11(11):1566–1569, 1973.

[31] O Sigmund. A 99 line topology optimization code written in Matlab. Technical report, 2001.

[32] Linwei He, Matthew Gilbert, and Xingyi Song. A Python script for adaptive layout optimization of trusses. *Structural and Multidisciplinary Optimization*, 60(2):835–847, 8 2019.

[33] Zhi Hao Zuo and Yi Min Xie. A simple and compact Python code for complex 3D topology optimization. *Advances in Engineering Software*, 85:1–11, 2015.

[34] Karol Bołbotowski, Michał Knauff, and Tomasz Sokół. Zastosowanie optymalizacji topologicznej w projektowaniu konstrukcji żelbetowych z wykorzystaniem modeli" Strut and Tie". *Budownictwo i Architektura*, 12(1), 2013.

[35] K. Bołbotowski and T. Sokół. New method of generating Strut and Tie models using truss topology optimization. In *Advances in Mechanics: Theoretical, Computational and Interdisciplinary Issues - 3rd Polish Congress of Mechanics, PCM 2015 and 21st International Conference on Computer Methods in Mechanics, CMM 2015*, 2016.

[36] Steven Diamond and Stephen Boyd. CVXPY: A Python-Embedded Modeling Language for Convex Optimization. Technical report, 2016.

[37] Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 1 2018.

[38] Yi Xia, Matthijs Langelaar, and Max A N Hendriks. Optimization-Based Three-Dimensional Strut-and-Tie Model Generation for Reinforced Concrete. Technical report.

[39] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Appendix A

Parameter Study

This Appendix is meant to illustrate the influence of the different input parameters for both the matlab implementation of the SIMP method, as the python script that uses the Ground Structure Method. In all tests, a computation time is included. These were run on a laptop computer with an i7-8550K at 1.8GHz with an overboost to 4.0GHz.

A.1 SIMP

In the first secion, the SIMP method is discussed. There are three different parameters that can be used to influence the result of the optimisation. These are the volume fraction, the penalty factor and the filter size. In the subsections below, for each of these parameters three different values are used. The outcomes are presented in table form, but also visual representations are given.

A.1.1 Varying volume fraction

Varying the volume fraction leads to results that can be seen in Table A.1 and Figure A.1. It can be observed that varying the volume fraction has no influence of the resulting shape of the structure. In both cases the same truss like structure can be identified, only the elements have have different thicknesses. Run *vf1*, with the lowest volume fraction, took significantly longer than the other runs. During the process, it could be seen that the solution had difficulty converging. A large part of the run saw flickering of elements.

Parameter	Run *vf1*	Run *vf2*	Run *vf3*
Elements X	40	40	40
Elements Y	20	20	20
Volume fraction	0.15	0.3	0.5
Penalty factor	3.0	3.0	3.0
Filter Size	1.2	1.2	1.2
Results			
Iterations 603	59	49	
Volume fraction	0.15	0.3	0.5
Time 21.0s	2.7s	1.7s	

Table A.1: Input parameters and result for three different volume fractions

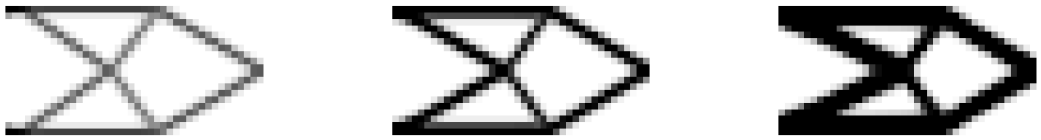


Figure A.1: SIMP result of three different volume fractions; 0.15(left), 0.3(middle), 0.5(right)

A.1.2 Varying penalty factor

The consequences of varying the penalty factor can be seen in Table A.2 and Figure A.2. Setting the penalty factor to 1.0 is equivalent to running the algorithm without any penalty factor. For explanation refer to section 3.3.1. When the penalty factor is not used, the result shows a lot of grey elements. These are elements with a density between 0 and 1. Setting the penalty factor to 2.0 in *pf2*, the solution is already much more discrete, but some grey areas can still be distinguished at the joints. Run *pf3* shows almost only black or white elements. When setting the penalty factor higher than 3.0, the script would not converge.

Parameter	Run *pf1*	Run *pf2*	Run *pf3*
Elements X	40	40	40
Elements Y	20	20	20
Volume fraction	0.3	0.3	0.3
Penalty factor	1.0	2.0	3.0
Filter Size	1.2	1.2	1.2
Results			
Iterations	12	64	59
Penalty factor	1	2	3
Time	0.4s	2.3s	2.7s

Table A.2: Input parameters and result for three different penalty factors

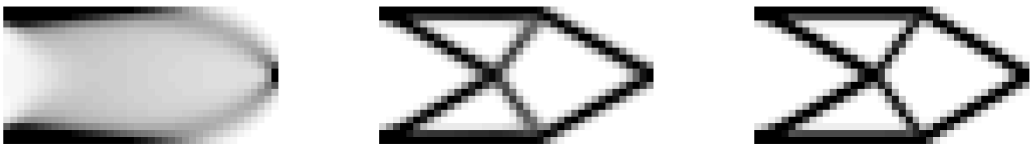


Figure A.2: SIMP result of three different penalty factors; 1.0(left), 2.0(middle), 3.0(right)

A.1.3 Varying filter size

The element filter that is present in the matlab script is made to prevent a phenomenon called checkerboarding. Explanation is given in section 3.3.2. The filter 'spreads out' forces over

neighbouring elements. As can be seen in the result in Table A.3 and Figure A.3, again there is no influence on the outline of the structure. The basic layout of the elements is the same. Increasing the filter size appears to smear out the thickness of the members, forcing them to be grey.

Parameter	Run *fs1*	Run *fs2*	Run *fs3*
Elements X	40	40	40
Elements Y	20	20	20
Volume fraction	0.3	0.3	0.3
Penalty factor	3.0	3.0	3.0
Filter Size	0	1.2	1.2
Results			
Iterations	42	64	97
Filter size	1.1	1.2	3.0
Time	1.6	2.3s	3.4s

Table A.3: Input parameters and result for three different filter sizes

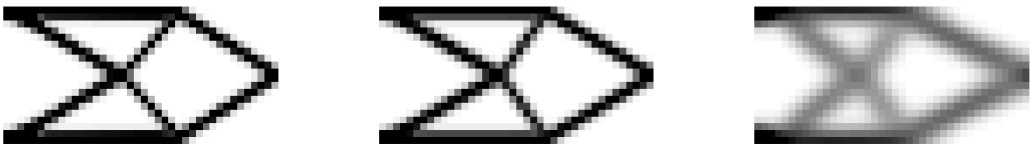


Figure A.3: SIMP result of three different filter sizes; 1.0(left), 2.0(middle), 3.0(right)

A.1.4 Model size

Now the influence of a bigger structure is investigated. The aspect ratio of 2:1 is preserved, but more elements are added in x and y direction. The results are in Table A.4 and Figure A.4. Again, no difference in shape is observed. Adding more elements simply sharpens the image. The computation time increases significantly with more elements. Run *ms1* had difficulty converging, hence the high number of iterations.

Parameter	Run *ms1*	Run *ms2*	Run *ms3*
Elements X	20	40	120
Elements Y	10	20	60
Volume fraction	0.3	0.3	0.3
Penalty factor	3.0	3.0	3.0
Filter Size	1.2	1.2	1.2
Results			
Iterations	113	59	141
Time	2.8s	2.3s	720s

Table A.4: Input parameters and result for three different model sizes

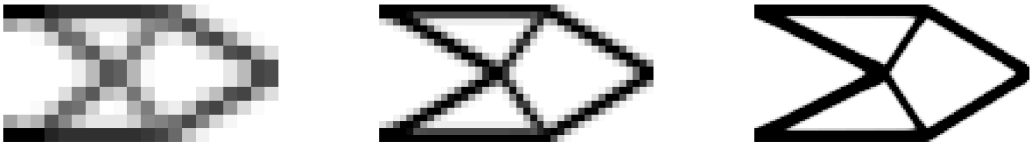


Figure A.4: SIMP result of three different model sizes; 20x10 (left), 40x20 (middle), 80x40 (right)

A.2 GSO

A.2.1 Difference in compression and tensile strength

To see how a difference in tensile and compression strength influences the structure, three runs are performed. These runs use the same structure, but vary in ratio between the strengths. Ratios of 1:1, 2:1 and 4:1 are used. The results can be found in Table A.5 and Figure A.5. The location of the members does not appear to have changed. The only difference is that the elements with higher strength, in this case the tension elements, have a reduced thickness. The total volume of the structure reduces with increasing tensile strength.

Parameter	Run *gso-sd1*	Run *gso-sd2*	Run *gso-sd3*
Elements X	40	40	40
Elements Y	20	20	20
σ_c	1.0	1.0	1.0
σ_t	1.0	2.0	4.0
Joint Cost	0	0	0
Results			
Iterations	10	11	11
Volume	140.8	105.1	85.6
Time	36s	73s	70s

Table A.5: Input parameters and result for three different ratios between tension and compression strength

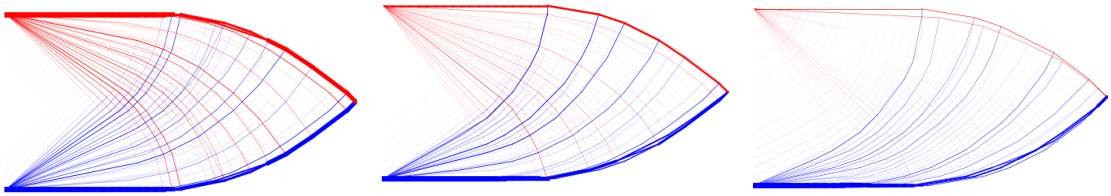


Figure A.5: GSO result for three different tensile/compressive ratios; 1:1 (left) 2:1 (middle) and 4:1 (right)

A.2.2 Joint Cost

The influence of a higher joint cost is clearly visible in the results in Table A.6 and Figure A.6. A joint cost of 0.1 already shows a significant reduction of the amount of members. With

increasing joint cost, we also observe an increase in total volume of the structure. In terms of material use, a structure with less members is therefore less efficient. The computation time is increased as well, but because this is a relatively small structure it can still be done in a couple of seconds.

Parameter	Run *gso-jc1*	Run *gso-jc2*	Run *gso-jc3*
Elements X	20	20	20
Elements Y	10	10	10
Results			
Iterations	13	15	64
Volume	70.8	72.1	79.3
Time	5.9s	5.9	7.1s

Table A.6: Input parameters and result for different joint cost

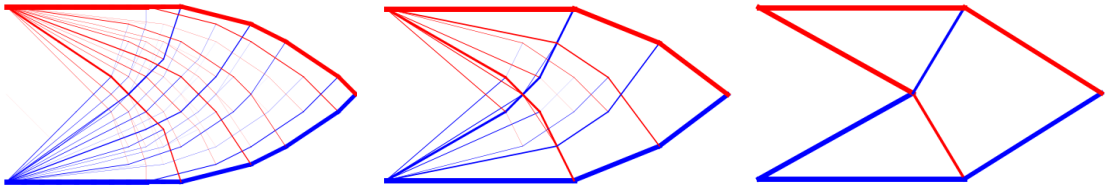


Figure A.6: GSO result of different joint cost, respectively 0 (left), 0.1 (middle) and 1.0 (right)

A.2.3 Model Size

The model size is now increased gradually, by setting the number of x- and y-elements. To illustrate the effect this has on the possible solutions, a row is added to Table A.7 with the amount of possible members. This number grows exponentially with increasing model size. Due to this increasing size of the calculation, the computation time shows an enormous increase as well. For Run *gso-ms3* it took as much as three hours to get a result. Looking at this result in Figure A.7, it shows that it is not a good solution. Large quantities of very small members can be seen, causing something that looks like a blurred image. The first test run, which is pictured on the left, already shows a very detailed design. This design only uses a 20x10 grid, but apparently this is enough to generate a good result. Therefore, it is the question whether it is ever really necessary to increase the model size as much as was done in *gso-ms3*. It should also be noted that this script uses intermediate plotting. The result is shown after each iteration. This plotting process also requires computation power, and increases the calculation time. Run *gso-ms2* was also performed without this plotting, which caused a reduction in time of about 8 seconds (25%).

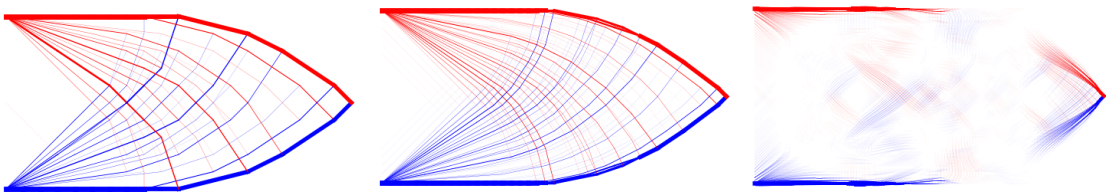


Figure A.7: GSO result of three different model sizes; 20x10 (left), 40x20 (middle), 120x60 (right)

Parameter	Run *gso-ms1*	Run *gso-ms2*	Run *gso-ms3*
Elements X	20	40	120
Elements Y	10	20	60
σ_c	1.0	1.0	1.0
σ_t	1.0	1.0	1.0
Joint Cost	0	0.1	1.0
Possible Members	19900	319600	25916400
Results			
Iterations	13	10	12
Volume	70.8	140.8	241.9
Time	4.9s	32s	3h 13min

Table A.7: Input parameters and result for three different model sizes

Appendix B

the 'Member Adding' scheme

The python script that is used in various forms throughout this research is made by Matthew Gilbert and Linwei He. The raw code can be found in section E.1 on Page 95. In this section the functionality of the script will be explained. This is done in four steps; Input definition, Solving, Plotting and Termination.

B.1 Input definition

The first step in the *trussopt* function is defining the inputs. The code in which this is done can be found on line 71-90. By default, the script is set up to receive a width and height as input parameters. These parameters are used to construct a polygon. This polygon can be modified to contain holes and passive areas if desired. At the same time, the numpy *meshgrid* function is used to create a grid of evenly spaced nodes inside the design space. The loads and support conditions are added to lists. The next step is generating a possible member list named *PML*. This list contains every possible connection within the design space. To loop over all possible combinations, the *combinations* function of the *itertools* package is used. To avoid overlapping elements, the *greatest common divisor* is used. This is illustrated in Figure B.1. *PML* is the main data container of the script. Each row in this list represents a separate member. Each member has four attributes; a startpoint, an endpoint, a length and a Boolean value representing whether the member is active in the current solution. Before the solving iterations commence, an initial set of member is activated. As pointed out by He and Gilbert [32], this method shows less memory consumption and therefore lower computation time than directly solving for the complete set. As an initial set, only the horizontals, verticals, and element diagonals are activated in the default configuration. This is done by checking which elements have a length smaller than or equal to $\sqrt{2}$. This initial set can be seen in Figure B.2.

B.2 Solving

The main optimization loop can be found on lines 97-102. From within this loop, different functions are called to perform the calculations. At the start of each iteration, a sub list with all active members is selected and stored in *Cn*. Together with the input parameters this list is passed to the *SolveLP* function(lines 19-34). This function performs two operations. First, it assembles the equilibrium matrix that is necessary to calculate the internal forces(lines 10-17). Then, it defines and solves an optimisation problem for Python library *CVXPY*. The reader

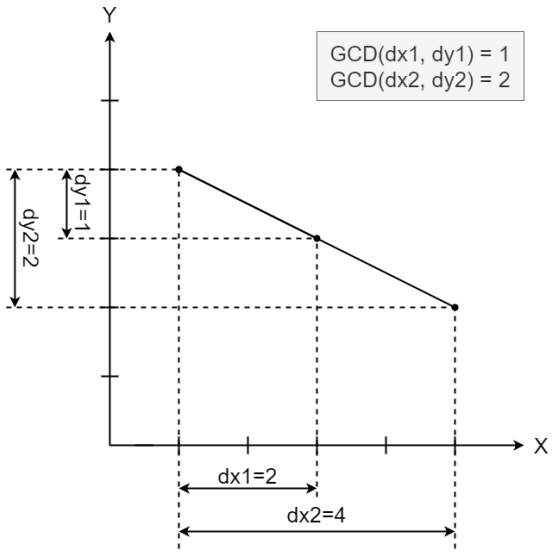


Figure B.1: *Greatest common divisor* function is used to eliminate overlapping elements

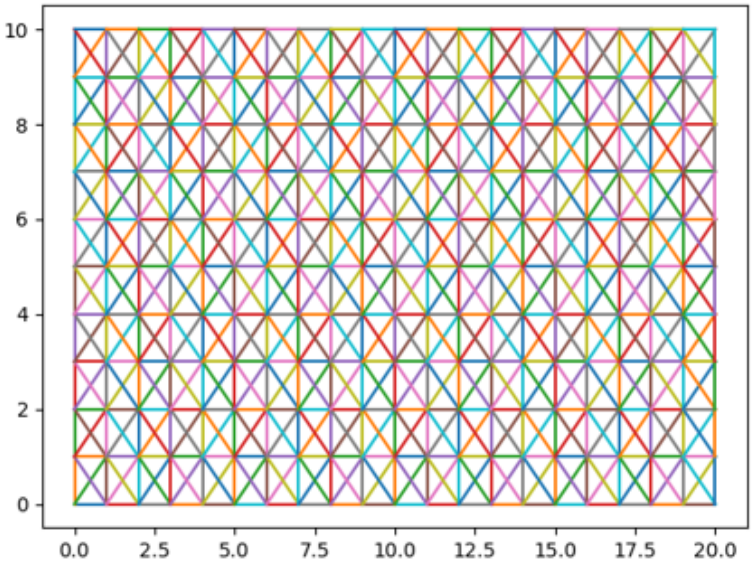


Figure B.2: Initial set of activated members for a rectangular design space of 20x10 elements

is referred to Diamond[36] and Agrawal[37] for detailed information on this library. *CVXPY* is given a set of variables and a list of constraints. In our case, this list of variables are the cross sectional areas of all members. The constraints are a limiting tensile stress and a limiting compressive stress. By evaluating the structure with the equilibrium matrix and adjusting the areas, an optimum material distribution is obtained. A joint cost is also implemented by this function, which is added to the length of the members in line 20. This penalises short members, and forces the solution to contain less joints.

B.3 Plotting

The plot function *plotTruss* can be found on lines 52-69. This function plots all members in *Cn*, which contained the active members in the solution space. There is an additional requirement on the cross sectional area of the element. If this area is smaller than 0.1% of the maximum area, the element is not plotted. All other members are plotted as line elements. The thickness of each element is scaled towards the maximum value of the list of areas. A blue color is assigned to elements under compression, and red is for elements under tension. Elements that switch between compression and tension in different load cases are plotted as grey lines.

B.4 Termination

Now the list of areas is optimised for the applied forcing, it needs to be checked whether this intermediate solution is the final solution. This is done in the *stopViolation* function on lines 36-50. This function uses Lagrange-functions to evaluate the virtual strain in the complete structure. In-depth discussion of this subject is beyond the scope of this research, but readers are referred to Boyd [39] for extensive information on this subject. Checking the virtual strain is done for all members in *PML*, which includes members that are not active in the current structure. A list of all these virtual strains is assembled and sorted by descending magnitude. To control the 'growth rate' of the structure, the amount of members that will be added to the set is determined with:

$$\Delta m = \left\{ \begin{array}{ll} \alpha m_V, & m_V \geq \beta m_p \\ \alpha\beta m_p, & \beta m_p > m_V > \alpha\beta m_p \\ m_V, & \alpha\beta m_p \geq m_V \end{array} \right\} \tag{B.1}$$

In this script both α and β are set to 0.05. In words, this step ensures the smoothness of the process. If a lot of members violate the maximum virtual strain, only a few will be added to the solution. If only a few members violate the strain, most will be added. We have now arrived at the end of one iteration. If there are violations, the members are added to the set and the next iteration starts at line 98. This process is repeated until no members violate the maximum virtual strain(0.01%).

B.5 Notes

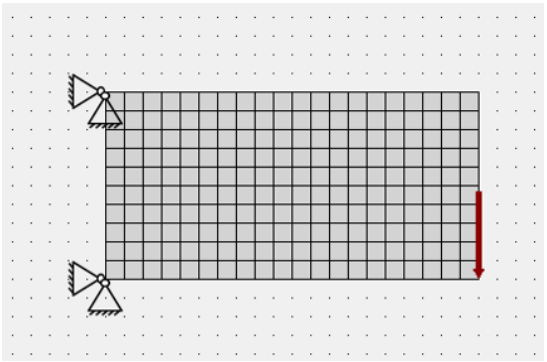
Considering the process described above, several side notes can be made. First we consider the input. No physical meaning is specified for the input parameters. No

B.6 Code

Appendix C

Reduced performance due to angle constraints

When the angle constraint is added to the optimisation loop, there is a risk of greatly reducing the overall performance of the structure. In this section, this risk is described by presenting an example. An obvious solution will be presented and discussed. The structure that will be used in this section is pictured in Figure C.1. A unit force is applied in the bottom right corner, the bottom and top left corner are supported in both directions. Width is set to 20 and height to 10. The limiting tensile stress is set equal to the limiting compression stress as 1. No joint cost is specified. The maximum initial member length is taken as $\sqrt{2}$.



Property	Value
Width	20
Height	10
σ_t	1
σ_c	1
Joint Cost	0
Minimum Angle	[0, 25]
Minimum Initial Length	$\sqrt{2}$

Figure C.1: Simple cantilever example

The optimisation is now run twice. Once without any specified minimum angle, one with a minimum angle of 25°. The results can be seen in Figure C.2 and Figure C.3. Since we minimise volume, this is the obvious choice to assess the performance of the structure. An increase in volume of 36% is observed when the angle constraint is added. This is significant, so a solution is required.

C.1 Manual Approach

In this relatively simple case, the solution is obvious. If we rotate the structure from Figure C.2 90^o counter-clockwise, the performance is expected to be significantly higher. We change the boundary conditions and loading to the situation pictured in Figure C.1.

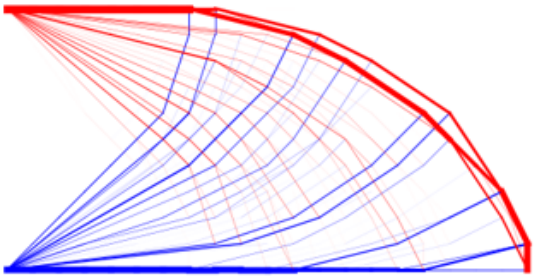


Figure C.2: Result without angle constraint, total volume: **73.59**

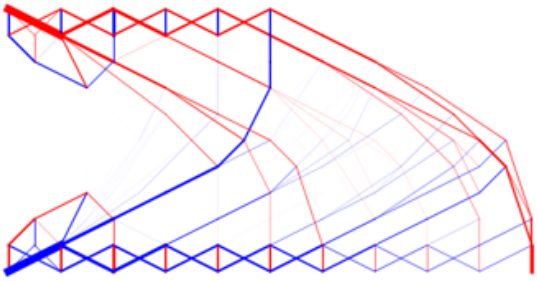


Figure C.3: Result with minimum angle of 25^o, total volume: **99.98**

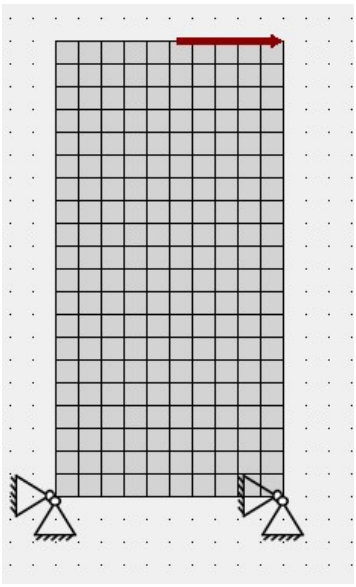


Figure C.4: Rotated cantilever example

Property	Value
Width	10
Height	20
σ_t	1
σ_c	1
Joint Cost	0
Minimum Angle	[0, 25]
Minimum Initial Length	$\sqrt{2}$

Again we run the optimisation twice. The results without angle constraint and with angle constraint can be found in Figure C.5 and Figure C.6. The increase in volume is now limited to 0.3%.

This may be an extreme scenario, but the issue is clear. Using the angle constraint function from *trussopttools* may not be the best choice in some cases. The python script has no functionality to determine the best performing orientation for the structure. In the case of this example, the solution is obvious. However, with increasing complexity of the structure, it may not always be directly clear what the best orientation will be. Section 4.2 presents an automated procedure for finding a good printing orientation.

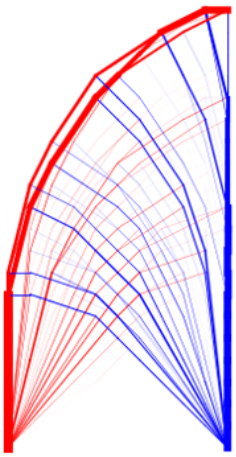


Figure C.5: Result without angle constraint, total volume: **73.59**

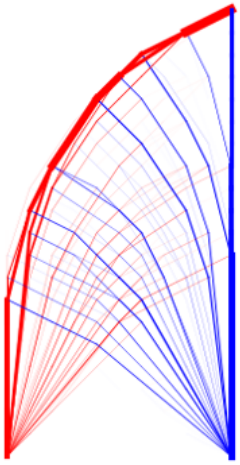


Figure C.6: Result with minimum angle of 25^o, total volume: **73.81**

Appendix D

Rotating an Array

The structure is stored in memory as a collection of lines. These lines are each described by a start point and end point. To find a suitable printing orientation, we would like to rotate the entire structure in small increments. At each new increment the Performance Index is evaluated. For simplicity, it is chosen to rotate the entire structure around the origin. To illustrate this, a single member is rotated around the origin by angle β in Figure D.1. As input for our rotation

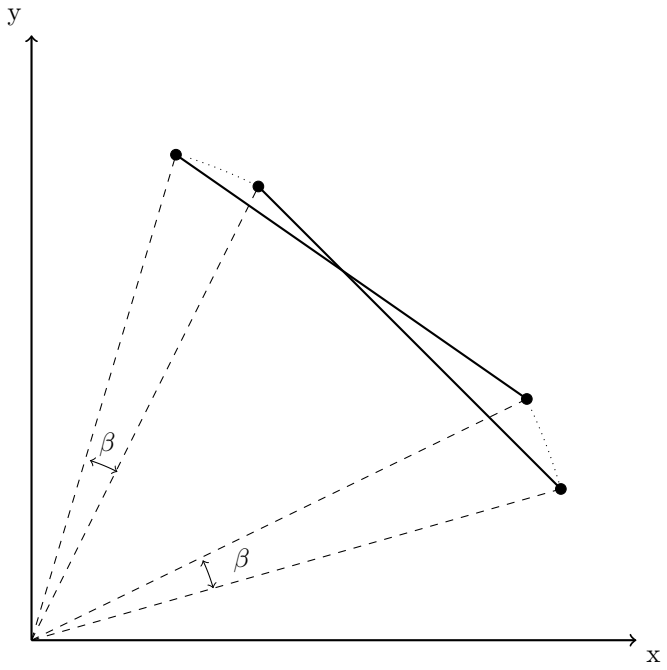


Figure D.1: Rotation of a single member around the origin

function, we expect two arrays of points. One describing x- and y-coordinates of the start points, the other describing the x- and y-coordinates of the end points of each member. For a structure containing n members, these arrays will be of the form:

$$p_{start} = \begin{bmatrix} x_{s,1} & y_{s,1} \\ x_{s,2} & y_{s,2} \\ \vdots & \vdots \\ x_{s,n} & y_{s,n} \end{bmatrix}, \quad p_{end} = \begin{bmatrix} x_{e,1} & y_{e,1} \\ x_{e,2} & y_{e,2} \\ \vdots & \vdots \\ x_{e,n} & y_{e,n} \end{bmatrix} \tag{D.1}$$

We are interested in rotating the complete structure around a certain point. To describe this rotation, we would like to express the location of all start points and end points in the same manner as before. This means we are looking for two new arrays $\bar{p}_{start}$ and $\bar{p}_{end}$.

$$\bar{p}_{start} = \begin{bmatrix} \bar{x}_{s,1} & \bar{y}_{s,1} \\ \bar{x}_{s,2} & \bar{y}_{s,2} \\ \vdots & \vdots \\ \bar{x}_{s,n} & \bar{y}_{s,n} \end{bmatrix}, \quad \bar{p}_{end} = \begin{bmatrix} \bar{x}_{e,1} & \bar{y}_{e,1} \\ \bar{x}_{e,2} & \bar{y}_{e,2} \\ \vdots & \vdots \\ \bar{x}_{e,n} & \bar{y}_{e,n} \end{bmatrix} \tag{D.2}$$

Since both arrays are of the same form and the rotation is the same, one rotation matrix will be sufficient. The coefficients of this rotation matrix are still unknown, so we will attempt to find these. To this end, we consider the rotation of a single point around the origin. This point has coordinates (x, y) before the transformation. We need to find the coordinates $(\bar{x}, \bar{y})$ after the rotation. Simple trigonometric relations can be used to find these. For this, refer to Figure D.2. In the figure, Point I with known coordinates (x,y) is rotated around the origin over β

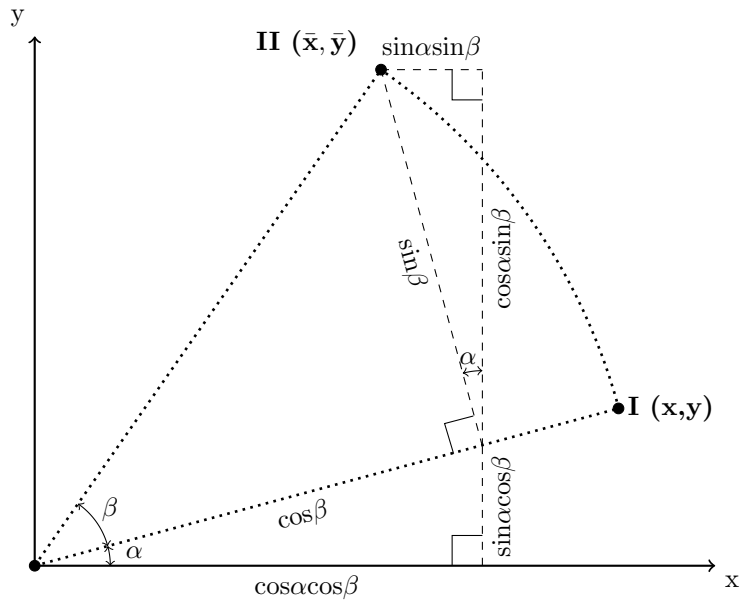


Figure D.2: Trigonometric identities

degrees. The distance between point I and the origin is given a unit value. Point II is the new location. The still unknown coordinates $(\bar{\text{x}}, \bar{\text{y}})$ need to be expressed in (x,y). By constructing three right-angled triangles, we find the following expressions for $\bar{\text{x}}$ and $\bar{\text{y}}$:

$$\begin{aligned} \bar{\text{x}} &= \text{cos}\alpha\text{cos}\beta - \text{sin}\alpha\text{sin}\beta \\ \bar{\text{y}} &= \text{cos}\alpha\text{sin}\beta + \text{sin}\alpha\text{cos}\beta \end{aligned} \tag{D.3}$$

We know the distance between Point I and the origin is 1. Therefore, we can express the coordinates x and y in angle α like so:

$$\begin{aligned} x &= \cos\alpha \\ y &= \sin\alpha \end{aligned} \tag{D.4}$$

Combining equation D.3 and D.4 yields the following relation:

$$\begin{aligned} \bar{\text{x}} &= \text{x}\cos\beta - \text{y}\sin\beta \\ \bar{\text{y}} &= \text{x}\sin\beta + \text{y}\cos\beta \end{aligned} \tag{D.5}$$

In matrix notation, this reduces to

$$\begin{bmatrix} \bar{\text{x}} \\ \bar{\text{y}} \end{bmatrix} = \begin{bmatrix} \cos\beta & -\sin\beta \\ \sin\beta & \cos\beta \end{bmatrix} \begin{bmatrix} \text{x} \\ \text{y} \end{bmatrix} \tag{D.6}$$

This can easily be extended to systems with more coordinate pairs, by lengthening the arrays. Linear algebra rules allow us to write the following:

$$\begin{bmatrix} \bar{\mathrm{x}}_1 & \bar{\mathrm{x}}_2 & \dots & \bar{\mathrm{x}}_n \\ \bar{\mathrm{y}}_1 & \bar{\mathrm{y}}_2 & \dots & \bar{\mathrm{y}}_n \end{bmatrix} = \begin{bmatrix} \cos\beta & -\sin\beta \\ \sin\beta & \cos\beta \end{bmatrix} \begin{bmatrix} \mathrm{x}_1 & \mathrm{x}_2 & \dots & \mathrm{x}_n \\ \mathrm{y}_1 & \mathrm{y}_2 & \dots & \mathrm{y}_n \end{bmatrix} \tag{D.7}$$

Recall the expected input from Equation D.1. Given a certain angle β, the system of members can be rotated around the origin in the following manner:

$$\begin{aligned} \bar{\mathrm{p}}_{\mathrm{start}}^{\mathrm{T}} &= \mathbf{R} \cdot \mathrm{p}_{\mathrm{start}}^{\mathrm{T}} \\ \bar{\mathrm{p}}_{\mathrm{end}}^{\mathrm{T}} &= \mathbf{R} \cdot \mathrm{p}_{\mathrm{end}}^{\mathrm{T}} \end{aligned} \tag{D.8}$$

In which

$$\mathbf{R} = \begin{bmatrix} \cos\beta & -\sin\beta \\ \sin\beta & \cos\beta \end{bmatrix} \tag{D.9}$$

Appendix E

Code

E.1 Member Adding

```
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
import time
#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))
#Solve linear programming problem
def solveLP(Nd, Cn, f, dof, st, sc, jc):
    l = [col[2] + jc for col in Cn]
    B = calcB(Nd, Cn, dof)
    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])
    prob = cvx.Problem(obj, cons)
    vol = prob.solve(max_iter=100000, verbose=False, eps_abs=1e-3)
    q = [np.array(qi.value).flatten() for qi in q]
    a = np.array(a.value).flatten()
    u = [-np.array(eqnk.dual_value).flatten() for eqnk in eqn]
    return vol, a, q, u
#Check dual violation
def stopViolation(Nd, PML, dof, st, sc, u, jc):
    lst = np.where(PML[:,3]==False)[0]
    Cn = PML[lst]
    l = Cn[:,2] + jc
    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):
        PML[lst[vioCn[vioSort[i]]]][3] = True
    return num == 0
#Visualize truss
def plotTruss(Nd, Cn, a, q, threshold, str, update = True):
    plt.ion() if update else plt.ioff()
    plt.clf(); plt.axis('off'); plt.axis('equal');  plt.draw()
    plt.title(str)
    tk = 5 / max(a)
    for i in [i for i in range(len(a)) if a[i] >= threshold]:
        if all([q[lc][i]>=0 for lc in range(len(q))]): c = 'r'
        elif all([q[lc][i]<=0 for lc in range(len(q))]): c = 'b'
        else: c = 'tab:gray'
        pos = Nd[Cn[i, [0, 1]].astype(int), :]
        #Preparing the output
        thickness = a[i] * tk
        color = 0
        if c=='r': color = 1;
        if c=='b': color = 2;
        #Plotting the truss
```

```
        plt.plot(pos[:, 0], pos[:, 1], c, linewidth = a[i] * tk)
    plt.pause(0.01) if update else plt.show()
#Main function
def trussopt(width, height, st, sc, jc):
    starttime=time.time()
    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
    for i, nd in enumerate(Nd):
        if nd[0] == 0 and nd[1]<=height: dof[i] = [0, 0]
        f+=[0,-1] if nd[0]==width and nd[1]==height/2 else [0,0]
    #Create the 'ground structure'
    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:
            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] )
    PML= np.array(PML)
    dof=np.array(dof).flatten()
    f = [f[i:i+len(Nd)*2] for i in range(0, len(f), len(Nd)*2)]
    print('Nodes: %d Members: %d' % (len(Nd), len(PML)))
    for pm in [p for p in PML if p[2] <= 1.42]:
        pm[3] = True
    #Start the 'member adding' loop
    for itr in range(1, 100):
        Cn = PML[PML[:,3] == True]
        vol, a, q, u = solveLP(Nd, Cn, f, dof, st, sc, jc)
        print("Itr: %d, vol: %f, mems: %d" % (itr, vol, len(Cn)))
        plotTruss(Nd, Cn, a, q, max(a) * 1e-3, "Itr:" + str(itr), True)
        if stopViolation(Nd, PML, dof, st, sc, u, jc): break
    print("Volume: %f" % (vol))
    print(f"Process completed in {time.time()-starttime} seconds")
    plotTruss(Nd, Cn, a, q, max(a) * 1e-3, "Finished", False)
#Execution function when called directly by Python
if __name__ =='__main__':
    trussopt(width = 20, height = 10, st = 1, sc =1, jc = 0)
################################################################################
# This Python script was written by L. He, M. Gilbert, X. Song                 #
# University of Sheffield, United Kingdom                                      #
# Please send comments to: linwei.he@sheffield.ac.uk                           #
# The script is intended for educational purposes - theoretical details        #
# are discussed in the following paper, which should be cited in any           #
# derivative works or technical papers which use the script:                   #
#                                                                              #
# "A Python script for adaptive layout optimization of trusses",               #
# L. He, M. Gilbert, X. Song, Struct. Multidisc. Optim., 2019                  #
#                                                                              #
# Disclaimer:                                                                  #
# The authors reserve all rights but do not guarantee that the script is       #
# free from errors. Furthermore, the authors are not liable for any            #
# issues caused by the use of the program.                                     #
################################################################################
```

E.2 findorientation

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import sys
import trussopttools as trto
import os
from pathlib import Path
import cvxpy as cvx
from tqdm import tqdm
from math import ceil
import warnings
warnings.filterwarnings('ignore')

'''
General Functions
'''
def ReportPerformance(filepath, minimumangle, showplots = True, stepsize = 5):
  modelpath = os.path.dirname(filepath)
  Result, is3D = GetResult(filepath)
  sp, ep, a, l, vol = GetModelParameters(Result, is3D, steelonly = True)
  if is3D:
    print(f"3D structure loaded")
    pi, best_rx, best_ry = Analyse3DPerformance(sp, ep, a, l, vol, minimumangle,
    showplots, stepsize)
    SaveUpdatedResult3D(sp, ep, a, best_rx, best_ry, modelpath)
    PlotRotatedTruss3D(sp, ep, a, best_rx, best_ry, pi, minimumangle, False)
  else:
    print(f"2D structure loaded")
    pi, best_rz = Analyse2DPerformance(sp, ep, a, l, vol, minimumangle, showplots
    , stepsize)
    SaveUpdatedResult2D(sp, ep, a, best_rz, modelpath)
    PlotRotatedTruss2D(sp, ep, a, best_rz, pi, minimumangle, False)

def GetResult(Result_location):
  Result = np.array(trto.readcsv(Result_location, astype='float'))
  is3D = True if len(Result[0]) == 8 else False
  return Result, is3D

def GetModelParameters(Result, is3D, steelonly = True):
  if steelonly:
    Result = [row for row in Result if row[-1]==1]
  Result = np.array(Result)
  if is3D:
    Startpoints = Result[:,[0,1,2]]
    Endpoints = Result[:,[3,4,5]]
    a = Result[:,6]
  else:
    Startpoints = Result[:,[0,1]]
    Endpoints = Result[:,[2,3]]
    a = Result[:,4]
  l = GetLengths(Startpoints, Endpoints)
  vol = sum(a*l)
  return Startpoints, Endpoints, a, l, vol

def RotateArray(array, alpha, axis):
  array = np.transpose(array)
  alpha = (np.pi/180)*alpha
  if axis == 'z':
    R = np.array([[np.cos(alpha), -np.sin(alpha)], [np.sin(alpha), np.cos(alpha)
    ]])
  if axis == 'x':
    R = np.array([[1, 0, 0],[0, np.cos(alpha), -np.sin(alpha)], [0, np.sin(alpha)
    , np.cos(alpha)]])
  if axis == 'y':
    R = np.array([[np.cos(alpha), 0, np.sin(alpha)], [0,1,0], [-np.sin(alpha), 0,
     np.cos(alpha)]])
  res = np.transpose(R.dot(array))
```

```
  return res

def CalculateAngles(startpoints, endpoints, plane):
  sp = startpoints
  ep = endpoints
  if plane == 'xy' or plane == 'yx':
    dxdy = np.transpose([abs(ep[:,0]-sp[:,0]), abs(ep[:,1] - sp[:,1])])
    deltas = dxdy
  if plane == 'xz' or plane == 'zx':
    dxdz = np.transpose([abs(ep[:,0]-sp[:,0]), abs(ep[:,2] - sp[:,2])])
    deltas = dxdz
  if plane == 'yz' or plane == 'zy':
    dydz = np.transpose([abs(ep[:,1]-sp[:,1]), abs(ep[:,2] - sp[:,2])])
    deltas = dydz
  angles = []
  for row in deltas:
    angles.append(np.arctan(row[1]/row[0])*180/np.pi)
  return angles

def GetLengths(startpoints, endpoints):
  lengths = []
  for i, sp in enumerate(startpoints):
    deltas = []
    for j, c in enumerate(sp):
      deltas.append((endpoints[i,j]-c)**2)
    lengths.append(np.sqrt(sum(deltas)))
  return lengths

def ReportUnprintableVolume(angles, a, l, minimumangle, is3D):
  UnprintableVolume = 0
  if is3D:
    for i, angle in enumerate(angles):
      for plane_angle in angle:
        if plane_angle < minimumangle:
          UnprintableVolume += a[i]*l[i]
          break
  else:
    for i in [i for i in range(len(angles)) if angles[i] < minimumangle]:
      UnprintableVolume += a[i]*l[i]
  return UnprintableVolume

'''
2D Functions
'''
def Analyse2DPerformance(sp, ep, a, l, vol, minimumangle, showplots, stepsize):
  rotations = range(0,360,stepsize)
  PI=[]
  for rotation in rotations:
    pi=PerformanceIndex2D(RotateArray(sp, rotation, 'z'), RotateArray(ep,
    rotation, 'z'), a, l, vol, minimumangle)
    PI.append(pi)
    # print(f"--- Angle: {rotation}"+u"\u00b0"+f" Performance Index: {pi:.2f} %
    ---")
    if showplots:
      PlotRotatedTruss2D(sp, ep, a, rotation, pi, minimumangle, True)
      if rotation in [80,110,130,160]:
        input("wait")
  PlotResult(rotations, PI, 80)
  res = np.transpose([rotations, PI])
  print(f'Best Performance Index found was {max(PI):.1f}%')
  best_indices = np.where(res[:,1]==max(PI))
  print("The following rotations have this Performance Index:")
  for i in range(len(best_indices[0])):
    index = best_indices[0][i]
    angle = res[index][0]
    print(f"{angle}"+u"\u00b0"+" Around the z-axis")
  rz = float(input("Type the rotation around the z-axis you would like to use: ")
    )
  return max(PI), rz
```

```
129
130 def PerformanceIndex2D(sp, ep, a, l, vol, minimumangle):
131   angles = CalculateAngles(sp, ep, 'xy')
132   uv = ReportUnprintableVolume(angles, a, l, minimumangle, False)
133   return ((vol-uv)/vol)*100
134
135 def PlotRotatedTruss2D(sp, ep, a, rotation, PI, minimumangle, update):
136   sp = RotateArray(sp, rotation, 'z')
137   ep = RotateArray(ep, rotation, 'z')
138   angles = CalculateAngles(sp, ep, 'xy')
139   plt.clf();
140   plt.axis('equal')
141   plt.tick_params(
142     axis='both',          # changes apply to the x-axis
143     labelbottom = False,
144     labelright = False,
145     labelleft = False,
146     labeltop = False,
147     tick1On = False,
148     tick2On = False)
149   plt.ion() if update else plt.ioff()
150   title = "Model rotation: {0} degrees, PI: {1:.2f}%".format(rotation, PI)
151   plt.title(title)
152   tk = 5/max(a)
153   for i in range(len(sp)):
154     linewidth = a[i]*tk
155     c = '#1D3557' if angles[i] >= minimumangle else 'grey'
156     plt.plot([sp[i,0],ep[i,0]],[sp[i,1],ep[i,1]], linewidth=linewidth, color=c)
157   plt.pause(0.01) if update else plt.show()
158
159 def PlotResult(rotations, PI, mediumlimit):
160   bestresult = max(PI)
161   bestrotation = rotations[PI.index(bestresult)]
162   perfect = []
163   medium = []
164   for i, rotation in enumerate(rotations):
165     if PI[i]>mediumlimit:
166       if PI[i] == 100:
167         perfect.append([rotation, PI[i]])
168       else:
169         medium.append([rotation,PI[i]])
170   plt.close()
171   plt.ion()
172   plt.title(f"Highest Performance Index: {bestresult:.2f}% at {bestrotation}"+u"\
      u00b0")
173   plt.xlabel("Rotation around the z-axis(degrees)")
174   plt.ylabel("Performance Index (%)")
175   plt.plot(rotations, PI, color='grey', linewidth = 2)
176   perfect = np.array(perfect)
177   medium = np.array(medium)
178   if len(perfect)>0:
179     plt.scatter(perfect[:,0], perfect[:,1], color='#1D3557', label= 'Good Result
      (100%)')
180   if len(medium)>0:
181     plt.scatter(medium[:,0], medium[:,1], color='#457B9D', label= 'Adequate
      Result (>80%)')
182   plt.legend()
183   plt.show()
184
185 def SaveUpdatedResult2D(sp, ep, a, best_rz, location):
186   sp = RotateArray(sp, best_rz, 'z')
187   ep = RotateArray(sp, best_rz, 'z')
188   output = np.ones((len(sp),6))
189   output[:,[0,1]]=sp
190   output[:,[2,3]]=ep
191   output[:,4]=a
192   trto.savecsv(output, location, f'updatedstruct.csv')
193
194 '''
```

```
3D Functions
'''
def Analyse3DPerformance(startpoints, endpoints, a, l, vol, minimumangle, showplots, stepsize):
  rotationsx = range(0, 360, stepsize)
  rotationsy = range(0, 360, stepsize)
  pbar = tqdm(total=len(rotationsx)*len(rotationsy))
  PI = np.zeros((len(rotationsy), len(rotationsx)))
  for i, ry in enumerate(rotationsy):
    for j, rx in enumerate(rotationsx):
      pbar.update()
      pi = PerformanceIndex3D(startpoints, endpoints, rx, ry, a, l, vol, minimumangle)
      PI[i,j] = pi
      if showplots:
        if (rx == 40 and ry == 40) or (rx == 40 and ry ==80) or (rx==120 and ry ==40) or (rx==120 and ry==80):
          PlotRotatedTruss3D(startpoints, endpoints, a, rx, ry, pi, minimumangle, True)
          input("hold")
  pbar.close()
  print(f"Highest performance index found was {np.amax(PI):.1f}%")
  best_indices = np.where(PI == np.amax(PI))
  best_orientations = []
  print("The following combinations have this performance index:")
  for i in range(len(best_indices[0])):
    rx = rotationsx[best_indices[1][i]]
    ry = rotationsy[best_indices[0][i]]
    print(f"({i}) rx: {rx} ry: {ry}")
    best_orientations.append([rx, ry])
  Visualise3DPerformance(PI, rotationsx, rotationsy, minimumangle)
  outcome = None
  while outcome != 'y':
    rx = int(input("Enter the desired value for rx: "))
    ry = int(input("Enter the desired value for ry: "))
    pi = PerformanceIndex3D(startpoints, endpoints, rx, ry, a, l, vol, minimumangle)
    PlotRotatedTruss3D(startpoints, endpoints, a, rx, ry, pi, minimumangle, update=False)
    Visualise3DPerformance(PI, rotationsx, rotationsy, minimumangle)
    outcome = input("Is this the model you would like to continue with? (y/n): ")
  print(f"Chosen orientation rx: {rx}, ry: {ry} with pi: {pi}")
  return pi, rx, ry

def PerformanceIndex3D(sp, ep, rx, ry, a, l, vol, minumangle):
  sp = RotateArray(sp, rx, 'x')
  ep = RotateArray(ep, rx, 'x')
  sp = RotateArray(sp, ry, 'y')
  ep = RotateArray(ep, ry, 'y')
  angles_xz = CalculateAngles(sp, ep, 'xz')
  angles_yz = CalculateAngles(sp, ep, 'yz')
  angles = np.transpose([angles_xz, angles_yz])
  uv = ReportUnprintableVolume(angles, a, l, minimumangle, True)
  return ((vol-uv)/vol)*100

def PlotRotatedTruss3D(sp, ep, a, rotationx, rotationy, pi, minimumangle, update):
  ax = plt.axes(projection='3d')
  ax.set_xlabel("x")
  ax.set_ylabel("y")
  ax.set_zlabel("z")
  # ax.set_xlim(-10, 10)
  # ax.set_ylim(-10,10)
  # ax.set_zlim(-10,10)
  plt.tick_params(
    axis='both',          # changes apply to the x-axis
    labelbottom = False,
    labelright = False,
    labelleft = False,
```

```
    labeltop = False,
    tick1On = False,
    tick2On = False)
  ax.view_init(azim=25, elev=10)
  title = f"rx: {rotationx}, ry: {rotationy} PI: {pi:.2f}%"
  plt.title(title)
  plt.ion() if update else plt.ioff()
  sp = RotateArray(sp, rotationx, 'x')
  ep = RotateArray(ep, rotationx, 'x')
  sp = RotateArray(sp, rotationy, 'y')
  ep = RotateArray(ep, rotationy, 'y')
  tk = 5/max(a)
  angles_yz = CalculateAngles(sp, ep, 'yz')
  angles_xz = CalculateAngles(sp, ep, 'xz')
  for i in range(len(sp)):
    linewidth = a[i]*tk
    c = '#1D3557' if angles_yz[i] >= minimumangle and angles_xz[i] >=
    minimumangle else 'grey'
    ax.plot([sp[i,0],ep[i,0]],[sp[i,1],ep[i,1]],[sp[i,2], ep[i,2]], linewidth=
    linewidth, color=c)
  plt.pause(0.01) if update else plt.show()
  pt = Path(f'C:\\Users\\Marijn\\Google Drive\\Civiel\\Afstuderen\\Python\\
    RotatedPlots\\3DRotated_{rotationx}{rotationy}.png')
  if rotationx == 40:
    if rotationy == 40 or rotationy == 80:
      plt.savefig(pt)
  if rotationx == 120:
    if rotationy == 40 or rotationy == 80:
      plt.savefig(pt)

def Visualise3DPerformance(PI, rotationsx, rotationsy, minimumangle):
  plt.ion()
  fig, ax = plt.subplots(1,1, frameon=False)
  plt.title(f'Members that can be manufactured if minimumangle = {minimumangle}')
  ticklocations = list(range(0, len(rotationsx)+1, int(len(rotationsx)/12)))
  ticks = list(range(0, 361, int(360/12)))
  ax.set_xticks(ticklocations)
  ax.set_yticks(ticklocations)
  ax.set_xticklabels(ticks, rotation=90)
  ax.set_yticklabels(ticks)
  ax.set_xlabel("Rotation around x-axis")
  ax.set_ylabel("Rotation around y-axis")
  # c = ax.pcolor(PI, cmap='YlOrBr')
  c = ax.pcolor(PI, cmap='Greys')
  cbar = fig.colorbar(c, ticks=[0,20,40,60,80,np.amax(PI)])
  cbar.ax.set_yticklabels([0, '20%', '40%', '60%', '80%', f'{ceil(np.amax(PI))}%'
    ])
  plt.show()


def SaveUpdatedResult3D(sp, ep, a, best_rx, best_ry, location):
  print('A csv named updatedstruct.csv with members rotated to the best
    orientation will now be saved in the modelpath')
  sp = RotateArray(sp, best_rx, 'x')
  ep = RotateArray(ep, best_rx, 'x')
  sp = RotateArray(sp, best_ry, 'y')
  ep = RotateArray(ep, best_ry, 'y')
  output = np.ones((len(sp), 8))
  output[:,[0,1,2]]=sp
  output[:,[3,4,5]]=ep
  output[:,6]=a
  trto.savecsv(output, location, f'updatedstruct.csv')

if __name__ == '__main__':
  if len(sys.argv) > 1:
    modelpath = sys.argv[1]
    minimumangle = sys.argv[2]
  else:
    pass
```

```
    # print("Where is the csv file of the model?")
    # filepath = input("Model Location: ")
    # print("Minimum angle from horizontal: ")
    # minimumangle = int(input("Angle: "))
  #2D example:
  # modelpath=Path("C:\\Users\\Marijn\\Google Drive\\Civiel\\Afstuderen\\Python\\
    DefaultModelName")
  #3D example:
  modelpath = Path("C:\\Users\\Marijn\\Google Drive\\Civiel\\Afstuderen\\Python\\
    Default_233_1320")
  # print("What is the minimum printing angle?")
  # minimumangle = 25
  # filepath = 'C:\\Users\\Marijn\\Google Drive\\Civiel\\Afstuderen\\Python\\
    CaseStudy1\\struct_bd10.csv'
  filepath = modelpath.joinpath("struct.csv")
  minimumangle = 60
  ReportPerformance(filepath, minimumangle, stepsize=1, showplots=False)
```

E.3 trussopttools

```
from tqdm import tqdm
import numpy as np
import itertools
from math import gcd, ceil
from shapely.geometry import Point, LineString, Polygon
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import csv
import os
from pathlib import Path
import time
import threading

#Process boundary conditions
def readcsv(path, astype='int'):
    print("Looking for csv at {}".format(path),flush=True)
    if os.path.exists(path):
        csv_file = open(path)
        csv_reader = csv.reader(csv_file)
        out = []
        newline = []
        #Iterate over data and add to output
        for line in csv_reader:
            for value in line:
                if astype == 'int':
                    newline.append(int(value))
                if astype == 'float':
                    newline.append(float(value))
            out.append(newline)
            newline=[]
        print(f"csv file read with dimensions {len(out)} x {len(out[0])}")
        return out
    else:
        raise Exception("No csv file found on this location")
def bcs(Nd, dof, directory):
    bcspath = os.path.join(directory, "bcs.csv")
    bcs = readcsv(bcspath, astype='int')
    for bc in bcs:
        node_nr = bc[0]
        bc_type = bc[1] # 0: free node, 1: x fixed, 2: y fixed, 3: both x and y
    fixed
        if bc_type == 1:
            dof[node_nr] = [0,1]
        elif bc_type == 2:
            dof[node_nr] = [1,0]
        elif bc_type == 3:
            dof[node_nr] = [0,0]
    return dof
def loads(Nd, f, width, directory):
    ldspath = os.path.join(directory, "loads.csv")
    lds = readcsv(ldspath, astype='float')
    for i,nd in enumerate(Nd):
        loadadded=False
        for ld in lds:
            nodenumber=int(ld[0])
            xcomponent=ld[1]
            ycomponent=ld[2]
            if nodenumber==i:
                f+=[xcomponent,ycomponent]
                print("Load added at node {0} with components {1},{2}".format(
    nodenumber,xcomponent,ycomponent), flush=True)
                loadadded=True
                continue
        if loadadded:
            continue
        else:
            f+=[0,0]
```

```
    return f
def ReadPassiveAreas(path, initialpolygon):
    passivepath = os.path.join(path, "passives.csv")
    passiveareas = readcsv(passivepath, astype='int')
    polygonlist = []
    for pa in passiveareas:
        points = int(len(pa)/2)
        print("points: ", points)
        xcoordinates = pa[:points]
        ycoordinates = pa[points:]
        print(xcoordinates, ycoordinates)
        polygonlist.append(MakePolygon(xcoordinates,ycoordinates))
    return SubtractPassiveAreasFromInitialPolygon(initialpolygon, polygonlist)
def MakePolygon(xcoordinates, ycoordinates):
    if len(xcoordinates) != len(ycoordinates):
        raise Exception("Passive area can not be created, x and y coordinates are not compliant ({0}, {1})".format(len(xcoordinates), len(ycoordinates)))
    pointlist = []
    for i in range(len(xcoordinates)):
        pointlist.append(Point(xcoordinates[i], ycoordinates[i]))
    poly = Polygon([[p.x, p.y] for p in pointlist])
    return poly
def SubtractPassiveAreasFromInitialPolygon(initialpolygon, passiveareas):
    poly = initialpolygon
    fig, (p1,p2,p3) = plt.subplots(nrows=3, ncols=1)
    plotpolygon(poly)
    for pa in passiveareas:
        plotpolygon(pa)
        poly = Polygon.difference(poly, pa)
    plotpolygon(poly)
    plt.savefig("{0}/testplot.png".format(os.path.dirname(os.path.abspath(__file__))))
    print(poly.exterior, flush=True)
    print(poly.interiors, flush=True)
    input("wait")
    return poly
def plotpolygon(polygon):
    p = polygon
    x,y = p.exterior.xy
    plt.plot(x,y)
    for interior in polygon.interiors:
        x,y = interior.xy
        plt.plot(x,y)
    plt.pause(0.01)
def savecsv(array, directory, filename):
    filepath = directory+"\\"+filename
    #Check whether file is present:
    if (os.path.exists(filepath)):
        print("Removing file at {0}".format(filepath))
        os.remove(filepath)
    try:
        np.savetxt(filepath, array, delimiter=",", fmt='%s')
        print(f"CSV with size {len(array)} x {len(array[0])} created in {filepath}")
    except Exception:
        print("Failed to save file")
        print(Exception)
#Save plot to file
def SaveToFile(modelpath, iteration):
    #Check if modelfolder is present:
    if os.path.exists(modelpath)==False:
        os.mkdir(modelpath)
    image_filename = Path("{0}\\Plots\\plot_{1}.png".format(modelpath, iteration))
    if os.path.exists(os.path.dirname(image_filename)) == False:
        os.mkdir(os.path.dirname(image_filename))
        print("Directory Created in {0}".format(os.path.dirname(image_filename)))
    try:
        if os.path.isfile(image_filename):
```

```
            print("Removing file that is already present..")
            os.remove(image_filename)
        plt.savefig(image_filename, transparent=True)
        print("Plot of iteration {0} saved at {1}".format(iteration, image_filename), flush=True)
    except:
        print("Failed to save plot of iteration {0} to {1}".format(iteration, image_filename), flush=True)

def CreateGroundStructure(Nd, jc, PML, convex, poly, minimum_angle, maximum_initial_length):
    print("Now creating ground structure...")
    pbar = tqdm(total=binomial(len(Nd), 2))
    for i, j in itertools.combinations(range(len(Nd)), 2):
        pbar.update()
        dx, dy = abs(Nd[i][0] - Nd[j][0]), abs(Nd[i][1] - Nd[j][1])
        angleok, lengthok = CheckAngleAndLength(minimum_angle, maximum_initial_length, dx, dy)
        if angleok:
            if gcd(int(dx), int(dy)) == 1 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), lengthok] )
    pbar.close()
    return PML

def CheckAngleAndLength(minimum_angle, maximum_initial_length, *argv):
    deltas = [arg for arg in argv]
    length = np.sqrt(sum([delta**2 for delta in deltas]))
    lengthok = length<maximum_initial_length
    angles=[]
    for i in range(len(deltas)-1):
        angles.append(np.arctan(deltas[-1]/deltas[i])*(180/np.pi))
    if min(angles)<minimum_angle:
        return False, lengthok
    if minimum_angle > 0 and len(deltas) == 3 and deltas[2]==0:
        return False, lengthok
    return True, lengthok

def CreateGroundStructure3D(Nd, jc, PML, convex, poly, minimum_angle, maximum_initial_length):
    print("Now creating ground structure...")
    pbar = tqdm(total=binomial(len(Nd), 2))
    PML = np.zeros((binomial(len(Nd),2),4))
    '''
    # Greatest Common Divisor: to eliminate overlapping elements
    '''
    counter=0
    for i, j in itertools.combinations(range(len(Nd)), 2):
        pbar.update()
        dx = abs(Nd[i][0] - Nd[j][0])
        dy = abs(Nd[i][1] - Nd[j][1])
        dz = abs(Nd[i][2] - Nd[j][2])
        angleok, lengthok = CheckAngleAndLength(minimum_angle, maximum_initial_length, dx, dy, dz)
        if angleok:
            if (gcd(gcd(int(dx), int(dy)), int(dz)))==1 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 + dz**2), lengthok ] )
                    PML[counter] = [i, j, np.sqrt(dx**2 + dy**2 + dz**2), lengthok]
                    counter+=1
    pbar.close()
    return PML

def binomial(n, r):
    ''' Binomial coefficient, nCr, aka the "choose" function
```

```
        n! / (r! * (n - r)!)
    '''
    p = 1
    for i in range(1, min(r, n - r) + 1):
        p *= n
        p //= i
        n -= 1
    return p
def TempPrintCn3D(Cn, Nd):
    startindices = Cn[:,0].astype(int)
    endindices = Cn[:,1].astype(int)
    startpoints = Nd[startindices]
    endpoints = Nd[endindices]
    ax = plt.axes(projection='3d')
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_zlabel("z")
    plt.ion()
    for i in range(len(startpoints)):
        startx, starty, startz = startpoints[i][0], startpoints[i][1],
    startpoints[i][2]
        endx, endy, endz = endpoints[i][0], endpoints[i][1], endpoints[i][2]
        ax.plot([startx, endx], [starty, endy], [startz, endz])
        plt.pause(0.01)
def AnalysePML(PML, Nd, number_of_intervals):
    active_members = [member for member in PML if member[3]==True]
    print("{0} active members counted".format(len(active_members)))
    angles = []
    for member in active_members:
        Node1 = Nd[int(member[0])]
        Node2 = Nd[int(member[1])]
        dx = abs(Node1[0]-Node2[0])
        dy = abs(Node1[1]-Node2[1])
        if dx==0:
            angles.append(90)
            continue
        angle = np.arctan(dy/dx)*(180/np.pi)
        angles.append(angle)
    intervals = np.linspace(0,90,number_of_intervals+1)
    intensities = ReportAngleIntensities(angles, intervals)
    print(intensities)
    PlotIntensities(intensities, intervals)

def ReportAngleIntensities(angles, intervals):
    intensities = np.zeros(len(intervals)-1)
    for angle in angles:
        for i in range(len(intervals)-1):
            lower_limit = intervals[i]
            upper_limit = intervals[i+1]
            if angle>lower_limit and angle<=upper_limit:
                intensities[i]+=1
    return intensities

def PlotIntensities(intensities, intervals):
    plt.close()
    labels = []
    degree_sign= u'\N{DEGREE SIGN}'
    for i in range(len(intervals)-1):
        label = "{0}{2} - {1}{2}".format(intervals[i], intervals[i+1],degree_sign
    )
        labels.append(label)
    print(labels)
    fig1, ax1 = plt.subplots()

    theme = plt.get_cmap('copper')
    ax1.set_prop_cycle("color", [theme(1. * i / len(intensities))
                                 for i in range(len(intensities))])

    _, _ = ax1.pie(intensities, labels = labels)
```

```
259
260     ax1.axis('equal')
261
262     # total = sum(intensities)
263     # plt.legend(
264     # loc='upper left',
265     # labels=labels,
266     # prop={'size': 11},
267     # bbox_to_anchor=(0.0, 1),
268     # bbox_transform=fig1.transFigure
269     # )
270
271     plt.show()
272
273 def plotTruss3D(Nd, Cn, a, q, threshold, str, modelpath, update = True):
274     output=[]
275     ax = plt.axes(projection='3d')
276     ax.set_xlabel("x")
277     ax.set_ylabel("y")
278     ax.set_zlabel("z")
279     # plt.ion()
280     plt.ion() #if update else plt.ioff()
281     # plt.clf(); plt.axis('off'); plt.axis('equal');  plt.draw()
282     plt.title(str)
283     tk = 5 / max(a)
284     for i in [i for i in range(len(a)) if a[i] >= threshold]:
285         # print("plotiteration {0}".format(i))
286         if all([q[lc][i]>=0 for lc in range(len(q))]): c = 'r'
287         elif all([q[lc][i]<=0 for lc in range(len(q))]): c = 'b'
288         else: c = 'tab:gray'
289         pos = Nd[Cn[i, [0, 1]].astype(int), :]
290         #Preparing the output
291         thickness = a[i] *tk
292         color = 0
293         if c=='r': color = 1;
294         if c=='b': color = 2;
295         if update == False:
296             outputline = []
297             outputline.extend(pos.flatten())
298             outputline.extend([a[i], color])
299             output.append(outputline)
300
301         #Plotting the truss
302         xs = pos[:,0]
303         ys = pos[:,1]
304         zs = pos[:,2]
305         ax.plot(
306             [xs[0],xs[1]], [ys[0],ys[1]], [zs[0],zs[1]], color=c, linewidth =
    thickness)
307     SaveToFile(modelpath, str)
308     if update:
309         plt.pause(0.01)
310     else:
311         savecsv(output,modelpath, "struct.csv")
312         plt.show()
313
314 def plotinput(poly, Nd, f, dof, final=False ):
315     if final == False:
316         ax = plt.axes(projection='3d')
317         ax.set_xlabel("x")
318         ax.set_ylabel("y")
319         ax.set_zlabel("z")
320         plt.title('Input')
321         plt.ion()
322     else:
323         plt.ioff()
324     plotpolygon(poly)
325     constrained_nodes = []
326     f = np.array(f)
```

```
    fmax = max(abs(f))
    f = np.split(f,len(f)/3)
    for i, node in enumerate(Nd):
        d = dof[i]
        if any(np.where(d==0, True, False)):
            constrained_nodes.append(node)
        if any(np.where(f[i]!=0, True, False)):
            for i, force in enumerate(f[i]):
                if force != 0:
                    plotdirectionline(node, i, (force*4/fmax))

    constrained_nodes = np.array(constrained_nodes)
    plt.plot(constrained_nodes[:,0],constrained_nodes[:,1], constrained_nodes
    [:,2], 'k1')
    plt.pause(0.01)
    plt.show()

def plotdirectionline(point, direction, length):
    # This function creates a line in a certain direction. It can be used to
    visualise an applied force or boundary conditions, including its direction.
    if direction == 0:
        plt.quiver(point[0], point[1], point[2], length, 0, 0)
    if direction == 1:
        plt.quiver(point[0], point[1], point[2], 0, length, 0)
    if direction ==2:
        plt.quiver(point[0], point[1], point[2], 0, 0, length)
    plt.pause(0.01)

def CreateCorbel(Nd, width, height, depth, difference):
    overhang = width/4
    corbel_height = width
    x = [overhang, overhang, 0,0, overhang, overhang, width-overhang, width-
    overhang, width, width, width-overhang, width-overhang]
    y = [height, height*3/4-difference, height*3/4-difference, height*3/4-
    difference-corbel_height/2,height*3/4-difference-corbel_height,0,0,height
    *3/4-corbel_height,height*3/4-corbel_height/2, height*3/4, height*3/4, height
    ]
    # x = [overhang, overhang, 0,0,overhang,overhang,overhang+width,overhang+
    width,2*overhang+width,2*overhang+width,overhang+width, overhang+width]
    # y = [height, height*3/4-difference, height*3/4-difference, height*3/4-
    difference-corbel_height/2,height*3/4-difference-corbel_height,0,0,height
    *3/4-corbel_height,height*3/4-corbel_height/2, height*3/4, height*3/4, height
    ]
    pts = [(x[i],y[i]) for i in range(len(x))]
    dof, f = np.ones((len(Nd),3)), []
    for i, nd in enumerate(Nd):
        # if (nd[0] == width/4 or nd[0]==3*width/4):
        #     if (nd[1] == height/4 or nd[1]==3*width/4):
        #         if (depth == 0 or depth == depth):
        #             dof[i]==[0,0,0]
        if nd[1] == 0 or nd[1] == height:
            if nd[0] >= overhang+1 and nd[0] <= width-overhang-1:
                if nd[2] >=1 and nd[2] <=depth-1:
                    dof[i] = [1,0,1]
        if nd[1] == height*3/4-difference:
            if nd[0] == 1:
                f += [0, -1, 0]
            else:
                f+=[0,0,0]
        elif nd[1] == height*3/4:
            if nd[0] == width-1:
                f += [0, -2, 0]
            else:
                f+=[0,0,0]
        else:
            f+=[0,0,0]

    return Polygon(pts), f, dof

```

```
386 def CountVolumes(Cn, a, q, threshold):
387     steelvol = 0
388     concvol = 0
389     l = Cn[:,2]
390     for i in [i for i in range(len(a)) if a[i] >= threshold]:
391         # print("plotiteration {0}".format(i))
392         if all([q[lc][i]>=0 for lc in range(len(q))]):
393             c = 'r'
394         elif all([q[lc][i]<=0 for lc in range(len(q))]):
395             c = 'b'
396         else:
397             c = 'tab:gray'
398         if c == 'r' or c == 'tab:gray':
399             steelvol+=a[i]*l[i]
400         if c == 'b':
401             concvol+=a[i]*l[i]
402     return steelvol, concvol
```

E.4 Adjust Member Diameter

```
import csv
import path
import numpy as np
import os
def ReadStructCSV(model_path, filename = 'struct.csv'):
    file_path = os.path.join(model_path, filename)
    struct = []
    with open(file_path) as csvfile:
        data = csv.reader(csvfile)
        for line in data:
            for i, cell in enumerate(line):
                if i == 6:
                    line[i] = float(line[i])
                else:
                    line[i] = int(float(line[i]))
            struct.append(line)
    return struct

def CountVolumesCSV(struct):
    Volume_Tension = 0
    Volume_Compression = 0
    for mem in struct:
        length = np.sqrt((mem[3]-mem[0])**2+(mem[4]-mem[1])**2+(mem[5]-mem[2])
    **2)
        area = mem[6]
        volume = area*length
        if mem[-1] == 1:
            Volume_Tension += volume
        elif mem[-1] == 2:
            Volume_Compression += volume
    return Volume_Tension, Volume_Compression

def AdjustBarDiameters(model_path, minimum_diameter, filename = "struct.csv"):
    print("Adjusting bar diameters...")
    file_path = os.path.join(model_path, filename)
    a_min = 0.25*np.pi*minimum_diameter**2
    struct = ReadStructCSV(model_path, filename = filename)
    Vol_T, Vol_C = CountVolumesCSV(struct)
    print(f'Original volumes tension/compression = {Vol_T:.2f}/{Vol_C:.2f}')
    for i, member in enumerate(struct):
        if member[-1] == 1 and member[6] < a_min:
            struct[i][6] = a_min
    Vol_T_new, Vol_C_new = CountVolumesCSV(struct)
    print(f'Adjusted volumes tension/compression = {Vol_T_new:.2f}/{Vol_C_new:.2f
    }')
    difference_T = Vol_T_new - Vol_T
    difference_C = Vol_C_new - Vol_C
    print(f'{Vol_T_new-Vol_T} added')
    filename = f"struct_bd{minimum_diameter}.csv"
    filepath = os.path.join(model_path, filename)
    np.savetxt(filepath, struct, delimiter=',', fmt= '%s')
    print(f'Updated CSV saved at {filepath}')

if __name__ == "__main__":
    model_location = path.Path("C:\\Users\\Marijn\\Google Drive\\Civiel\\
    Afstuderen\\Python\\CaseStudy1")
    AdjustBarDiameters(model_location, 10)
```

Appendix F

Calculation of pile cap

VAN ROSSUM RAADGEVENDE INGENIEURS

Ir. D.J. Kluft
Ir. A.G. van der Sluis

Ir. A.F.H.M. Melssen
Ir. R.E. van Alphen
Ir. M. Eschweiler

Project	**RAI hotel**
Ordernummer	8943
Opdrachtgever	Pleijsier Bouw
Rapportnummer	B003
Omschrijving	Wapening poeren
Fase	Uitvoeringsgereed ontwerp

Status	Datum	Omschrijving
Definitief	6-1-2017	Eerste uitgave

Opgesteld door:	Gecontroleerd door:	Voor akkoord:
ir. W. Vos	ir. M.C. Kranenburg	ir. R.E. van Alphen

Van Rossum Raadgevende Ingenieurs bv Amsterdam

Pedro de Medinalaan 3a
1086 XK Amsterdam
Postbus 37290
1030 AG Amsterdam
T +31(0)20 615 37 11
amsterdam@vanrossumbv.nl
www.vanrossumbv.nl

Van Rossum Raadgevende Ingenieurs bv Rotterdam

Westblaak 5e
3012 KC Rotterdam
T +31(0)10 404 51 11
rotterdam@vanrossumbv.nl
www.vanrossumbv.nl

Bank NL53 INGB 0006 6632 57
KvK 34 147 396
BTW NL 8101.54.869.B.01

Op alle door ons aanvaarde opdrachten is de DNR 2011 Rechtsverhouding opdrachtgever-architect, ingenieur en adviseur van toepassing.
Lid NLingenieurs

VAN ROSSUM RAADGEVENDE INGENIEURS

Referentie documenten

	Document	Organisatie	Datum
1.	8943 - B001 – Gewichts- en stabiliteitsberekening	Van Rossum Raadgevende Ingenieurs	19-08-2016
2.	8943_UO_P1_01 – palenplan	Van Rossum Raadgevende Ingenieurs	09-12-2016
3.	RA16173d1 Funderingsadvies en zakkingsanalyse RAI hotel	CRUX Engineering BV	04-07-2016

VAN ROSSUM RAADGEVENDE INGENIEURS

Inhoudsopgave

VAN ROSSUM RAADGEVENDE INGENIEURS

Inleiding

In deze berekening is de wapening van de fundering op niveau -2 van het RAI hotel te Amsterdam bepaald. Deze berekening betreft verschillende poerconfiguraties. De uitwerking van de berekening vindt plaats aan de hand van de Eurocode. Van toepassing is het bouwbesluit 2012.

De keldervloer op niveau -2 heeft een dikte van 400 mm. De poeren worden in de vloer opgenomen. De hoogte van de poeren is afhankelijk van de poerconfiguratie en varieert tussen de 1600 mm en 2000 mm.

Voor alle verschillende poerconfiguraties is de maatgevende poer bepaald. Voor deze poeren is de wapening berekend aan de hand van de maximaal optredende kolomlast werkend op de poer. De afwijkende poeren – met gedeelde palen en/of excentrisch geplaatste kolommen – zijn apart berekend.

De berekening is uitgevoerd op basis van het paaldraagvermogen van de funderingspalen bepaald in het funderingsadvies van CRUX Engineering BV en aangegeven op het palenplan of op basis van de kolomlasten bepaald in berekening B001. Conform NEN-EN 1992-1-1 zijn gedrongen poeren berekend op basis van de staafwerktheorie. Voor de 16-paals poeren is een aanvullende berekening op basis van de buigtheorie uitgevoerd. De maatgevende wapening uit beide berekeningen is toegepast. In de berekeningen op basis van de staafwerktheorie zijn de staafkrachten, de boven- en onderknopen en de scheurwijdte getoetst. Daarnaast zijn de minimale staafafstand en de verankeringslengte voor de verschillende configuraties bepaald en gecontroleerd.

VAN ROSSUM RAADGEVENDE INGENIEURS

Uitgangspunten

- Betonklasse C35/45;
- Milieuklasse bovenzijde vloer XC3, XD1;
- Milieuklasse poer XA2, XD3, XF4;
- Wapening B500B;
- Belastingen volgens berekening B001 d.d. 19-08-2016.

VAN ROSSUM RAADGEVENDE INGENIEURS

1. Overzicht poeren en maatgevende kolomlasten

Figuur 1: Overzicht poerconfiguraties

VAN ROSSUM RAADGEVENDE INGENIEURS

Poerconfiguraties	Poernummer	Belasting UGT [kN]	# poeren
Driepaals poer	P12/P13/P14	3x2550	20
Vierpaals poer 2400kN	P15/P16.1/P17	4x2400	18
Vierpaals poer 3000kN	P05.2/P08.2/P09.2/P16.2	4x3000	5
Driepaals lijnpoer	P03.2	3x2500	1
16-paals poer	P01/P02/P03.1/P06/P08.1/P09.1/P10	34000	8
	P04.1	30000	1
	P04.2	34000	1
	P04.3	7650	1
	P05.1	34000	1
	P07.1	34000	1
	P07.2	30000	2
	P07.3	7650	1
	P11.1	34000	1
	P11.2	10200	1

Voor paaldraagvermogen (PDV) zie palenplan (8943_UO_P1_01).
Voor belastingen zie berekening B001.

2. Berekening

In dit hoofdstuk worden de wapeningsconfiguraties voor de poeren berekend. Dit is gedaan voor de poerconfiguraties gegeven in hoofdstuk 1.

Order

Blad nr

Deel

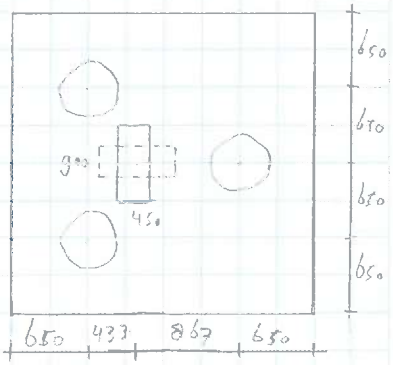
Datum

VAN ROSSUM RAADGEVENDE INGENIEURS

RAI hotel - driepaalrpoer

Gegevens

Funderingspalen	Ø460/560
Kolom	450 x 900
Poer	2600 x 2600 x 1600
Betonklasse	C35/45
Wapeningsklasse	FeB500B
Milieuklasse	XD3
Kolombelasting	
SLS (ULS/1,4)	3 x 1821 kN
ULS	3 x 2550 kN



Betondekking

$c_{min,dur}$ = 40 mm

$c_{min,b}$ = 25 mm

Δc_{dev} = 5 mm

staven tegen bekist oppervlak

$c_{nom} = c_{min,dur} + \Delta c_{dev} = 40 + 5 = 45$ mm

Toepassen c = 60 mm op hoofdwapening i.v.m. scheurwijdte beoordeling

c = 45 mm op flankwapening.

Gedrongen ligger

Eis: $L_{av}/h < 3$

$L_{av} = \sqrt{1300^2 + 650^2} = 1453$ mm

1453/1600 = 0,91 < 3 Voldoet

Order

Blad nr

Deel

Datum

VAN ROSSUM RAADGEVENDE INGENIEURS

Inwendige hefboomsarm z

kies $z = 1250$ mm

$\theta_C = \tan^{-1} \frac{1250}{867} = 55°$

$\theta_B = \tan^{-1} \frac{1250}{\sqrt{433^2 + 650^2}} = 58°$

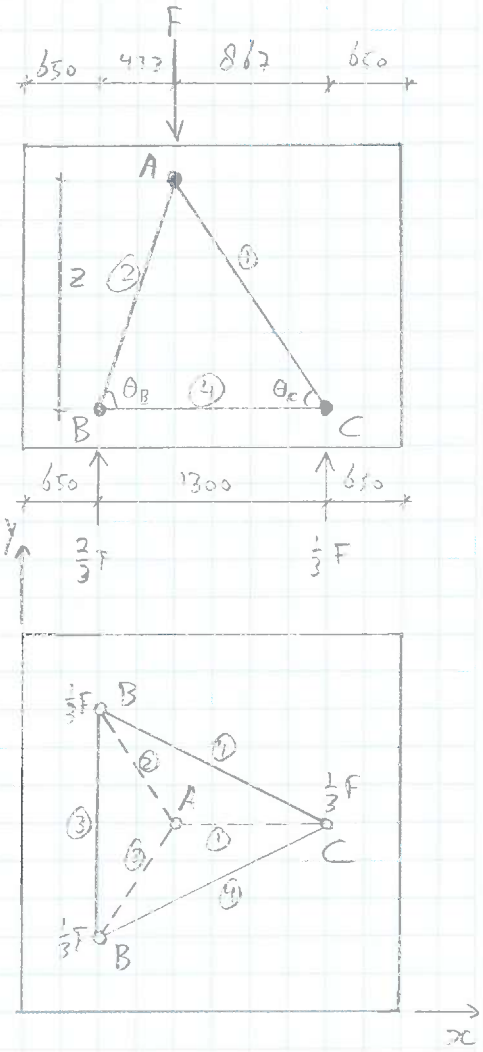
* krachten evenwicht (zie volgende pagina's)

① $F = 3101$ kN

② $F = 3008$ kN

③ $F = 835$ kN

④ $F = 938$ kN



* controleer min hoogte knopen en check z

$h_{knoop} = \frac{F_{trek}}{b_{knoop} \cdot \sigma_{Rd,max}}$

$\sigma_{Rd,max} = k_2 \cdot \nu' \cdot f_{cd}$

- knoop A (C-C-C knoop)

$b_{knoop,A} = b_{kolom} = 450$ mm

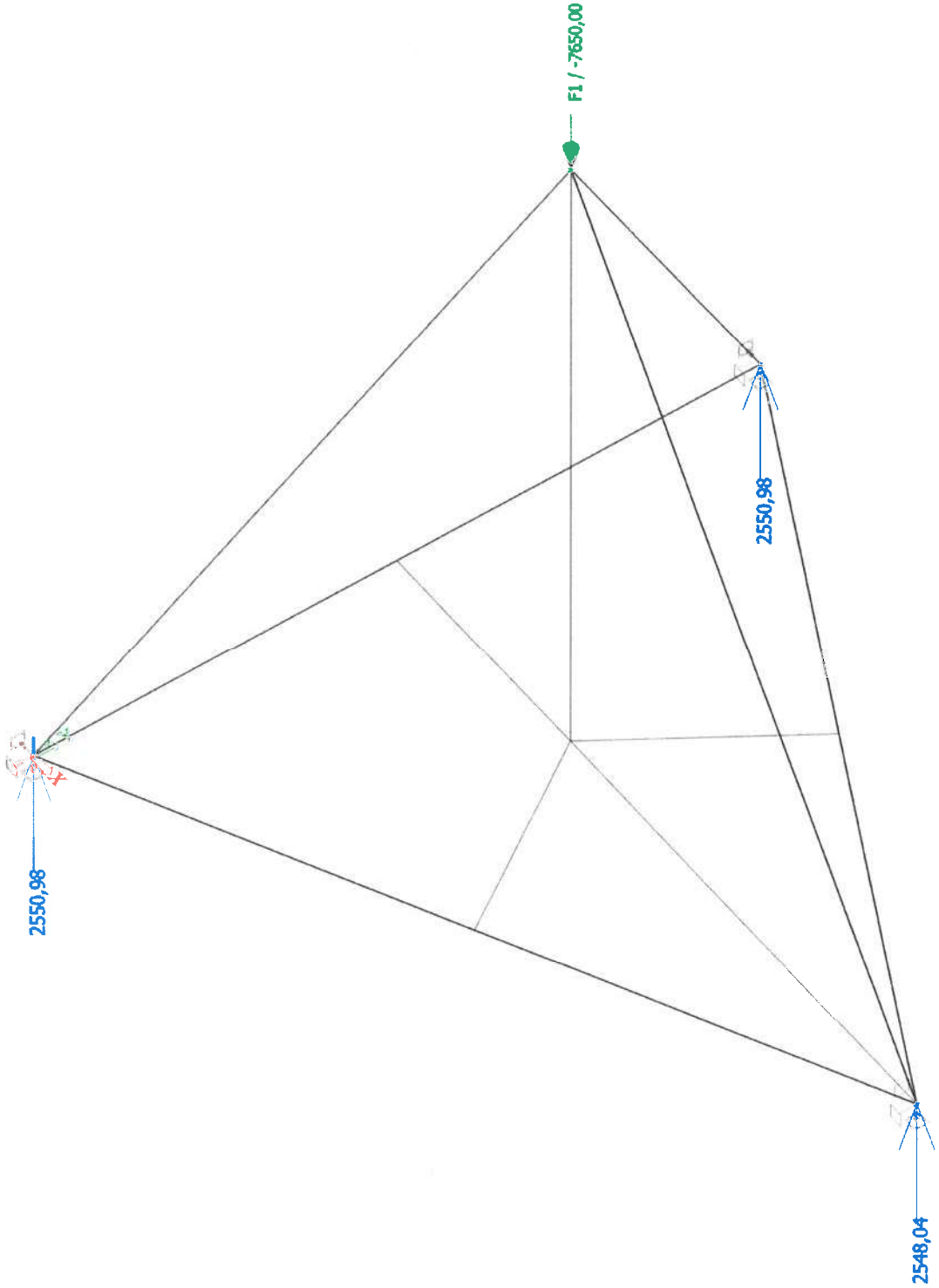
$k_2 = 1,0$

$\nu' = 1 - \frac{35}{250} = 0,86$

$h_{knoop,A} = \frac{988 \cdot 10^3}{450 \cdot 1,0 \cdot 0,86 \cdot \frac{35}{1,5} \cdot 1,1} = 100$ mm

ordernummer: 8943
rapportnummer: B003
blz:

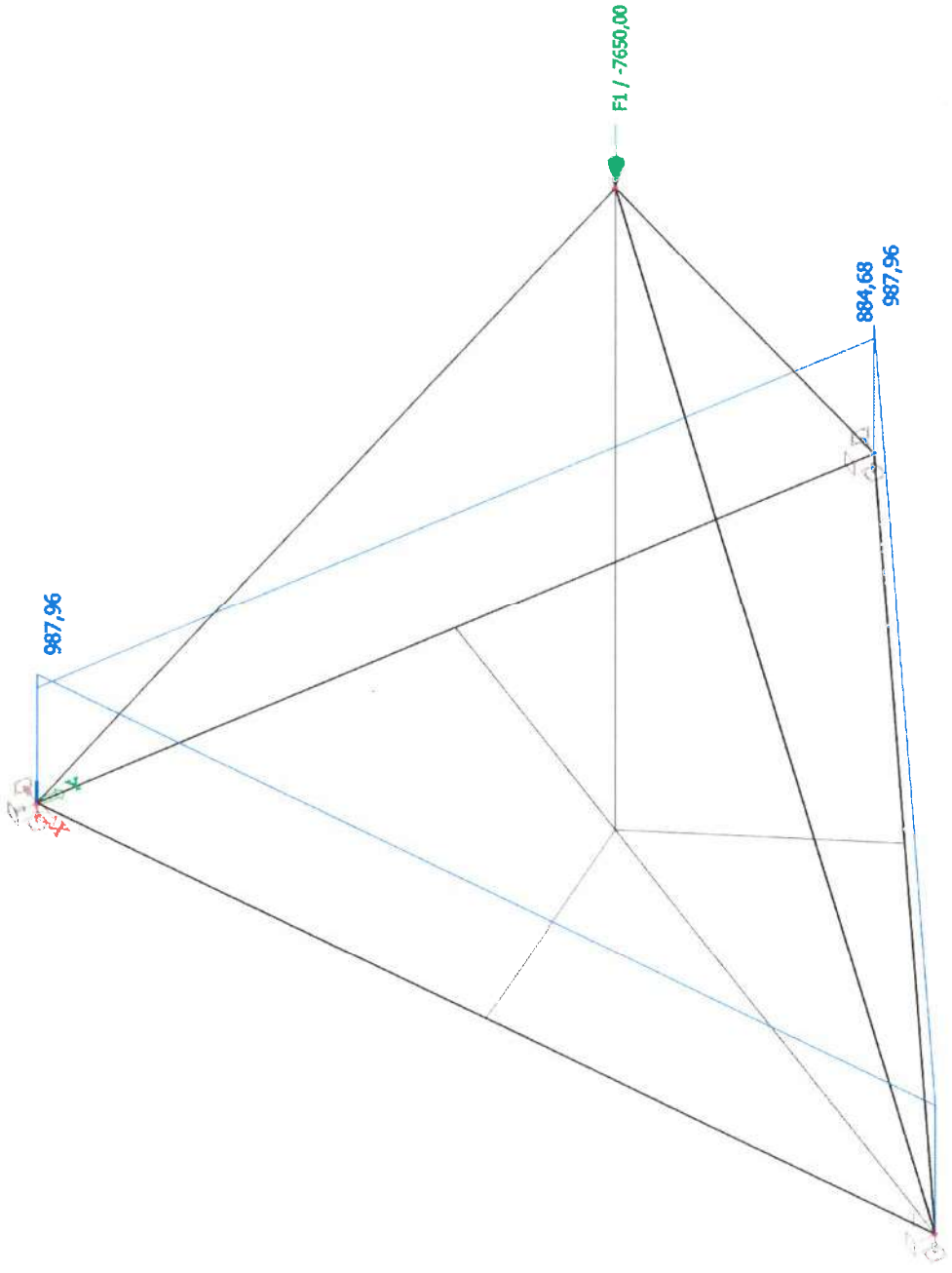
VAN ROSSUM RAADGEVENDE INGENIEURS



Figuur 1: Oplegreacties driepaals blokpoer

ordernummer: 8943
rapportnummer: B003
blz:

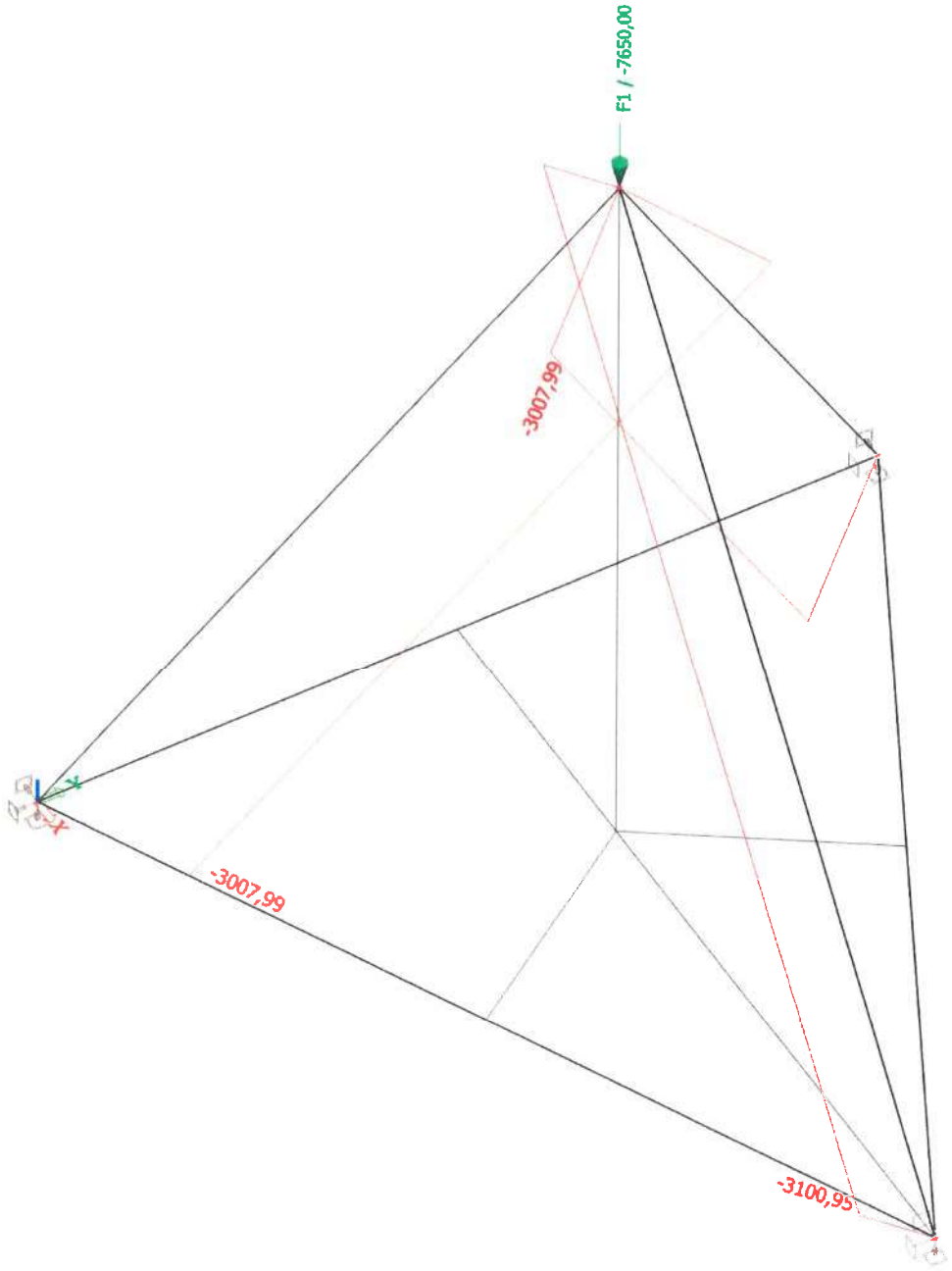
VAN ROSSUM RAADGEVENDE INGENIEURS



Figuur 2: Staafkrachten trekbanden driepaals blokpoer

ordernummer: 8943
rapportnummer: B003
blz:

VAN ROSSUM RAADGEVENDE INGENIEURS



Figuur 3: Staafkrachten drukdiagonalen driepaals blokpoer

Order

Blad nr

Deel

Datum

VAN ROSSUM RAADGEVENDE INGENIEURS

- knoop B/C (C-C-T-T-knoop)

$b_{knoop} = b_{eff} = b_{eq,D} + (s_0 + \frac{1}{2}s)$

$b_{eq,D} = \sqrt{\frac{1}{4}\pi D^2} = \sqrt{\frac{1}{4}\pi 460^2} = 408$ mm

$s_0 = c + \emptyset/2 = 60 + 32/2 = 76$ mm

$s = 0$ mm (wapening in één laag)

$b_{knoop} = b_{eff} = 408 + 76 = 484$ mm

$k_2 = 0{,}75$

$\nu' = 0{,}86$

$h_{knoop,C} = \frac{988 \cdot 10^3}{484 \cdot 0{,}75 \cdot 0{,}86 \cdot \frac{35}{45} \cdot 1{,}1} = 123$ mm

- max hoogte z

$z_{max} = h - \frac{1}{2} h_{knoop,B} - \frac{1}{2} h_{knoop,C} = 1600 - \frac{1}{2} \cdot 100 - \frac{1}{2} \cdot 124$

$= 1488$ mm

$z = 1250 < z_{max}$ Voldoet.

In bovenstaande berekeningen is de sterkte van de knopen verhoogt met 10% i.v.m. $\Theta \geq 55°$ (NEN-EN 1992-1-1 art. 6.5.4 (5))

Order

Blad nr

Deel

Datum

VAN ROSSUM RAADGEVENDE INGENIEURS

Controleren knoopspanningen

* knoop A

maak gebruik van principe hydrostatische knoop

$$\sigma_{Ed,0} = \frac{F_{kolom}}{A_0} = \frac{3 \times 2550 \cdot 10^3}{450 \cdot 900} = 18{,}9 \ N/mm^2$$

$$\sigma_{Rd,max} = k_2 \cdot \nu' \cdot f_{cd} \quad , k_2 = 1{,}0 \quad , \nu' = 0{,}86$$

$$= 1{,}0 \cdot 0{,}86 \cdot \frac{35}{1{,}5} \cdot 1{,}1 = 22{,}1 \ N/mm^2$$

$\sigma_{Ed,0} < \sigma_{Rd,max}$ Voldoet

* knoop B/C

maak gebruik van principe hydrostatische knoop

$$\sigma_{Ed,0} = \frac{F_0}{A_0} = \frac{2550 \cdot 10^3}{\frac{1}{4}\pi \, 460^2} = 15{,}3 \ N/mm^2$$

$$\sigma_{Rd,max} = k_2 \cdot \nu' \cdot f_{cd} \quad , k_2 = 0{,}75 \quad , \nu' = 0{,}86$$

$$= 0{,}75 \cdot 0{,}86 \cdot \frac{35}{1{,}5} \cdot 1{,}1 = 16{,}6 \ N/mm^2$$

$\sigma_{Ed,0} < \sigma_{Rd,max}$ Voldoet

zwaartepunt trekstaven: $s_0 = 76$ mm

$s_0 > \frac{1}{2} h_{knoop,c} = 62$ mm Voldoet

Order
Blad nr
Deel
Datum

VAN ROSSUM RAADGEVENDE INGENIEURS

Controle druk- en trekstaven

* Drukstaven

- drukspanningen

$\sigma_{Rd,max} = f_{cd} = 23{,}3$ N/mm² (geen trekspanning in dwarsrichting)

$\sigma_{Ed,max} = 18{,}9$ N/mm² < $\sigma_{Rd,max}$ Voldoet.

- trekspanningen

$T_1 = 3101$ kN

$a_1 = \min\{a_1^A; a_1^B; a_1^C\} = \frac{h_{knoop,A}}{\cos 55} = \frac{100}{\cos 55}$

$= 174$ mm

$H = \sqrt{z^2 + l_g^2} = \sqrt{1250^2 + 867^2} = 1521$ mm

$\frac{H}{2} = 761$ mm < $b = 2600$ mm

$h = \frac{H}{2} = 761$ mm

$T = \frac{1}{4}\left(1 - 0{,}7\,\frac{a_1}{h}\right) F_1 = \frac{1}{4}\left(1 - 0{,}7\,\frac{174}{761}\right) \cdot 3101 = 651$ kN

$A_{s,ben} = \frac{2T}{f_{yd} \cdot H} = \frac{2 \cdot 651 \cdot 10^3}{435 \cdot 1521} = 1968$ mm²/m

$A_{s,hor} = A_{s,ben} \cdot \sin\theta = 1968 \cdot \sin 55 = 1612$ mm²/m

$A_{s,ver} = A_{s,ben} \cdot \cos\theta = 1968 \cdot \cos 55 = 1129$ mm²/m

$A_{s,min} = 0{,}001 \cdot b_w \cdot 1000 = 0{,}001 \cdot 484 \cdot 1000 = 984$ mm²/m (per zijde)

Toepassen: horizontaal Ø16-120 = 1676 mm²/m

verticaal Ø16-175 = 1149 mm²/m

Order

Blad nr

Deel

Datum

VAN ROSSUM RAADGEVENDE INGENIEURS

* Trekstaven

③: $F_{o.l} = 885$ kN

$$A_{s,ben,y} = \frac{885 \cdot 10^3}{435} = 2034 \text{ mm}^2$$

④: $F = 988$ kN

$F_x = 988 \cdot \cos 27 = 880$ kN

$F_y = 988 \cdot \sin 27 = 449$ kN

$$A_{s,ben,x} = \frac{880 \cdot 10^3}{435} = 2023 \text{ mm}^2$$

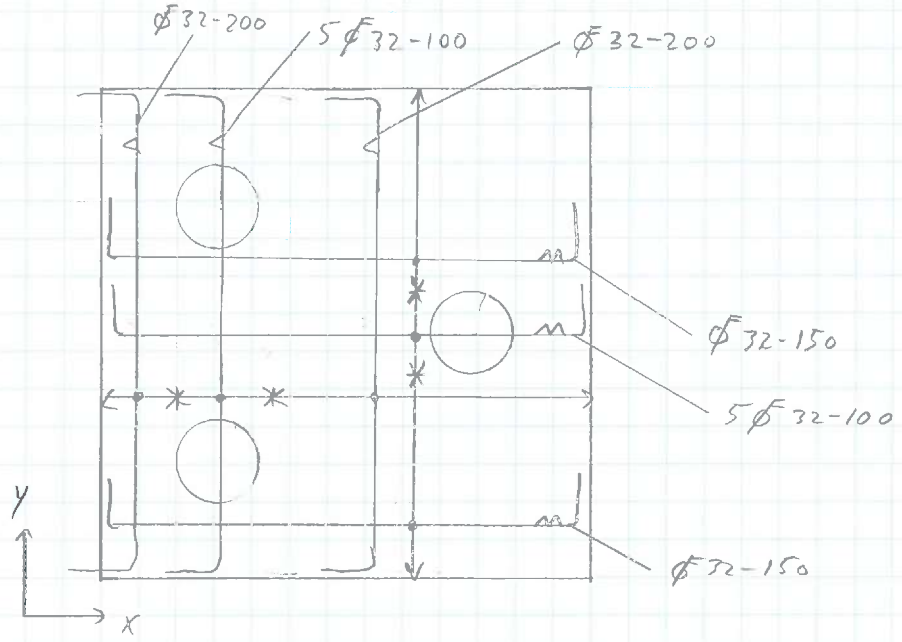
$$A_{s,ben,y} = \frac{449 \cdot 10^3}{435} = 1032 \text{ mm}^2$$

Toepassen: x-richting: 5 Ø 32 $A_s = 4021$ mm²

y-richting

boven de 2 palen: 5 Ø 32 = 4021 mm²

1 paal + erbussen: 6 Ø 32 = 4021 mm²



Order

Blad nr

Deel

Datum

VAN ROSSUM RAADGEVENDE INGENIEURS

Scheurwijdte & scheurafstand

* Toelaatbare scheurwijdte

$w_{max} = k_x \cdot w_{max,EC}$

$k_x = \frac{\text{Cloegepast}}{c_{nom}} = \frac{60}{45} = 1{,}33 \leq 2$

$w_{max,EC} = 0{,}2$ mm

$w_{max} = 1{,}33 \cdot 0{,}2 = 0{,}27$ mm

* Scheurafstand

$\rho_{eff} = \frac{A_s}{A_{c,eff}}$ $\qquad A_s = 4021$ mm²

$A_{c,eff} = b \cdot h_{c,eff}$, $h_{c,eff} = 2{,}5(h-d)$, $b = 484$ mm

$A_{c,eff} = 484 \cdot 2{,}5 \left(60 + \frac{1}{2} \cdot 32\right) = 91\,960$ mm²

$\rho_{eff} = \frac{4021}{91\,960} = 0{,}044$

$k_1 = 0{,}8$ (hoge aanhechting) , $k_2 = 1{,}0$ (zuivere trek)

$s_{r,max} = 3{,}4 \cdot 60 + \frac{0{,}8 \cdot 1{,}0 \cdot 0{,}425 \cdot 32}{0{,}044} = 451$ mm

* scheurwijdte

$\sigma_s = \frac{F_{rep}}{F_d} \cdot \frac{A_{s,ben}}{A_{s,toe}} \cdot f_{yd} = \frac{1821}{2530} \cdot \frac{2034}{4021} \cdot 435 = 157$ N/mm²

$\varepsilon_{sm} - \varepsilon_{cm} = \frac{157 - 0{,}4 \cdot \frac{3{,}2}{0{,}044}(1 + 6{,}2 \cdot 0{,}044)}{2{,}1 \cdot 10^5} \geq 0{,}6 \cdot \frac{157}{2{,}1 \cdot 10^5}$

$= 5{,}7 \cdot 10^{-4} \geq 4{,}5 \cdot 10^{-4}$

$w_k = s_{r,max} (\varepsilon_{sm} - \varepsilon_{cm}) = 451 \cdot 5{,}7 \cdot 10^{-4} = 0{,}26$ mm

$w_k < w_{max} = 0{,}27$ Voldoet.

Order

Blad nr

Deel

Datum

VAN ROSSUM RAADGEVENDE INGENIEURS

Minimale tussenstaafafstand (NEN-EN 1992-1-1 art. 8.2)

$b = 500$ mm

$a = (b - n \cdot \phi) / (n-1)$, $n = 5$

$= (400 - 5 \cdot 32) / 4$

$= 60$ mm

$a \geq k_1 \cdot \phi = 1,0 \cdot 32$ mm — Voldoet

$a \geq a_g + k_2 = 32 + 5 = 37$ mm — Voldoet

$a \geq 20$ mm — Voldoet.

Verankering

* basis verankeringslengte

$l_{b,rqd} = \frac{\phi}{4} \cdot \frac{\sigma_{sd}}{f_{bd}}$, neem maximale staalspanning ($\sigma_{sd} = 435$ N/mm²)

$f_{bd} = 2,25 \cdot \eta_1 \eta_2 \cdot f_{ctd} = 2,25 \cdot 1,0 \cdot 1,0 \cdot \frac{2,0}{1,5} = 3,3$ N/mm²

$l_{b,rqd} = \frac{32}{4} \cdot \frac{435}{3,3} = 1055$ mm

* Rekenwaarde van verankeringslengte

$c_d = \min\{a/2 ; c_1\} = \min\{30, 45\} = 30$ mm

$c_d < 3\phi \Rightarrow a_1 = a_2 = 1,0$, $a_3 = a_4 = a_5 = 1,0$

$l_{b,min} \geq \max\{317 ; 320 ; 100\} \geq 320$ mm

$l_{bd} = 1,0 \cdot 1,0 \cdot 1,0 \cdot 1,0 \cdot 1,0 \cdot 1055 = 1055$ mm ≥ 320 mm

Order

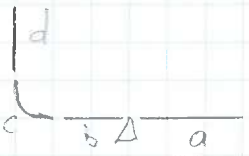
Blad nr

Deel

Datum

VAN ROSSUM RAADGEVENDE INGENIEURS

* Detailering



Kies doorndiameter van 5Ø

Deel a = ½ Dpaal = 230 mm

Deel b = 650 − 45 − 16 − 32 − 2,5·32 = 477 mm

Deel c = π (5 + ½) · 32/4 = 138 mm

Deel d = 1055 − 230 − 477 − 138 = 210 mm > 5Ø

Toets doornd iameter volgens

NEN-EN 1992-1-1 art. 8.3(3)

$$\varnothing_{mmin} \geq F_{bt}\left(\frac{1}{a_b} + \frac{1}{2\varnothing}\right) f_{bd}$$

$$F_{bt} = 0{,}25 \cdot \pi \cdot 32^2 \left(1 - \frac{477 + 230}{1055}\right) \cdot 435 = 115 \text{ kN}$$

$$f_{bd} = \frac{3{,}5}{1{,}5} = 2{,}33 \text{ N/mm}^2$$

$$a_b = \tfrac{1}{2} s = \tfrac{1}{2} \cdot 100 = 50 \text{ mm}$$

$$\varnothing_{m,min} \geq 115 \cdot 10^3 \left(\frac{1}{50} + \frac{1}{2 \cdot 32}\right) / 233 \geq 176 \text{ mm}$$

Kies doornd iameter van 8Ø = 256 mm

Deel a = 230 mm

Deel b = 650 − 45 − 16 − 32 − 4·32 = 429 mm

Deel c = π (8 + ½) · 32/4 = 214 mm

Deel d = 1055 − 230 − 429 − 214 = 182 mm > 5Ø

$$F_{bt} = 0{,}25\ \pi \cdot 32^2 \left(1 - \frac{429 + 230}{1055}\right) \cdot 435 = 131 \text{ kN}$$

$$\varnothing_{m,min} \geq 131 \cdot 10^3 \left(\frac{1}{50} + \frac{1}{2 \cdot 32}\right) / 233 \geq 200 \text{ mm}$$

$\varnothing_m$ = 256 mm > $\varnothing_{m,min}$ Voldoet