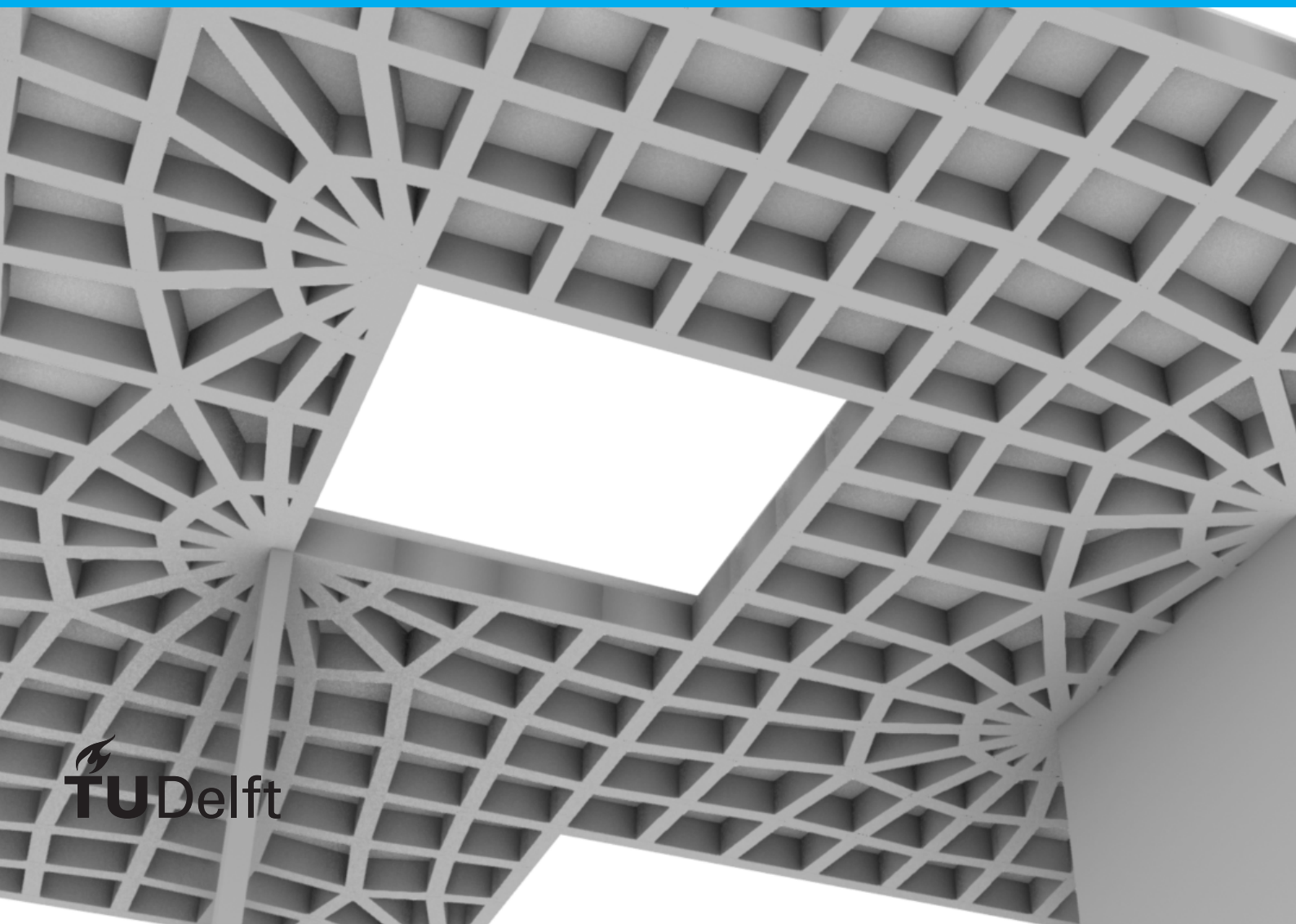
The optimization workflow for the configuration of a modular ribbed floor system

S.C. Pol

04-09-2025



TU Delft

The optimization workflow for the configuration of a modular ribbed floor system

by

S.C. Pol

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday September 29, 2025 at 13:30 PM.

Student number:	4776240	
Project duration:	February 1, 2025 – September 29, 2025	
Thesis committee:	F. Kavoura,	TU Delft
	C. Andriotis,	TU Delft
	J. Cupać,	TU Delft
Supervisor:	R. Oval,	ENPC

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

TU Delft

Preface

During the first year of my Masters, I really enjoyed some courses that looked back at methods from great engineers from the past such as Nervi and Gaudi. They actually already performed structural optimization following principles from nature and physics that resulted in the most beautiful structures. Modern project examples inspired me to take more courses related to parametric design and structural optimization, but never allowed the time to really go in depth, to understand all the principles and the whole process.

When I first approached Robin to help me with a thesis subject within this field, I was quite hesitant when I found out how much programming was involved and the advice to follow a full quarter course on data science and AI seemed for me the hard way to finish my Master. In the end, it was not so scary, of course, and now I am very grateful for all the new skills that I learned and took with me for the start of my career. I want to thank Robin for giving this opportunity to explore this very cool field of engineering and all my other supervisors for asking the most intriguing questions that led me wanting to explore more and more every time, almost forgetting to finish what I was doing first.

Of course I also want to thank my parents for supporting me not only during this thesis, but through my entire education. It probably helped that I knew quite early what I wanted to study and that this had to be at the TU Delft. They always helped me with everything that was needed to get there, which ever path I decided to take, no questions asked.

The most difficult part of writing a thesis for me was that you do it completely on your own for quite some months. So that is why the biggest help for me were the people that were also there day after day usually at the ECHO building going through the same process. So I want to thank everyone who agreed to also be there in the early mornings to get the best spots, who provided free coffee often escalating into too long, chaotic but much needed breaks, who kept on working together through the summer break, who listened when I was frustrated that something was not working, who were also there when I needed to take a day of to do something fun, and who stayed until the end of the day to motivate you to go on working a little bit longer.

S.C. Pol
Delft, September 2025

Contents

Summary

This research addresses the optimization of modular ribbed concrete floor systems as a pathway towards circular construction. Concrete is one of the largest contributors to global carbon emissions, and floors account for a major share of structural weight. Ribbed floor systems can significantly reduce material use and embodied carbon compared to conventional flat slabs. Historically, customized ribbed slabs aligned with principal stress directions demonstrated high structural efficiency but lacked scalability and reusability, whereas standardized waffle slabs offered better manufacturability but lower efficiency.

To address the trade-off of the efficiency of ribbed floor systems and the ability to mass produce and reuse, a modular approach is implemented allowing for different rib designs. The large amount of possible configurations of these modular elements leads to a combinatorial problem with a large solution space. To improve the process and results of this optimization problem compared to heuristic and generic methods a Deep Generative Design workflow is implemented using a Variational Autoencoder and a Gradient Descent algorithm.

The modular characteristic of the problem results in discrete data represented by a bitmap. With this data input structural performance is decoupled from the actual geometry, allowing for a very simple VAE model with only one layer. This model can quickly train on the structural performance given a dataset with configurations of modules created in Grashopper with Karamba3D as FEM solver. The VAE model supports increasing problem sizes without loosing the quality of predictions, but computational cost increase and the ability to optimize decreases. Different problem sizes require some tuning of the VAE model with the number of latent dimensions having the largest effect. After training, the VAE can generate new samples with predicted structural performance.

Training on structural performance allows for a flexible optimization workflow with the possibility to include a wide variety of optimization objectives and constraints without the need to recreate datasets or train the model again.

A Gradient Descent optimizer is implemented in the workflow to optimize new generated VAE samples towards a certain objective. This GD optimizer first draws a large amount of samples in latent space, decodes and calculates the score. With the best scoring samples a gradient descent optimization process is started in latent space. VAEs trained with random created datasets resulted in the best score improvements, also beating scores found by running generic algorithms.

The optimization workflow is flexible for the implementation of new features. Stock constraints are implemented in the calculation of the performance score of the optimization process and can thus be implemented without the need of new datasets and training. The VAE outperformed the evolutionary solver in a problem optimizing for elastic energy with stock constraints by finding configurations with 1.5% lower elastic energy on average. Also did the VAE use on average only 0.5 of 13 modules that were not in stock compared to 2.2 modules for the evolutionary solver. Optimizing for embodied carbon was also done without creating new datasets, but with simply defining new performance functions and optimization objectives. Other implementations such as the optimization of the placement of columns and walls and the possibility to include multiple cross sections need changes in the structural model and thus also require new dataset generation and training and remain challenges for future work.

I

Introduction

1

Introduction

In this section, the research problem, together with its context, objectives, and scope, is explained. This leads to the main research question, sub-questions, and an introduction to the methodology.

Towards a circular floor design

Numbers often cited for the carbon emissions produced by the concrete industry are around four billion tons of CO2 annually, accounting for 8% of global emissions. The Netherlands aims to have a circular economy by the year 2050. From 5 main sectors the government claims the building industry uses 50 percent of our resources and creates a large amount of waste (26).

Although many solutions are being developed, including reducing CO2 emissions from concrete production and using more timber, no single solution appears to be the silver bullet. All the ways we built need to be reconsidered. Many guidelines and laws how to make and asses a circular product are still in development. A study on building design and construction strategies for a circular economy (8) summarizes 16 design and construction strategies from different studies. A modular floor system satisfies most of these strategies such as disassembly, adaptability, modularity, prefabrication, standardization, and component optimization. As concrete floor slabs can be 85% of the total weight of structures, floors are an important building element to target (19). However, material selection and secondary material usage are still more challenging building strategies to comply with when using concrete. One viable approach towards reducing material use and working towards a circular building industry is a modular ribbed floor system.

Ribbed Floor systems

One famous example and main inspiration for most research on this subject are the designs of Pier Luigi Nervi. From 1951-1952 he applied re-usable Ferro-cemento formwork to construct the floors of the Tabacco Factory in Bologna. The formwork allowed for an efficient shaped ribbed floor design that needed less material. The regular grid made it easy to re-use the formwork by moving it to a next section as seen in figure 1.1

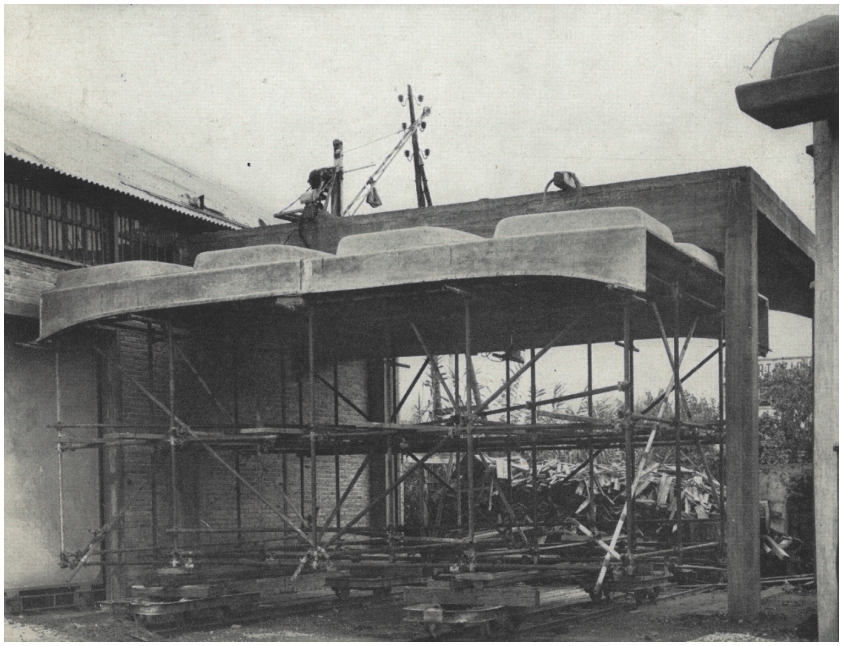


Figure 1.1: Ferro-cemento formwork of the Tabacco Factory in Bologna (13)

The Gatti Wool Factory in Rome built from 1951-1953 implemented the same system but with a slightly more complicated design. The ribs in this design follow the isostatic lines of principal bending moments (19), further optimizing the material use.

Figure 1.2: Gatti Wool Factory (13)

Multiple methods are available to find the principal bending moments in a slab. When Nervi developed his floor systems two main methods of experimental stress analysis were the strain gauge method and photoelasticity. However, as the strain gauge method was very costly and time consuming and photo-elastic methods were more suitable for local stress visualization instead of entire structures, it is believed that Nervi relied more on mathematical methods (12). At the time the patent for isostatic ribbed floors was filed, the Kirchhoff-Love thin plate theory and the approximate and design solutions of the partial differential equations, allowed the use of these mathematical methods.

To visualize the isostatics of principal bending moments the isostatic line tool can be used, which today can be automated in for example GrassHopper. Figure 1.3 shows an example of the analysis of the Gatti Wool Factory by Halpern. The red lines correspond to the maximum principal bending moments and the blue lines to the minimal, which are nicely aligned with the actual placement of the ribs.

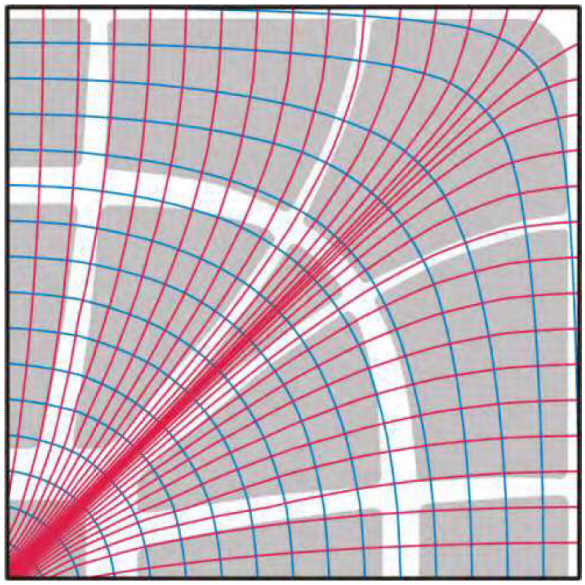


Figure 1.3: Gatti Wool Factory Quarter Slab Analysis (12)

The architectural and artistic value of these designs that automatically result from putting these theories into practice can not be better summarized than by Nervi (13) himself:

> ”The aesthetically satisfying result of the interplay of ribs placed in this way is a clear reminder of the mysterious affinity to be found between physical laws and our own senses.”

Ribbed floor systems offer significant potential for reducing carbon emissions in construction by utilizing less material compared to conventional flat slabs. However, their adoption remains limited due to challenges in mass production and re-usability. Historically, highly customized ribbed floor systems have been designed with ribs aligned along principal stress directions to optimize structural performance. While these solutions are highly efficient, they are difficult to mass-produce and reuse. In contrast, waffle slabs, which are easier to manufacture and reuse, exhibit lower structural efficiency because their ribs do not align with the principal stress directions.

Figure 1.4: Waffle slab (5)

To address this trade-off, this research implements a catalog of six distinct ribbed floor modules proposed by Oval (24) as shown in Figure 1.5 designed to balance both structural efficiency and manufacturability, allowing for circularity.

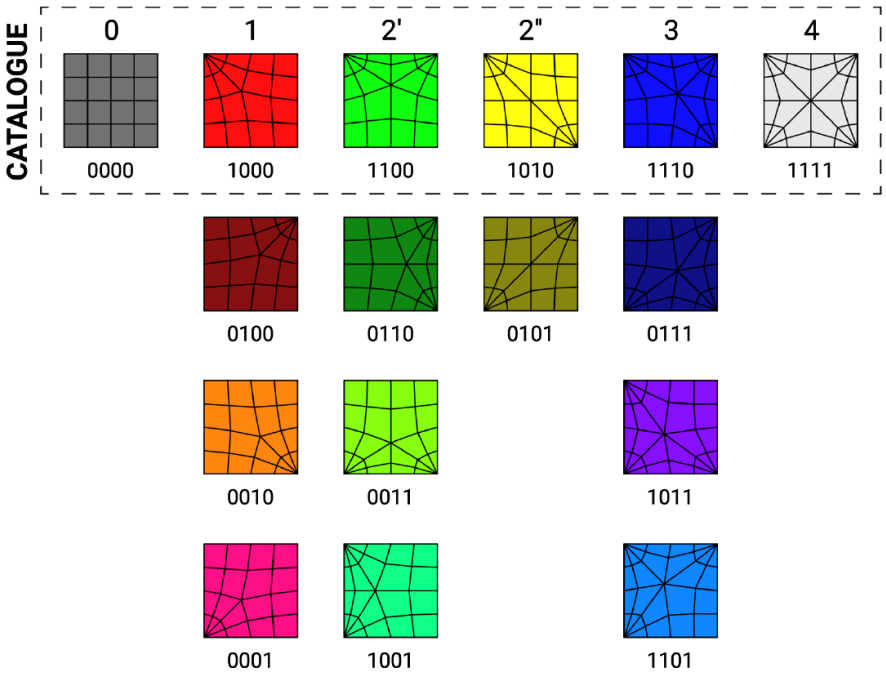


Figure 1.5: Catalog of possible modules and orientations presented as bitmap (24)

All modules have a square boundary of 2 by 2 meters and are subdivided by ribs with a spacing of 50 cm. The ribs are aligned at the same location at the boundaries for all modules to ensure geometric compatibility, while their location is optimized across the module plane. The bi-directional ribs are represented by quad-mesh patterns, yielding both grid- and pole-like areas. In pole-like areas, the ribs converge into a pole at the module's corner points. Also, the ribs must have the same height and width at the interfaces for compatibility at their interfaces, but throughout the modules this could be varied.

The design problem is encoded using a bitmap, with each pixel corresponding to a module. The modules are labeled with a binary sequence indicating the presence (1) or absence (0) of a pole at the module's corner, starting from the top left and proceeding clockwise. The module catalog consists of six modules, resulting in 16 different orientations. The modules and their binary sequence labels are illustrated in Figure 1.5.

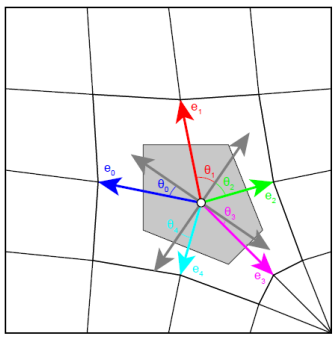


Figure 1.6: Module (24)

Research Problem

The six different modules allow for 16 options for selecting a module and its orientation, presented as bitmaps. Configuring a complete system of these modules leads to a configuration optimization bitmap problem. The number of possible combinations is m^p, where m is the number of different modules and orientations and p is the number of pixels (modules in the floor). For instance, using all 16 module orientations for a floor with 20 modules results in 10^{24} combinations.

Previous research on this modular ribbed floor system ends with a case study assessing several optimization methods to tackle this problem. Methods are compared based on the time taken to find the final solution and the quality of the solution based on elastic energy. Heuristic methods reduced the elastic energy by 21%, while stochastic optimization methods achieved reductions of 25-27%. A manual selection informed by these results yielded slightly better outcomes that were not discovered by the previous methods. The heuristic methods allow for scalability to larger floor plans, whereas the computational time for the optimization methods considered would increase significantly.

By considering multiple load cases and other project requirements, such as stock constraints, a better optimization method can contribute to the modular floor concept.

Adapting generative machine learning methods to optimization problems within the structural engineering field is the current state-of-the-art. An algorithm that can be trained to yield better solutions could effectively address this optimization problem.

Research Questions

The main question this research aims to answer is:

> 'How can the combinatorial problem of modular ribbed floor systems be optimized using the latest optimization techniques, including generative machine learning?'

The sub-questions to address this main question are:

- Which structural analysis objectives need to be optimized?
- Can an existing optimization workflow be adapted to optimize for these objectives?
- What are the advantages and disadvantages of different optimization workflows?

After choosing the optimization workflow, the following questions need to be addressed:

- Can a heuristic or generic method be used to create an initial dataset with configurations?
- How many samples should a dataset have to sufficiently train the model?
- How is the performance of a configuration evaluated?
- Can stock constraints be included in the optimization method?
- Can the chosen method be used to generate better configurations than solvers in GH?
- Do the generated configurations only use the assigned modules?
- What kind of surrogate optimizer is best to include based on the optimization objectives?
- What are the limits on the use of the method?
- Can the trained model optimize other floor plans?
- How can the method be used by others?

Research Objectives

The objectives of this research are to:

- Select a better optimization method based on the capabilities to include structural analysis and the optimization objectives.
- Develop a method that can optimize the configuration of the modular elements based on structural analysis objectives and other project constraints.
- Evaluate the performance of the method with a case study.

Research Scope

The scope of this research is focused on the optimization problem, while other issues concerning the modular ribbed floor system are not considered. The following boundary conditions narrow down the scope:

- The modular floor system is given, but the script should be structured in a way that allows for the inclusion of similar but different modules in the future. If the modules share the same data structure, this should not pose a problem, but retraining may be necessary.

- The algorithm will seek to find the optimal configuration with modules using a given cross-section. The optimization of modules itself or using smaller or multiple different cross-sections could achieve further material decrease, but this is not researched.
- The structural analysis is limited to the chosen optimization objectives and does not include reinforcement or connections.
- At the start more methods are compared and at some moment one method has to be chosen to further develop. The other methods could still be promising, but are not investigated further.
- It should be assessed whether it is possible to create a method that can apply to different floor plans beyond one case study, depending on the available time.

Research Theory and Methods

To solve an optimization problem, knowledge of optimization techniques is necessary. The 10 EC cross-over module Data Science and Artificial Intelligence for Engineers will be used to expand knowledge before starting this thesis.

Grasshopper (GH) and Python are the most popular software available to integrate geometry and optimization algorithms in one interface. Grasshopper is accessible to TU Delft students, and many open-source plugins are available that could assist with structural analysis or optimization such as Karamba3D for structural analysis with Finite Element Analysis (FEA) and Galapagos for generic solvers. Python also contains a wide range of open-source plugins, including the latest developments in generative models with PyTorch.

Research Structure

This thesis consists of three parts containing multiple chapters:

1. The first part covers the introduction and the literature review in which an optimization method and workflow are selected.
2. The second part contains the methodology explaining the structural model and the VAE workflow
3. The third part is the application were the final methodology is tested and stock constraints and other extensions and generalizations are implemented.

2

Literature review

This section aims to provide an overview of the existing state-of-the-art related to the research problem and methodology.

A literature review needs to address the questions: “Can an existing optimization workflow be adapted to optimize for these objectives?” and “What are the advantages and disadvantages of different optimization workflows?”. Following the literature review, it should become clear which workflow and methods will be employed and how they fit into the knowledge gap.

Structural optimization is a rapidly evolving field, predominantly addressing optimization problems with continuous variables. Bitmap problems are common in other fields of study, yet it is unclear whether structural optimization has previously been combined with bitmap problems. The literature review should determine if this knowledge gap truly exists.

The literature review will commence with explaining topology optimization of ribbed floors followed by an examination of the paper proposing the problem: ”Nervi Puzzle: a topologically reconfigurable modular ribbed floor.” The data input for this problem will be described, along with the methods used thus far to find solutions. Important optimization and machine learning terminology will be summarized to facilitate understanding of the various methods and workflows. Different methods are assessed by looking at examples in literature. Limitations of the methods and workflows will also be discussed.

2.1. Topology Optimization of ribbed floors

Topology optimization is an advanced structural design method which can obtain the optimal configuration of a structure by distributing the material, satisfying specified load conditions, performance and constraints (37). New automated manufacturing methods allow the use of topology optimization for concrete structures, such as the elastic design of concrete beams (14). The state of the art research on ribbed floors has an focus on the topology optimization resulting in unique designs, assuming that economical and practical difficulties can be overcome in the future. Although topology optimization is not performed in this thesis it provides important background in the considerations to come to different module designs and it gives insight in the optimization objectives and constraints that can be used here as well. The topology optimization of a concrete ribbed floor is usually performed in multiple steps, due to the combination of concrete and the reinforcement, with the inclusion of serviceability requirements like crack widths leading to extra complexity (3).

In recent years a few custom made ribbed floors prototypes have been developed with the use of topology optimization and different fabrication methods.

The RMIT University, Melbourne developed a prototype column and slab based on Nervi’s Gatti Wool Factory and Palace of Labor. A topology optimization method using constraint mapping and complexity control was adapted to optimize the stiffness of the slabs (19). The number of ribs and cavities and the continuity of the ribs can be controlled, to result in feasable designs. Digital fabrication is applied by 3D printing polylactic acid and laser-cut acrylic sheets into a column formwork, combined with CNC machining plywood sheets for the flat ares of the slab formwork. The use of digital fabrication and the re-usability of the formwork aim to overcome the labor intensity of Nervi’s original designs.

Figure 2.1: PrintNervi column and slab (19)

ETH Zurich designed another reinforced concrete prototype using a three-dimensional printed plastic-based formwork (6). The method includes Eurocode (EC2) constraints for deflection, punching, deviation forces and fire resistance. A complete digital design-to-fabrication workflow is used, see Figure 2.2, creating a rib layout from principal bending moments, optimizing a 3D model using GH and FEM analysis in RFEM. It also includes the design of the reinforcement and the digital manufacturing of the formwork.

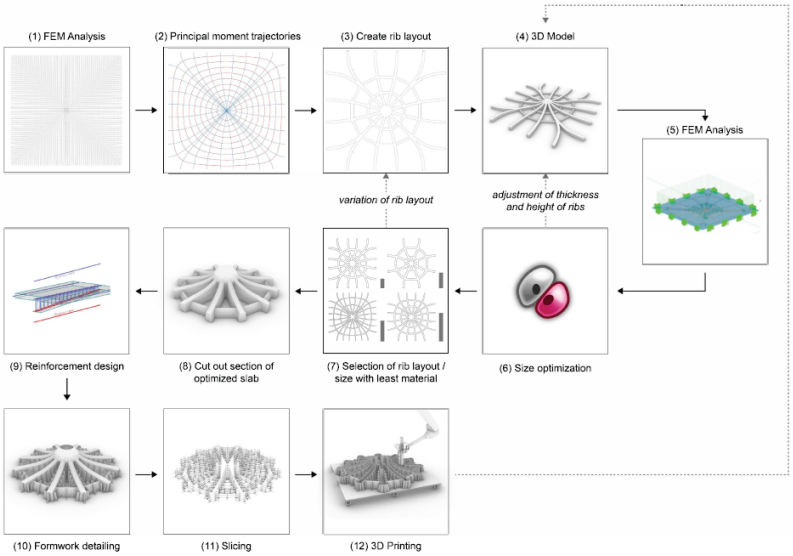


Figure 2.2: Design-to-fabrication workflow (6)

The Danish engineering firm Søren Jensen Designed a ribbed concrete floor based on isostatic lines for the transformation of a building in Copenhagen, Denmark as seen in Figure 2.3. The design is for two retail floors, with more office floors above following a design with a flat slab in combination with some extra columns and beams. The ribs placed based on the isostatic curves were divided in three categories of 550, 492 and 430 mm in height, allowing the slab thickness to decrease from 300 mm to 150 mm, necessary for the fireproofing but not for the structural capacity. The ribs are quite tall to minimize displacements, as was the wish of the client. This project focused on the aesthetic value and engineering heritage, accepting the high fabrication costs of the formwork, although digital fabrication was used to some degree. The structural calculation was based on FEM models and hand calculations and was simplified as much as possible, lacking an optimization step to further decrease material use.

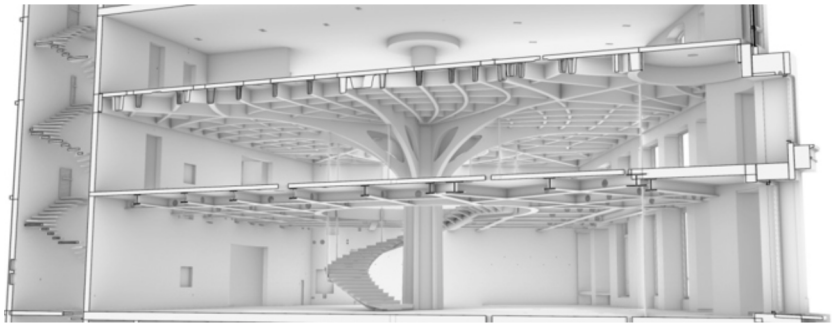


Figure 2.3: Design of ribbed floors by Søren Jensen (33)

2.2. Optimization Method requirements

As was shown in Figure 1.5 the different modules can be presented as bitmaps indicating the presence (1) or absence (0) of a pole. This gives every module a 4 bit code, one for every corner of the module. $2^4 = 16$ different combinations are possible and all are included in the catalog. A 4 module design for example can have as a design the configuration shown in Figure 2.4. Counted from the bottom left to the top right first increasing in x-direction and then in y-direction the bitmaps become: $[0101, 1010, 1010, 0101]$. Multiple options are possible to create an input from these bitmaps for the optimization methods.

One option is to create an array/tensor containing all the information as separate inputs so with the example that would become: $[0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1]$ Another option is to give the modules another index from 0-15 or 1-16, so that would result in: $[10, 9, 9, 10]$. This results in less dimensions and makes it also easier to include an index for an empty space in the floor plan. However, the information on the individual poles might be more difficult to extract for an algorithm and integer might prove difficult in training requiring normalization.

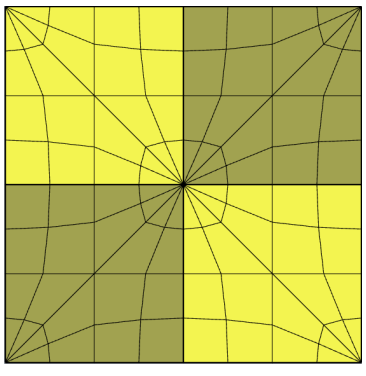


Figure 2.4: 2x2 Module example

To be suitable an optimization method should be able to implement:

- one of the possible discrete input types

- The ability to learn to predict a performance score for configurations
- The option to generate new configurations with a performance score

2.3. Heuristic Methods

A heuristic method typically employs simplified rules or guidelines derived from established domain knowledge. These methods can quickly derive solutions based on simple rules.

Heuristic: Support Conditions

The initial heuristic serves as a starting point based on the flow of forces towards column and wall supports. Ribs are aligned with columns and wall extremities to position poles at these locations. This method is automated, requiring only boundary conditions as input, without necessitating structural analysis. In the case study conducted (24), with minimizing elastic energy as the sole objective, it yielded a solution in under one second.

Heuristic: Stress Field

The stress field method performs initial structural analysis on a continuous plate with the same boundary conditions. First, the local stress field is computed, and for each pixel P, the module M that best fits the local stress field based on a scoring system is selected. This ad hoc fitness score is evaluated as a weighted sum of the average smallest angle between each edge and the local cross field to find the optimal module M_P. The vertices represent key points in the structural model for evaluating structural behavior, and the local cross field comprises the principal stress directions.

2.4. Discrete Numerical Optimization

The bitmap encoding of the problem renders the search for modular configurations suitable for discrete numerical optimization (24). This approach includes stochastic methods like evolutionary algorithms and simulated annealing. Key definitions include:

- **Generic Solvers:** Algorithms or software frameworks designed to solve a broad class of optimization problems without being tailored to a specific problem structure or type (27).
- **Phase Space:** A collection of all possible states of a given system.
- **Fitness Function:** Computes the desirability of a solution as a single numerical value.
- **Extrusion of a Phase Space:** Involves sampling solutions from a small minority of phase-space locations.

Surrogate models serve as approximate representations of complex or computationally expensive functions. They expedite the optimization process by providing a simpler, cheaper-to-evaluate alternative that captures essential features of the original function.

Simulated Annealing:

The simulated annealing algorithm is inspired by the formation of crystalline structures in cooling molten metal. The equations used can help locate peaks in the landscape by traversing it in decremental steps. If the next result is inferior, the algorithm reverts to the previous location. This method is adept at navigating rugged landscapes, initially identifying a valuable point and subsequently fine-tuning it.

Evolutionary Algorithms:

Evolutionary algorithms utilize biological principles of mutation, selection, and inheritance to improve solutions iteratively. The algorithm combines two solutions to discover a better one, proving effective at rapidly identifying reliable intermediate solutions.

Both the simulated annealing and the evolutionary algorithms are available for use in GH.

2.4.1. Limitations of generic optimization methods

While quick heuristic methods yield satisfactory results, they do not guarantee optimal solutions. Generic methods exhibit increased computation time when scaled to larger problems, potentially missing superior solutions identified through manual methods. In the following sections Deep Generative Models are introduced to overcome these limitations.

2.5. Generative Design

Due to the modular nature of this optimization problem, an analogy can be drawn with the floor plan design of modular buildings. Similar to structural optimization, such problems require the arrangement of predefined units into coherent and high-performing configurations.

For example, ModulePacking generates floor plans by applying a genetic algorithm to search over feasible layouts (16). In a related direction, Dai integrates ventilation, daylight, and traffic noise models into a Bayesian optimization framework for modular building layouts (7). Their parameterization of the floor plan includes up to 20 variables, such as the number of wings, apartment counts, and unit types.

Beyond algorithmic optimization, Mirra highlights the potential of AI models to emulate three human learning mechanisms in design: expertise, playfulness, and analogical reasoning (21). Expertise derives from precedents, playfulness fosters innovation, and analogical reasoning enables the transfer of knowledge across domains, stimulating creativity and discovery. Translating these mechanisms into computational models implies that generative systems should be able to:

- autonomously interpret knowledge from datasets,
- organize and structure knowledge into meaningful categories,
- communicate with users interactively,
- interpret partial or ambiguous information, and
- provide real-time design suggestions.

Regenwetter presented an overview of Deep Generative Models (DGMs) in engineering design (25). Figure 2.5 shows an overview of methods split up into different components of the process. The problem statement already results in the design task, which is the configuration of a set of modules. The representation method fits best in the category pixels, although the data does not exist of pixels, but is presented in a bitmap per pixel. The choice of generative model results in a lot of choices to be discussed in the rest of this literature review. The rest of the literature review mainly focuses on understanding and comparing Variational Autoencoders (VAE) and Reinforcement Learning (RL) as main methods.

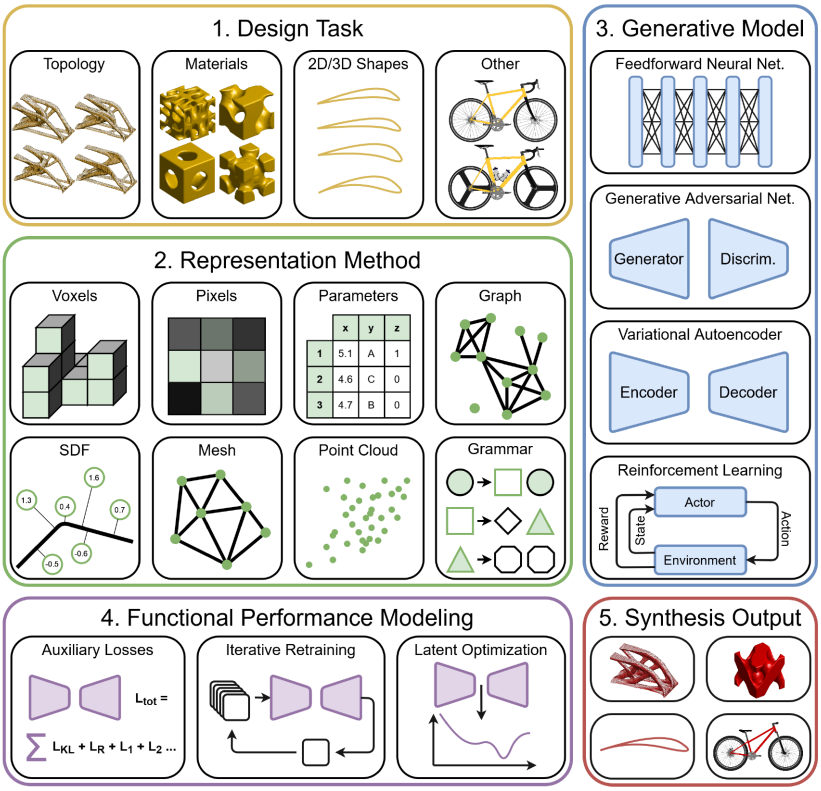


Figure 2.5: DGM methods overview, (25)

2.6. Bayesian Optimization

Bayesian optimization is a probabilistic model-based technique for finding the maximum or minimum of a black-box function that is expensive to evaluate. It is particularly advantageous in scenarios where function evaluations are costly, time-consuming, or noisy, such as hyperparameter tuning or engineering design. The key elements of Bayesian optimization are (28):

- **Surrogate Model:** A statistical model, typically a Gaussian process, that approximates the objective function, predicting outputs without direct evaluation.
- **Acquisition Function:** A criterion guiding the selection of new points to assess based on the predictive model, balancing exploration and exploitation.
- **Iterative Process:** The method involves iteratively updating the model with new data points (function evaluations) and using the acquisition function to determine the next sampling point until a stopping criterion is met.

Bayesian optimization is particularly valuable when dealing with high-dimensional spaces and functions lacking a closed-form expression, enabling efficient optimization with fewer evaluations.

An discrete form of this method could be used to predict a performance score that is calculated in Karamba3D for a set of initial samples. After the Gaussian Process is fit with these samples the surrogate model should be able to pick a new sample that is expected to yield the highest performance score.

An example of a discrete bayesian optimizer is Discrete-BO (17). The approach uses a Upper Confidence Bound acquisition function, also known as GP-UCB. The method works for discrete problems because it avoids sampling pre-existing observations by increasing the exploitation-exploration factor β and adjusting the length scale l. An algorithm is proposed to find β and l.

2.7. Reinforcement Learning (RL)

Reinforcement learning (RL) is a type of machine learning where an agent interacts with an environment and learns to make decisions by receiving rewards or penalties based on its actions (23). The agent explores different states and actions, aiming to maximize cumulative rewards over time through a trial-and-error process as schematized in Figure 2.6 It utilizes a policy to determine which actions to take in given states, continually updating this policy based on the feedback from the environment.

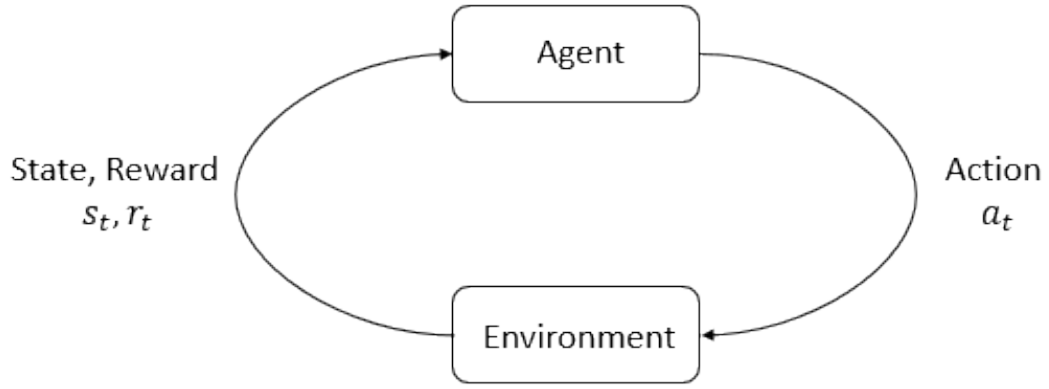


Figure 2.6: Agent-environment interaction loop (23)

Key concepts of RL are:

- A **state** s is a complete description of the state of the world. There is no information about the world which is hidden from the state. An **observation** o is a partial description of a state, which may omit information. When the agent is able to observe the complete state of the environment, we say that the environment is fully observed. When the agent can only see a partial observation, we say that the environment is partially observed.
- The **action space** is the set of all valid actions. Depending on the environment, this action space can be discrete or continuous.
- **Policies** are rules that guide agents in deciding what action to take. A policy can be deterministic or stochastic. In deep RL parameterized policies allow for optmization of hyperparameters. Categorial policies can be used in discrete action spaces and diagonal Gaussian policies in continuous action spaces.
  The Log-likelihood for an action a is $log\pi_\theta(a|s) = log[P_\theta(s)]_a$
- A **trajectory** τ is a sequence of states and actions in the world. Deterministic state transitions $s_{t+1} = f(s_t, a_t)$

- The reward function R depends on the current state of the world, the action just taken and the next state of the world: $r_t = R(s_t, a_t, s_{t+1}$ The agents wants to maximize some cumulative rewards over a trajectory. Two possibilities of a return function are: **finite-horizon undiscounted return** $R(\tau) = \sum_{t=0}^{T} r_t$ **infinite-horizon discounted return** $R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$
- **The RL problem**: the goal in RL is to select a policy which maximizes expected return when the agent acts according to it.

Value Functions Functions to obtain the expected return

1. **On-Policy Value Function**, $V^{\pi}(s)$: This function gives the expected return if you start in state s and always act according to policy π:

$$V^{\pi}(s) = \tau \sim \pi R(\tau) \mid s_0 = s$$

2. **On-Policy Action-Value Function**, $Q^{\pi}(s, a)$: This function gives the expected return if you start in state s, take an arbitrary action a (which may not have come from the policy), and then forever after act according to policy π:

$$Q^{\pi}(s, a) = \tau \sim \pi R(\tau) \mid s_0 = s, a_0 = a$$

3. **Optimal Value Function**, $V^{*}(s)$: This function gives the expected return if you start in state s and always act according to the optimal policy in the environment:

$$V^{*}(s) = \max_{\pi} \tau \sim \pi R(\tau) \mid s_0 = s$$

4. **Optimal Action-Value Function**, $Q^{*}(s, a)$: This function gives the expected return if you start in state s, take an arbitrary action a, and then forever after act according to the optimal policy in the environment:

$$Q^{*}(s, a) = \max_{\pi} \tau \sim \pi R(\tau) \mid s_0 = s, a_0 = a$$

2.8. Variational Autoencoders

The Variational Autoencoder (VAE) was first introduced in 2013 with the famous paper from Kingma and Welling (15). A Variational Autoencoder integrates deep learning and Bayesian machine learning to achieve a useful non-linear generative dimensionality reduction model. Within its encoder-decoder architecture, depicted in Figure 2.7, an encoder learns to map a sample x to a lower-dimension latent representation z, while a decoder reconverts z back to a reconstruction $\tilde{x}$. Both encoder and decoder undergo training until distortion measures are minimized. This bottleneck through z may lead to information loss, which can impede the generation of new data.

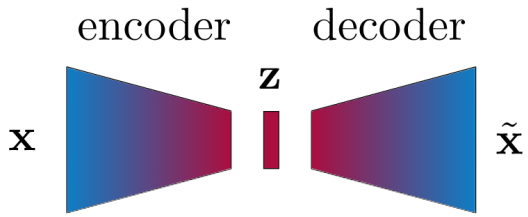


Figure 2.7: Encoder decoder architecture, (31)

Since the first introduction VAEs have been widely used, mainly for processing of image data resulting in models that can create new images. In the past few years it is more and more used to optimize structures. 2D or 3D geometry can be used to apply VAEs for topology optimization (22).

An example is the generation of shell structures (20).

2.8.1. Discrete VAE

VAE's are most often used for continuous problems. When the input data represents spatial features and performance metrics, this could involve multiple channels for the input data and both geometric and numerical outputs. When 2D or 3D representations of the structure are used, convolutional layers

could be needed to process the spatial patterns of the structure. After these convolutions the performance data can be integrated.

The discrete combinatorial representation of this problem simplifies the data representation significantly. As discussed before the modules are represented by 4-bits, instead of complex continuous geometric information. The encoder of the VAE no longer needs to extract complex spatial patterns using convolutional layers or large dense layers. Simpler models could potentially be used to map the 1D tensor into a latent space. The compact input data avoids the need of large tensors or high dimensional feature maps. This can lead to fewer parameters in the model, which makes it easier to train and less prone to overfitting, especially with smaller datasets. With this simple discrete representation of the structure, the model could be more focused on the combinatorial aspects of the design.

To handle discrete data with a VAE several approaches can be considered to include in the architecture. VQ-VAE (Vector Quantized VAE) or Discrete VAE can be used to quantize the latent space (32). Instead of using the typical MSE for the reconstruction loss, a categorical cross-entropy could be used, where each module is treated as a discrete class.

Using this simplified discrete representation could have significant advantages in the training process.

- Faster convergence: Reduced complexity of input data can lead to a model requiring fewer epochs to converge
- Easier Regularization: More predictable data could lead to easier regularization of the model
- Smaller latent space: The discrete data representation might be able to use a smaller latent space compared to a continuous or highly complex representation.

The simple VAE uses a combination of discrete and continuous input and output, the bitmap (discrete) and the structural performance (continuous). This could make the implementation of a discrete VAE like the VQ-VAE difficult. The question is if it is worth the extra effort for setting up a completely different architecture.

2.8.2. Proposed Architecture

Selecting an architecture for the VAE depends on the type of data and the complexity of the problem. The 1D tensor input could be processed using shallow dense layers or simple embedding layers, as oppossed to deep convolutional networks typically used for image data. The encoder could simply map the 4-bit values to a latent representation, and the decoder could map it back to the 1D tensor. When treating each 4-bit module as a discrete class, an embedding layer could be used to map the 4-bit representations to continuous vector spaces before encoding them into the latent space.

The latent space will capture the underlying design principles of how different modules can be arranged, and the decoder will learn to regenerate valid configurations of the modules. It is possible that the VAE generates (almost) correct bitmaps automatically and the numeric values only need to be clipped to 0 or 1. Otherwise, softmax or categorical cross-entropy can be used for reconstruction to ensure the output stays discrete.

The hypothesis is that a simplified discrete representation with a 1D tensor including the module bitmaps and performance metrics can be used to train a VAE using a simple VAE architecture with shallow dense layers and a small latent space. To test this hypothesis the following questions can be researched:

- Are shallow deep layers (1-2) sufficient to train the VAE?
- Is a small number of neurons (32, 64, or 128) sufficient?
- Is a small number of latent dimensions sufficient?

The encoder and decoder can be coupled using the architecture in Figure 2.8.

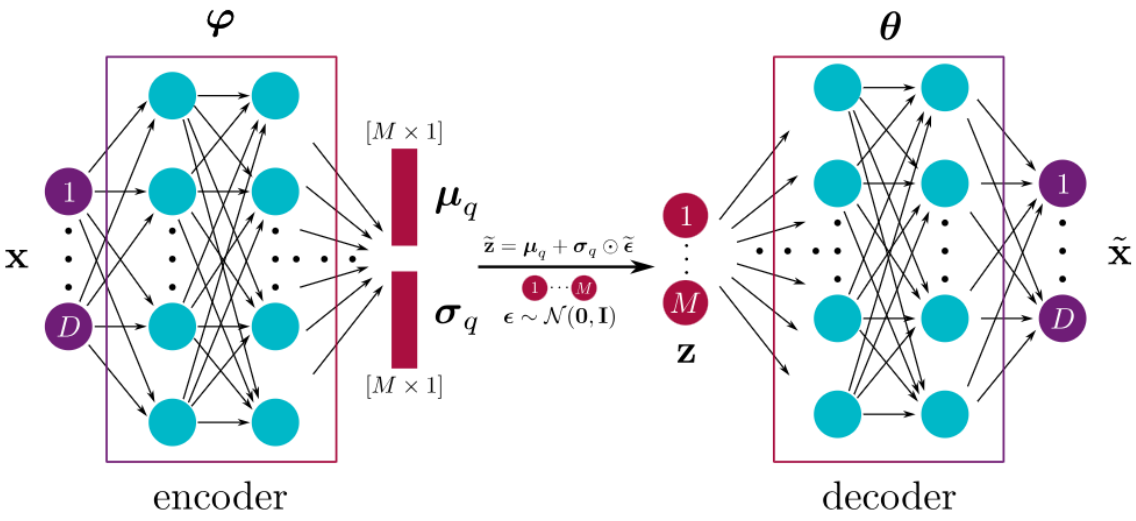


Figure 2.8: Encoder decoder architecture, (31)

The loss function is a trade off between reconstruction and regularization, with the left term being the reconstruction loss and end the right term being the KL divergence. The KL divergence term β is used to weigh this trade off, being another hyperparameter to tune.

$$Loss(X_n) = \| x_n - \tilde{x}_n \|^2 - \frac{\beta}{2} \sum^{m} (1 + \ln(\sigma_{nm}^2) + \mu_{nm}^2 + \sigma_{nm}^2) \tag{2.1}$$

To avoid breaking the backpropagation with autograd, a re-parametrization function is needed. This function performs a trick taking a sample ϵ from $\epsilon \sim \mathcal{N}(0, I)$, which lies outside the model. Using the hadamard product it is multiplied with the standard deviation, resulting in:

$$\tilde{z} = \mu_q + \sigma_q \odot \tilde{\epsilon} \tag{2.2}$$

With the encoder and decoder coupled they can be trained together and the model can be used to generate new samples as well.

2.8.3. workflow

VAEs have not been used in every sector because of data scarcity. The implementation of a way to generate data is a very important step. VAE workflows have been adapted for chemical design having the advantage of large datasets of molecules to train on (11). Mirra (20) used a Grasshopper script to create 800 3D shell structures inside a human defined design space. Zhang (36) used a latent space design crossover technique to create datasets of new designs using a generic optimizer. This enabled a large amount of the dataset to meet the target performance score. With the configuration problem of the modules the design space is already defined. A question to research is if using a generic algorithm could be beneficial for the creation of datasets.

2.8.4. Workflow example 1: Optimization of 3D Spatial Truss using VAE

In her thesis, Amy Sterrenberg developed a deep generative design framework to optimize a 3D spatial truss (29). Although this problem differs from the bitmap issue, it provides a strong foundation for the workflow and potential methodologies. The workflow consists of the following steps:

1. Creating an input dataset describing the spatial truss structure across various configurations: using a random number seed to generate 10,000 configurations.
2. Measuring the performance of different configurations, addressing:
   - Structural performance (Karamba)

- Material usage
- Similarity to stock library

3. Translating results into a single numerical performance indicator.
4. Employing the dataset to train a Variational Autoencoder (VAE) with a neural network architecture.
5. Generating a new data set based on VAE results.
6. Incorporating performance and geometry into a surrogate model (neural network) to predict structural performance from inputs.
7. Both the VAE and surrogate model can be utilized for backpropagation to identify superior solutions within newly generated geometries using a gradient descent optimizer. The surrogate model maps design geometries to performance indicators with a 80%-20% split.

The thesis concluded that a VAE model could be successfully trained to predict the performance of geometric configurations. It was also found that an evolutionary algorithm was more effective at creating an input dataset than a random approach, suggesting potential benefits in applying generic or heuristic methods to develop input datasets for subsequent processes.

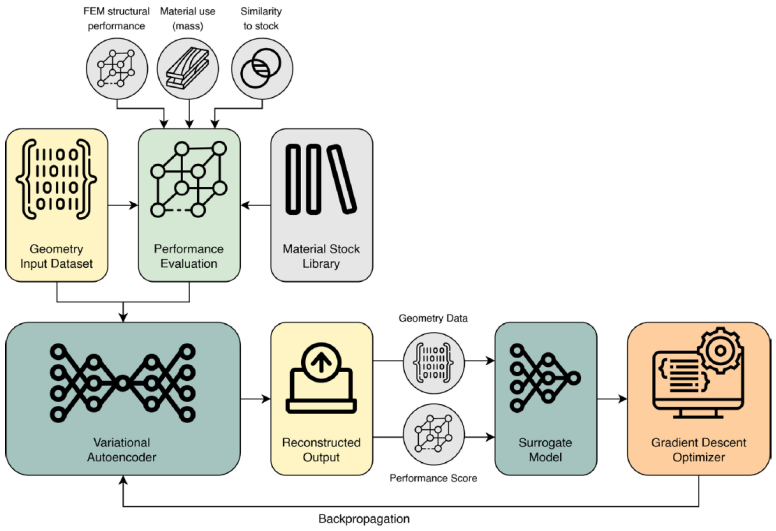


Figure 2.9: Workflow 3D spatial truss optimization (29)

2.8.5. Workflow example 2: Variational Auto-encoders and Bayesian Optimization

Joep Storm (30) developed a distinct workflow for the high-dimensional numerical optimization of fiber-reinforced polymers, utilizing a VAE to generate new samples, while the optimal solution was determined with Bayesian optimization. This is a hybrid approach which could be interesting to look into if it appears to be difficult to find the optimal solution after evaluating VAE samples. The workflow is depicted in Figure 2.10.

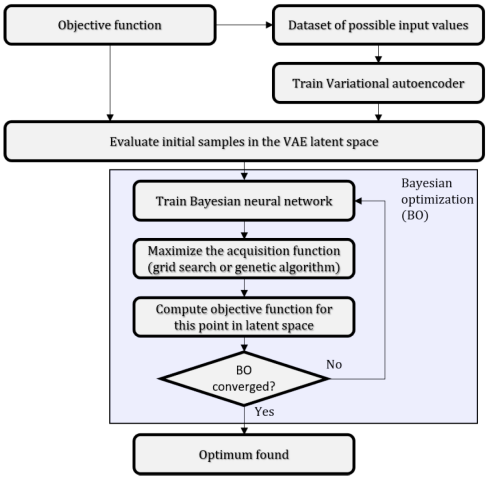


Figure 2.10: Workflow Bayesian Optimization (30)

2.9. Functional Performance Modeling

As depicted in Figure 2.5 after the generative model is in place there are three main options to perform the functional performance modeling in the DGM workflow.

With auxillary losses the loss function of the VAE is extended with an additional performance-based term. The optimization objectives and constraints can be included in this term. A latent space is learned where high performance designs are clustered. This method has the risk of destabilizing training if performance loss conflicts with reconstruction loss or KL-loss and might have difficulty in enforcing constraints.

Iterative training alternates between training the VAE and a surrogate model predicting the performance. The VAE is retrained to prioritize designs with better predicted performance, resulting from the surrogate model. This method requires a good surrogate and can be time consuming.

Latent optimization is performed directly in the latent space to find the best performing decoded output. In this method latent vectors z are drawn, decoded and evaluated with a performance function. Gradient descent, evolutionary strategies or bayesian optimization can be used to improve z. This method is direct, effective, flexible and performance focused. Possible risks are the dependency on a good trained VAE, high computational cost and the risk of drawing poor starting points.

Latent optimization is the most flexible allowing a performance function to be changed without the need of training the model again. The optimization algorithm can be run separately.

2.10. Conclusions: Literature Results

The current state-of the-art in optimization seems to lie on the interface of Bayesian machine learning and deep learning with the VAE and RL models as promising methods. Both share underlying principles related to handling uncertainty and exploration of complex spaces with a still tractable model. Both VAE and RL offer possibilities for discrete and generative models and the primary difference between the models lies in their underlying mechanisms and objectives:

- *Objective and Learning Paradigm:*
  RL: In RL, the focus is on learning a policy that maximizes cumulative rewards through a trial-and-error approach. The agent learns from the interactions with the environment, exploring various states and actions to improve its decision-making over time.
  VAE: VAEs are generative models that are used for data generation by learning a latent representation of the input data. They are trained through unsupervised learning, where the model learns to encode data into a lower-dimensional latent space and then decode it back to reconstruct the original data. The goal is to generate new samples that resemble the training data.
- *Exploration vs. Exploitation:*
  RL: An RL model balances exploration (trying new configurations) with exploitation (refining known good configurations) to discover optimal policies and solutions amidst potentially complex reward landscapes.
  VAE: A VAE generates samples based on the learned distribution of the input data, focusing on sample diversity and reconstruction accuracy without an explicit reward mechanism guiding it.
- *Handling constraints:*
  RL: In reinforcement learning, constraints can be integrated into the reward function, guiding the agent towards compliant designs while learning to optimize other objectives simultaneously.
  VAE: Constraints would typically need to be incorporated into the training process or the sampling method, potentially complicating the generation of valid configurations.

- *Output Nature:*
  RL: An RL agent yields outputs (configurations) based on an action-value structure, prioritizing those configurations that lead to better rewards.
  VAE: A VAE generates new samples based on learned distributions, which may not necessarily optimize any particular criterion unless explicitly conditioned.

Based on these four characteristics should be decided which of the two models fits the problem better. One option is two create a simplified model with both architectures to see which works better with a focus on the generative capablilites and the handling of the constraints.

Additionally a hybrid model of VAEs or RL with Bayesian Optimization can be considered. For

instance, VAEs can be employed to generate possible candidate solutions in the optimization space, and Bayesian Optimization can be used to identify which of these candidates should be evaluated in the real world, balancing the need for exploration of the generated space with the efficiency of Bayesian optimization.

Because the methodology of Reinforcement Learning and Variational Autoencoders are very different it was decided to continue with only one of the methods. The last characteristic gives the use of a VAE an advantage over RL. Due to the output nature of a VAE, the optimization takes place after the generation of new samples from a trained model. The constraints and objectives can be changed after the training process. This makes the VAE more flexible than RL, that needs to define the optimization objectives from the start to include in a reward function. Furthermore, prior knowledge of VAEs obtained during the TU Delft DSAIE course and a lack of knowledge on RL largely influenced this decision. A basic VAE was set up and experimented with during early stages of this thesis and the results supported the decision to continue with the VAE method.

2.11. Research Gap

Beyond assisting research into modular ribbed floor systems, this thesis aims to address a gap in the field of computational modeling. Currently, no previous workflows applied to similar combinatorial problems in structural engineering have been found. The main difference with the literature discussed is that it is now proposed to use a bitmap representation instead of the complete geometry of the structure as input for the VAE. This thesis will determine if and how good an optimization workflow works for a configuration problem with a bitmap input.

II

Methodology

3

Structural Model

This chapter covers how from a set of modules a structural model is set up including important assumptions and limitations. The metrics calculated in the structural analysis are explained together with the optimization objectives and constraints important for the design of ribbed floors.

3.1. Module Catalogue Design

The geometry of the modules begins with a wireframe structure of the ribs for each module as seen in Figure 3.1. In GH the individual lines of the modules can be extracted. The structural analysis is performed with the Karamba3D plugin for GH, by means of analyzing a set of beam elements. The lines extracted from the modules can be used to define beam elements.

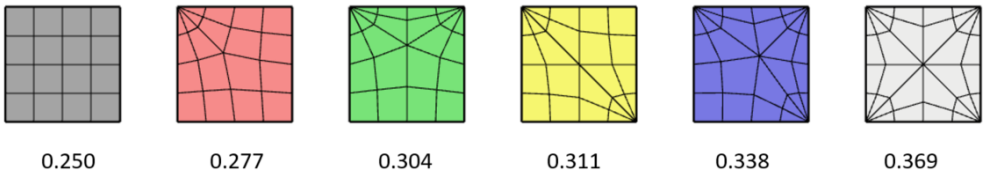


Figure 3.1: Module designs in Rhino8 with weight in ton/m^2 with a 20 cm by 10 cm cross section.

The rib depth and width must remain consistent at the interfaces of modules for physical compatibility, unless detailing of the connections could provide otherwise. During the optimization process, these dimensions could vary across all modules, or a constant interface value may be assigned while differing dimensions are used throughout the remaining modules. Additionally, material properties may be adjusted, as utilizing higher or lower quality concrete may influence the total embodied carbon of the floor modules. However, as the emphasis of this thesis is on the optimization of the configuration of the modules and not the module design itself, most parameters are kept fixed as much as possible.

The rib dimensions are set to an equal value for all modules. This way the normal ribbed floor is the lightest and modules with more poles become heavier. Minimizing weight will punish heavier designs. Modules with more corners will show less deflections and elastic energy so it is possible to look for a balance between these two conflicting objectives.

Alternatively, it was considered to tweak the rib dimensions per module to make all 6 modules equal weight. In this way, it is not possible to minimize the mass of the floor configuration as all modules have the same weight. Minimization of elastic energy and/or displacement can be performed to find an optimized design. This alternative was not chosen as an optimization of the cross-section dimensions

would be needed to reduce the embodied carbon in the structure.

3.1.1. Connections

As previously mentioned the simplification is made that the whole floor is assumed to be one system of ribs all completely rigid connected. A modular system would need connections between the modules that ensure it can be disassembled. Modular floors are very common, but usually they are connected in a permanent way or are simply supported on walls or beams. Demountable systems are still very novel, due to their expansive nature. The temporary courthouse in Amsterdam connects hollow core slabs with bolts to the widened bottom flange of steel beams, see Figure 3.2 (18). This connection enables to transfer horizontal forces and torsion as well. With this modular system multiple modules are needed to cross a single span so they also have to be connected to each other mid span. To make a similar connection for every side of every module would be complicated and expensive. A simpler connection might be favorable that does not allow for diaphragm action in the floor, but only transfers the shear force and moment. Further investigation into the connections is out of scope for this thesis

If the connections would be included in the model, that could be represented by adding hinges between the beams at the connections of two modules. In Karamba3D a moment connection can be made by adding a rotational spring stiffness. Depending on how stiff the connections are, mainly the displacement will be influenced. A very stiff connection will be closer to the completely fixed connection that is assumed, and a lower stiffness will increase the displacement as the hinges result in a kink in every module connection. It would only make sense to include the connections in the model if this stiffness is known, but it can be said for sure that the displacement will increase when adding multiple joints in one span.

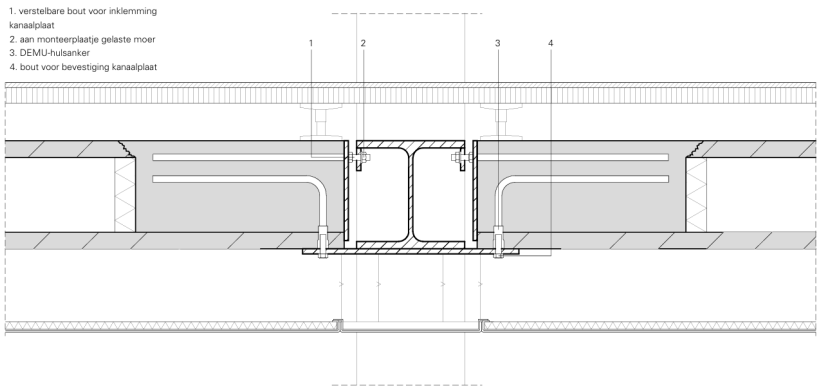


Figure 3.2: Demountable moment connection (18)

3.1.2. Multiple cross sections

When including multiple different cross sections in a configuration the simplification to model the modules as one continuous structure causes some problems. To have the floor at an equal level vertical eccentricities are needed for modules with smaller cross sections, see Figure 3.3.

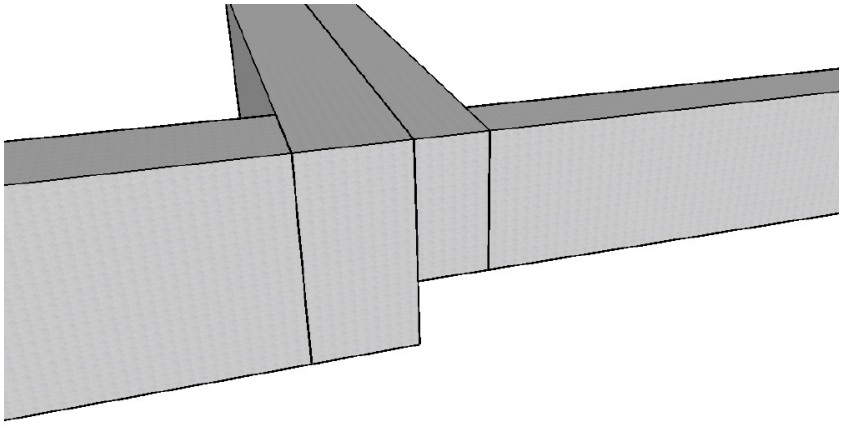


Figure 3.3: Drawing of compatibility issues with multiple cross sections

When two modules are modeled separately and connected, the beam in between the two is modeled twice, see Figure 3.4. For the beams in between two modules, one of the cross sections could be selected, or half of both could be modeled with an inward eccentricity. Other compatibility issue are the edge beams of the floor.

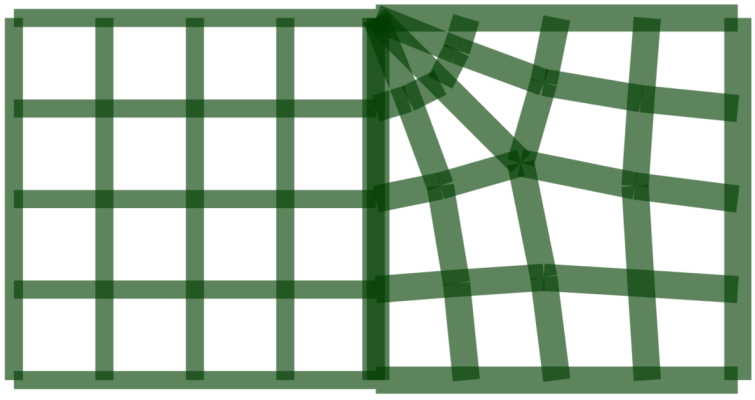


Figure 3.4: Overlapping beam between two cross sections

Different cross sections also cause vertical jumps in beams. One question is what the effect of the shear forces is at these interfaces. One more practical question is how to model the edge beams of the modules. It could be unpractical when there are vertical jumps and different thicknesses, especially at columns and walls. An option to overcome this is to keep the cross sections of the edge beams of the modules constant and only vary the inner beams as is done in Figure 3.5

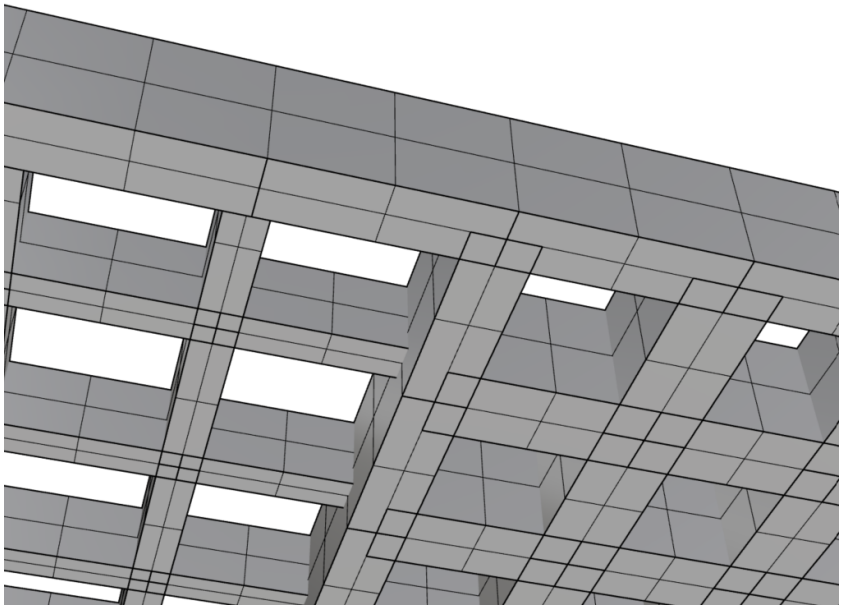


Figure 3.5: Jump in cross section with outer beam kept constant

3.1.3. Floor Layout

To start with a more basic structural model symmetric floor layouts are considered with the smallest one being 2 by 2 modules (2x2), being 4 by 4 meters, and the largest one being 5 by 5 modules (5x5) being 10 by 10 meters. The smallest designs might not always be the most efficient and the largest not always feasible in terms of the large span, but that is not always important in case of researching the scalability of the optimization method.

Furthermore, some simplifications are made to the modules that are important to consider to make a modular system work in practice. A configuration of a set of modules is now combined with the designs in Figure 3.1. From this wireframe duplicate lines at the interfaces are removed and a set of beam elements is created. This way a connection between two modules becomes one beam, which in reality would not be possible in a modular system. As a result the system is now modeled as a whole floor with all completely fixed connections.
Another simplification is that reinforcement is not implemented in Karamba3D. That causes problems for some calculations Karamba3D can perform such as utilization. Instead of including reinforcement directly, Eurocode checks for the ratio of reinforcement are performed. Including the reinforcement in the structural model would add another workflow to the optimization process adding a lot of computation time to the dataset generation process.

3.2. Structural Model

For every configuration the lines are collected, duplicates removed and the beam elements are created and can be used for the model assembly. To all beams the same cross section is assigned with the beam height, width and material. In the model assembly, the supports and loads need to be defined as well.

3.2.1. Supports

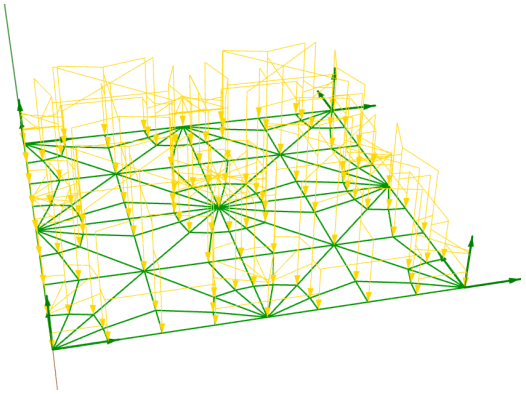
The structure is assumed to be on infinite stiff simple supports on the four corners. All translational degrees of freedom are fixed and all rotational degrees of freedom are free.

Another option would be to make the supports translational and rotational fixed. In the 4 modules grid example this would greatly reduce the displacement from 10 to 4.4 mm. The shear force would stay the same and the moment would shift lowering the moment at mid span and introducing hogging moments at the supports. Introducing these hogging moments would require a different kind of reinforcement at the connections. One could decide to use completely fixed supports to lower the displacement as this often is the limiting factor. In this case it was decided to use the simple supports as

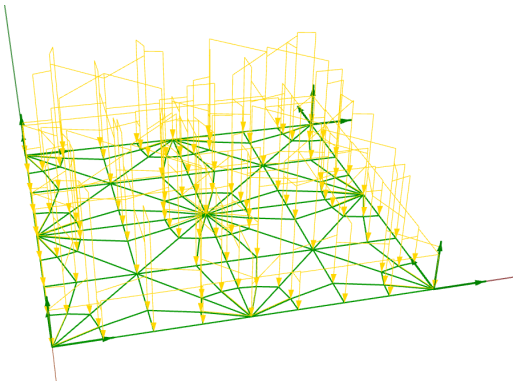
this is assumed easier and more workable to achieve in practice.

3.2.2. Loads

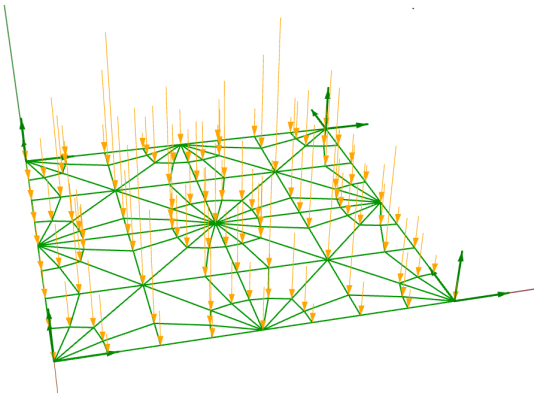
In Karamaba3D it is possible to define different loads and load cases. The self weight is considered as a gravity load G and a uniform load as Q. To project the uniform load on the rib structure it is possible to mesh the floor area where the load acts on. Karamba3D automatically distributes the load over the beam elements. It is possible to generate the loads on the beams as either point loads or line loads, see Figure 3.6. For the regular grid the distributed load will be evenly distributed over the ribs for a mesh resolution smaller than 0.25 meters. For modules with more poles the mesh resolution and the choice for point or line loads does have an effect on the distribution of loads, but it has a very small effect on the maximum stresses and displacements, so a mesh resolution of 0.25 meter is preferred. Line loads are used as that is seen as a more realistic representation.



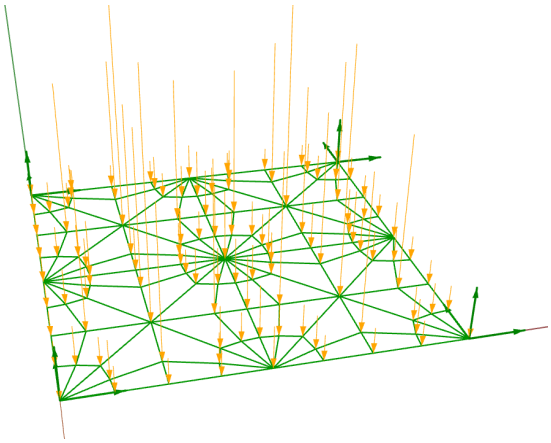
(a) Line loads with 0.1 meter mesh resolution (own work)



(b) Line loads with 0.25 meter mesh resolution (own work)



(c) Point Loads with 0.25 meter mesh resolution (own work)



(d) Point Loads with 0.1 meter mesh resolution (own work)

Figure 3.6: Projection of loads with different mesh resolutions

The load combinations considered are ULS and SLS with self weight G and a distributed load Q. The ULS and SLS Load combinations are.

$$ULS = 1.35G + 1.5Q \tag{3.1}$$

$$SLS = 1.0G + 1.0Q \tag{3.2}$$

3.2.3. Analyzing the model

After the model assembly the analyze component can compute the mechanical response for each load case. For both load cases the model returns the maximum nodal displacement and the structures

internal deformation energy or elastic energy. The maximum displacement is stored for the SLS load case and the elastic energy for the ULS load case. To keep computation time to a minimum only one more calculation is added. The beam forces component returns the forces at both ends of a beam element for a specific load case. The maximum shear force and moment are stored for the ULS load case. Together with the mass, the displacement and the elastic energy, a total of 5 performance metrics are stored every time the model computes a module configuration.

3.2.4. Verification of the structural model

A first check if the model is functioning properly is to visually look at the displaced result after running the model analysis.

Figure 3.7 shows that all beams are correctly connected at all the joints as there are no discontinuities in the displacements. The displacement field is also what is expected from a symmetric simply supported slab. Although easier to see moving around in the Rhino viewport, the displacement is zero at the supports and the beams can rotate as the rotational degrees of freedom are free. The floor follows a parabolic displacement towards the middle in both directions. Giving the outer beams an eccentricity inwards results in a larger maximum displacement (about 2%) and slightly higher forces as well, but has no large effect on the model. In the final model the eccentricities are ignored as the effect is small and not important when methods are compared with the same structural model.

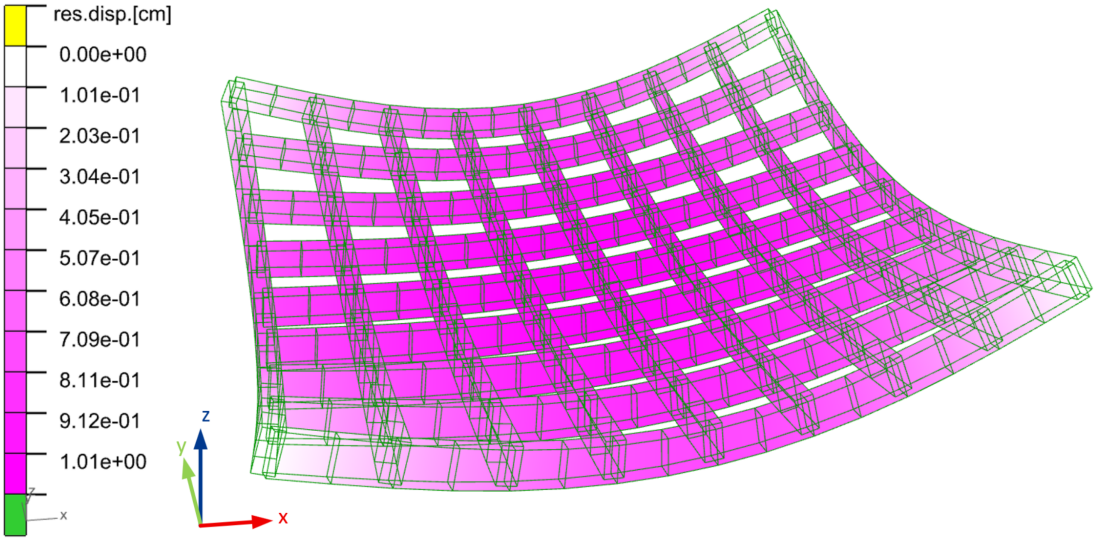


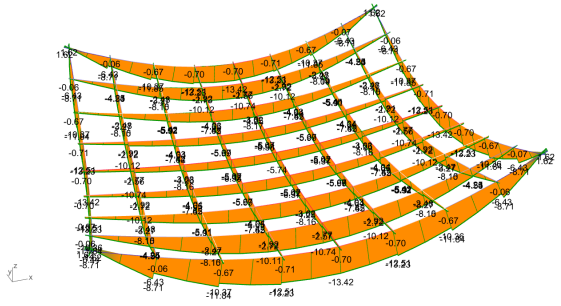
Figure 3.7: Displacement to check if all beams are connected

The Karamba3D beamview element also allows to see the forces acting on every element. Considering the ULS load combination with self weight and a distributed load of $5\ kN/m^2$ the moments and shear forces are shown in Figures D.1 and D.4. The model view element gives the support reactions being $42.15\ kN$ for every support, matching the V_z in Figure D.4 and resulting in a total of $168.6\ kN$. With Q being $16m^2 * 5kN/m^2 = 80\ kN$ and G being $3600kg * 0.01 = 35.316\ kN$ that results in $1.35 * 35.316kN\ + 1.5 * 80kN = 168.6kN$, using $g = 10\ m/s^2$.

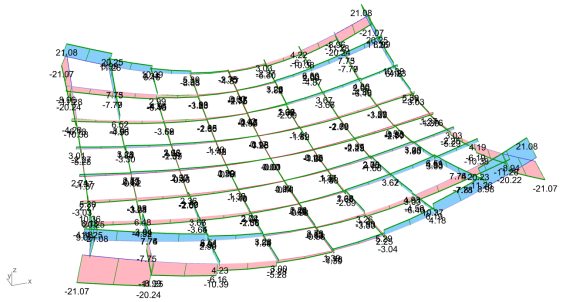
The same grid structure is modeled in RFEM for further verification. Considering the same loading conditions the maximum $M_y = 13.35\ kNm$ and the maximum $V_z = 21.08\ kN$, see Figures D.3 and 3.9b. The displacement of $9.9mm$ also closely matches the displacement found in Karamba3D with $10.0mm$.

3.2.5. Limitations of the structural model

The structural model using Karamba3D in GH for the structural analysis is limited to shell and beam elements. Although it is possible to assemble a model representing a ribbed floor using concrete beam elements, it is not possible to include reinforcement in the concrete structure. Due to the very limited capacity of concrete without reinforcement, it is difficult to analyze the structural behavior. Large tensional forces and large shear forces can cause the model to show extremely high utilization values,

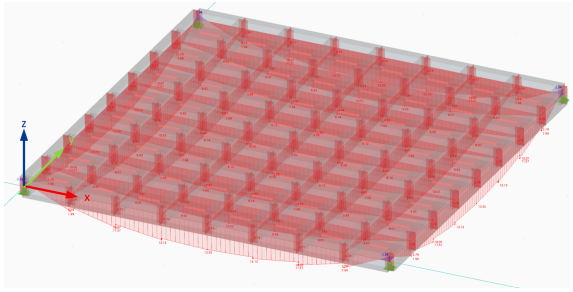


(a) M_y for ULS with 5 kN/m^2 distributed load

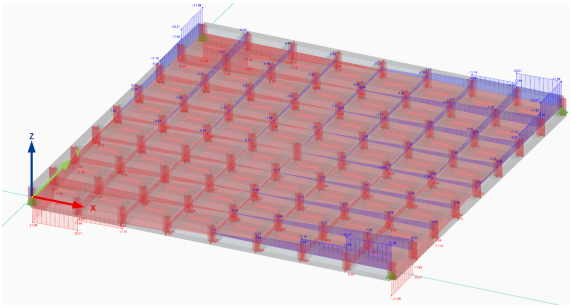


(b) V_z for ULS with 5 kN/m^2 distributed load

Figure 3.8: Forces with ULS loading



(a) M_y for ULS with 5 kN/m^2 distributed load



(b) V_z for ULS with 5 kN/m^2 distributed load

Figure 3.9: Forces with ULS loading, see Appendix D for larger size figures

making utilization tools in GH not very useful for concrete structures.

For a more sophisticated structural analysis of ribbed floor-systems in other research (6) the software Dlubal RFEM is used. This software gives more possibilities to model the structure in more detail with reinforcement and perform code checks. However, the method in this thesis requires to run a structural analysis a numerous amount of times, (probably 10,000 - 60,000 times) thus to create datasets it is necessary to do this in the GH environment with a minimal computation time and direct export of results.

To overcome this issue, the datasets are created storing the mass, displacement, and elastic energy and the maximum values of the moment and shear force. Code checks can be done outside of the dataset generation decreasing the computation time during this step. Two options arise:

1. Train the VAE to predict all the structural data
2. Compute performance scores with the datasets generated and train the VAE to predict this numeric score

Both options can be done with very simple calculations in python, which should make the computation time minimal.

3.3. Optimization Objectives

The design of concrete structures can be distinguished in two categories, performance-based and code-based design (4). Concrete performs good under compression and bad under tension, resulting in the need for reinforcement. The cracking of concrete and the yielding of steel makes the behavior of concrete highly non-linear and makes the use of a simple linear-elastic analysis difficult. Linear elastic modeling can still be used for preliminary design (2) to calculate certain performance based measures. According to article 5.4 from NEN-EN 1992-1-1+C2+A1 linear elastic analysis can be used to calculate dimensions of elements for ULS and SLS conditions.

However, to asses if a structure is safe code compliance is needed. Code compliance can be integrated in parametric design tools to make code-based designs. Witheley (35) performed a code-based grillage optimization of a ribbed floor using the internal bending moments and cross-sectional

areas of beams as constraints, minimizing the volume. In this case the cross-sections are already set but similar code-based constraints can be used to perform code-checks and assign scores to the results.

This section explores which performance based objectives and code-based constraints can be used to asses the structure. In the next chapter these measures are combined into a performance score.

3.3.1. Performance based objectives

In structural optimization common objectives are minimization of mass, elastic energy or compliance.

Elastic energy is the energy stored in a structure due to deformation under applied loads.

$$U = \frac{1}{2}\int \sigma : \epsilon d \tag{3.3}$$

Compliance is a measure of how much a structure deforms under a given load. It is defined as the work done by the external forces on the structure, which is equal to twice the elastic energy for linear elastic materials

$$C = F^T u \tag{3.4}$$

In many structural optimization problems, minimizing compliance is a common objective because it leads to stiffer structures. Elastic energy is proportional to compliance $C = 2U$ so minimizing one will also minimize the other.

Both mass or compliance or elastic energy need certain constraints when minimizing. Without correctly defined constraints minimizing mass will lead to very light structures that cannot fulfill their function. Minimizing for compliance can lead to very heavy structures that are extremely stiff. This is a inverse relation that cannot be simultaneously minimized both, resulting in a trade-off.

The goal of reducing embedded carbon in this modular and circular floor-system makes it logical to minimize the mass in the optimization process and not stiffness. Constraints to the stiffness can be set including constraints on the displacement. The workflow allows for minimization of either mass or elastic energy with a combination of constraints.

3.3.2. Code based constraints

The reinforcement makes concrete highly nonlinear when cracking. To ensure structural safety and comply with serviceability criteria some code based constraints can be included in the design process. These measures do not necessarily need to be minimized, but do need to stay below a certain threshold to allow for a design to be valid. To explore the code based constraints a 20 cm by 10 cm cross section with concrete class C30/37 is used with a 2 by 2 module floor plan.

Moments

According to article 6.1(9) of EC2 the maximum reinforcement ratio to prevent brittle failure of the concrete is $\rho_{1max} = 1.85\%$. The amount of reinforcement required should stay below this maximum. With the moment and cross section known this is a simple check to see if the required reinforcement is reasonable.

$$\frac{M_{Ed}}{f_{yd}0.9d} < \rho_{1max} bd \tag{3.5}$$

Assuming $d = h - c = 200 - 30 = 170mm$, $A_{s,max} = \rho_{1max} bd = 0.0185 * 100 * 170 = 314.5mm^2$. Which is about one 20 mm diameter strand that should easily fit inside the cross section.

With all parameters known we can also see that M_{Ed} should stay below $f_{yd}0.9d\rho_{1max}bd = 435 * 0.9 * 170 * 0.0185 * 100 * 170 = 20.93\ kNm$

The moment at which cracking occurs is assumed to be the moment where the tensile strength of the concrete is reached.

$$M_{cr} = \frac{f_{ct} b h^2}{2} = \frac{2.9 * 100 * 200^2}{2} = 5.8 kNm \tag{3.6}$$

After cracking, the concrete in tension is not as effective, and the beam's stiffness is dominated by the steel reinforcement and the concrete in compression. After cracking, the bending moment is redistributed, and the stiffness reflects the contribution of reinforcement and compression zone, resulting in a reduced stiffness.

Shear Force

The shear stress for which shear reinforcement is needed is

$$v_{Rd,c} = v_{min} = 0.035 * k^{3/2} * \sqrt{f_{ck}} = 0.035 * 2.0^{3/2} * \sqrt{30} = 0.54\ N/mm^2 \tag{3.7}$$

$$V_{Rd.c} = v_{min} b d = 0.54 * 100 * 170 = 9180N = 9.2kN \tag{3.8}$$

A simple check if shear reinforcement is needed is then

$$V_{Ed} < V_{Rd,c} \tag{3.9}$$

The higher the shear force above $V_{Rd,c}$ the more shear reinforcement is needed with a maximum of $V_{Rd,max} = v_{Rd,max} b d = 5.42 * 100 * 170 = 92.14\ kN$

Displacements

Displacement is often a governing measure in designing floors and beams. The following definitions of maximum displacement were used to analyze and score a module configuration.

Under SLS conditions according to A1.4.3 from NEN-EN 1990+A1+A1/C2/NB floors carrying walls vulnerable to cracking the deflection should stay below:

$$w_{max} = l/500 = 0.004 * 2 * 2000 = 8\ mm \tag{3.10}$$

And for other floors and roofs for aesthetics the deflection should stay below :

$$w_{max} = l/250 = 0.004 * 2 * 2000 = 16\ mm \tag{3.11}$$

The structural analysis results from the configurations in Figure 3.10 are compared in table 7.9. The first and last designs are the ones with the lowest and highest mass. The second design has poles at the columns and the third one also has a pole in the middle. The two other designs are randomized.

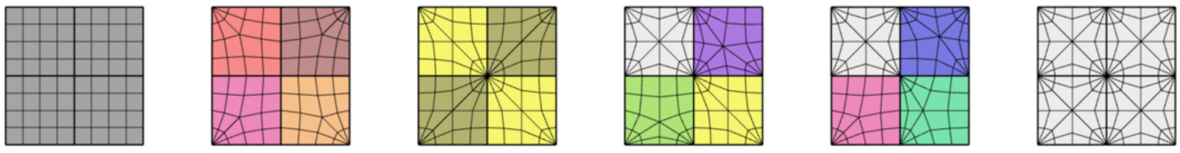


Figure 3.10: 2x2 Module samples to compare

Table 3.1: Exploration of different design constraints

Modules	Mass $[ton/m^2]$	$My_{Ed}[kNm]$	$Vz_{Ed}[kN]$	d [mm]
Grid	0.225	13.34	21.07	10.13
4 Corners	0.252	11.33	9.71	9.50
8 Corners	0.286	10.97	11.10	8.36
Random 1	0.306	15.88	21.67	8.95
Random 2	0.297	17.36	21.78	9.05
Heaviest	0.344	12.10	12.74	8.30

From Table 7.9 it can be seen that My_{Ed} gets close but stays underneath the maximum value My_{Rd}. Vz_{Ed} however, stays far below Vz_{Rd}, but is 1-3 times larger than $Vz_{Rd,c}$. A lower value of Vz_{Ed} means less shear reinforcement is needed, which would lead to less material use. So designs with Vz_{Ed} higher above $Vz_{Rd,c}$ could be punished. The shear seems to be divided in two categories and does not seem to follow one distribution.

The designs that were expected to perform better indeed have lower moments and shear forces, which occur at positions were they are to be expected. Maximum shear force at the supports and maximum moment at the midspan. The not symmetric randomized designs have the highest moments and shear forces. The forces are not spread equally to the supports leading to peaks in shear forces and moments.

Design against punching, fire and deviation effects of curved reinforcement are important in ribbed floors design (6), bit might prove difficult to implement only using Karamba3D.

Punching failure: right now a slab is excluded from the model but the connection between beams and the column can still be critical because of high shear forces causing punching/shear failure. Ways to prevent punching failure are to apply mushroom columnheads or designing a smooth transition between the columns and ribs. The fact that the modular elements are not designed to all have columns at all four corners, makes it difficult to implement a smooth transition in the design. Some kind of column head could of course be possible assuming a connection could be made to the modules that allows for disassembly. For this thesis it is assumed that a column module connection can be made that ensures punching shear at the columns is not governing.

Curved ribs and moments: All modules are designed with ribs consisting of straight elements with kinks at most of the connections between different ribs. Ignoring the reinforcement or assuming the reinforcement is also kinked and not bend, there is no deviation force due to curved ribs in between two connections.

Fire safety: Eurocode 2 REI60 fire resistance class gives two options for enough concrete cover to protect the reinforcement. $b_{min} = 120$ mm with minimum bar axis distance $a_{min} = 25$ mm or $b_{min} = 100$ mm with minimum bar axis distance $a_{min} = 35$ mm

3.4. Selecting a cross-section

In the GH script the different concrete classes can be selected as material and C30/37 is selected. From the material properties and the cross-section some estimations can be made for the maximum moments and shear forces this cross-section can take up. These maxima are based on rules for maximum reinforcement in relation to the rib dimensions from EC2.

$$V_{Rd,max} = v_{Rd,max} bd = 5.42 * bd \tag{3.12}$$

$$V_{Rd,c} = v_{min} bd = 0.54 * bd \tag{3.13}$$

$$M_{cr} = \frac{f_{ct} bh^2}{2} \tag{3.14}$$

$$M_{Rd} = f_{yd} 0.9 d \rho_{1,max} bd \tag{3.15}$$

The following cross-sections were considered:

Table 3.2: Different cross sections and their resistance

Cross-section	height [cm]	width [cm]	$M_{Rd}[kNm]$	$V_{Rd,c}[kN]$	$V_{Rd,max}[kN]$
20cm x 10cm	20	10	27.8	9.2	92.1
30cm x 15cm	30	15	105.3	19.7	219.5
40cm x 20cm	40	20	263.7	31.4	401

To select one of the cross-sections above the structural analysis is performed for all three cross-section options with only the normal grid module increasing from the 2x2 to the 5x5 problem. For the 20 cm by 10 cm cross section the larger floor plans were not considered due to the limited moment capacity.

Table 3.3: Exploration of different cross-sections, 20cm x 10cm

20x10	$M_{Ed}[kNm]$	$V_{Ed}[kN]$	mass $[ton/m^2]$	d [mm]	elastic energy [kNm]
2x2	13.3	21.1	0.22	10	0.89
3x3	32.6	46.9	0.22	55	11

Table 3.4: Exploration of different cross-sections, 30cm x 15cm

30x15	$M_{Ed}[kNm]$	$V_{Ed}[kN]$	mass [ton/m^2]	d [mm]	elastic energy [kNm]
2x2	18.0	28.7	0.51	3	0.33
3x3	43.8	63.4	0.49	15	3.9
4x4	81.2	111.6	0.48	49	23
5x5	130	173	0.47	121	90
max	105.3	220	-	nx4	-

Table 3.5: Exploration of different cross-sections, 40cm x 20cm

40x20	$M_{Ed}[kNm]$	$V_{Ed}[kN]$	mass $[ton/m^2]$	d [mm]	elastic energy [kNm]
2x2	25	39	0.90	1	.2
3x3	59	86	0.87	7	2.3
4x4	110	151	0.85	21	13
5x5	176	236	0.84	53	52
max	264	401	-	nx4	-

The 30cm x 15cm cross sections was selected as most interesting to use because the maximum moment resistance and maximum allowed displacement are surpassed somewhere half way when increasing the floorsize from 2x2 to 5x5 modules. This means that for the smaller problems the minimization of weight should be quite simple, but when increasing the problem size the constraints take a larger role and only grid modules will not be sufficient.

4

Optimization Workflow using VAE

In the first section of this chapter the optimization workflow is proposed. An overview is given how this workflow is developed and how it will be implemented in the following chapters. In the other sections the VAE architecture and the Gradient descent algorithm and workflow are discussed.

4.1. General Workflow

The optimization workflow takes the following steps:

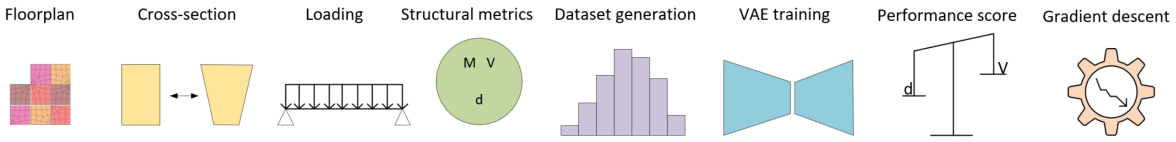


Figure 4.1: Workflow

1. Define a floor plan
2. Choose the modules and their parameters
3. Choose a structural model and load cases
4. Select metrics to calculate structural behavior
5. Select a dataset generation method and calculate the structural behavior
6. Perform model selection / hyperparameter tuning on the VAE and train the model to predict structural behavior
7. Select a performance score to optimize the design for
8. Generate new samples with the VAE and optimize with gradient descent on the selected performance score

The general setup of the structural model and the possible optimization objectives and constraints and how a performance score can be calculated, were discussed in the previous chapters. The architecture of the VAE is explained in this chapter. The following sections explain different considerations in the optimization workflow and how they lead to a number of research questions. These question lead to numerical experiments with results in the next chapters

4.1.1. Dataset Generation and the effects of problem size and data quality

To train the VAE model a dataset is needed. Datasets are created by storing the bits for every corner point, a pole (1) or not (0) in one tensor together with some kind of performance. What this performance is and how to store it in the datasets depends on the rest of the workflow.

Two different approaches for training the VAE were considered:

1. Create one single numeric performance score to train the VAE. This approach was tried first and showed to work well. The downside of this approach is that the structural knowledge is lost in the model and the results are completely depended on the definition of the performance score.

2. Train the VAE on all structural performance data collected. The VAE could learn to predict the actual structural performance in the measures shown above. With generated samples from the VAE different optimizations can be performed without training the VAE again.

The first option results in a tensor with one numeric score at the end, and the second option with multiple values.

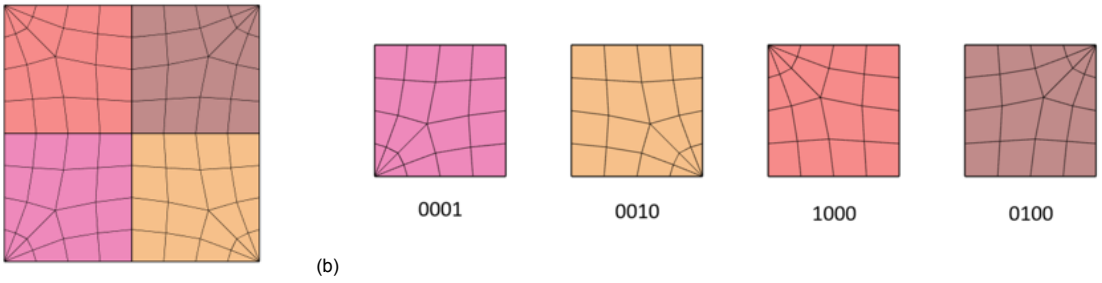
For both approaches the questions arises how to create the datasets. The quality of the predictions of the VAE will be dependent on the size and quality of the datasets. For the first approach it is easy to create datasets of ”good” samples, that have a high performance score. This could be done by using generic algorithms to create datasets. This is done in chapter 6 to test if the VAE can generate better configurations than the generic algorithms.

When training to predict all structural data it is not clear what are good samples as it is not yet defined what good performance is. The effect of different dataset creation strategies, the size of the datasets, and the problem size on the prediction quality is researched in chapter 7.

To test the effect on the performance of the VAE four different symmetric floor plans ranging from 2x2 modules up to 5x5 modules are considered. All use the same structural model with columns at the corners and the same loading and cross-sections. The number of possible configurations increase significantly from $16^4 = 10^5$ for the 2x2 problem up to $16^{25} = 10^{30}$ for 5x5 modules.

The 1D tensor is created by first taking the bitmaps of the modules increasing in x-direction and then in y-direction, as depicted in Figure 4.2. The performance score or in this case multiple performance metrics are normalized and added to the tensor. The resulting tensor is:

$$Tensor : [0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0.466, 0.234, 0.876, 0.564, 0.765] \tag{4.1}$$



(a) (b)

Figure 4.2: Order of modules in 1D tensor

4.2. VAE Architecture

From the literature review the hypothesis arose that a simplified discrete representation with a 1D tensor, including the module bitmaps and performance metrics, can be used to train a VAE using shallow dense layers and a small latent space. This approach was tested first and is explained in the next section. Because the VAE architecture in this approach was not designed for discrete problems, alternatives were considered as well. Suggestions from the literature review are to use a VQ-VAE to quantize the latent space and to use a categorial cross-entropy for the reconstruction loss to further increase the performance of the VAE.

4.2.1. Shallow VAE

Figure 4.3 shows the architecture of this VAE. An example input is given with the input dimensions D being equal to 4 times the number of modules plus the number of performance metrics collected. Because the VAE does not need to extract complex spatial patterns, no convolutional layers are needed. The architecture is shallow because it only uses a few hidden layers and in this case only one.

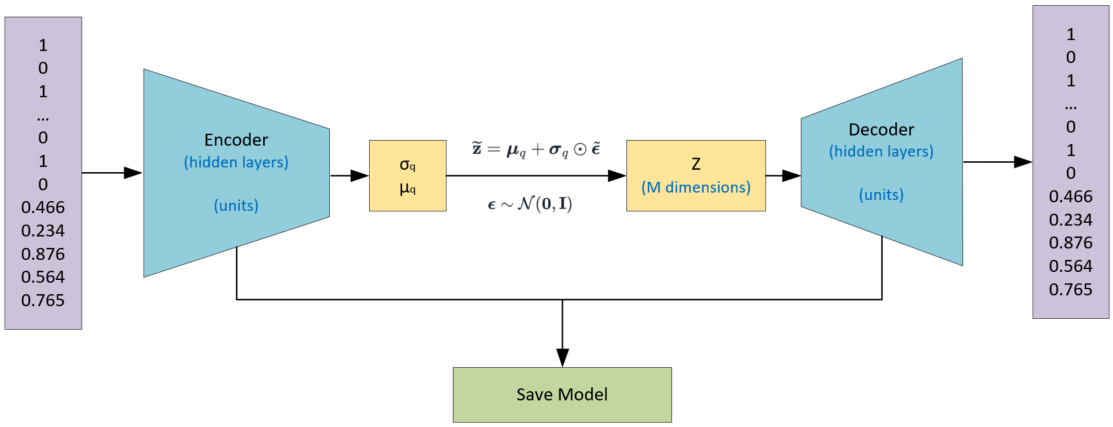


Figure 4.3: VAE architecture

Functions

The used VAE Architecture is based on an example from the DSAIE course used for generating microstructures. PyTorch is used as main package for the VAE. The VAE is initialized in a class with the following functions:

- `init`: initializes the layers that make up the two halves of the autoencoder. This function receives the hyperparamters.
- `encode`: Encodes data from real space x to a Gaussian approximation $q(z)$ of $p(z|x)$
- `decode`: Decodes data from latent space z to real space x
- `reparameterize`: Performs the reparametrization trick
- `forward`: Performs a complete forward pass through the autoencoder
- `generate`: Samples from the prior $p(z)$ and uses the decoder to generate new data
- `loss function`: Computes the loss function of the VAE
- `performance function`: Allows to choose a performance function and the weights
- `generate_withoptimizer`: Samples new data with the decoder and uses the performance function and gradient descent to optimize the performance score.

4.2.2. Architecture and hyperparameters

This VAE architecture is set up with adaptable values for the input dimensions, latent dimensions, hidden layers and hidden units. The defined architecture makes the model symmetric, but this could be changed. Activation functions for the hidden layers can be changed in the definition of the VAE itself and are set to SELU activation by standard. The train one epoch function is defined to train the model for a full epoch. The model can be initialized by defining the number latent dimensions, hidden layers and hidden units. The input dimensions are given by the input dataset. In the model initialization the number of epochs and the optimizer can also be chosen, which is by standard set to the Adam optimizer.

This VAE is tested on the simple 4 modules case in chapter 6. This smaller problem allows to test and compare methods more easily, because solutions for the problem can be found with brute force, i.e. computing all possible solutions. From this case the problem can be extended in more complex cases for which the VAE should give better and quicker results than the previously discussed heuristic and generic methods.

4.2.3. Training

During the training of the VAE the workflow as seen in figure 4.3 is used. In the VAE class a model is initialized by defining the number of latent dimensions and hidden units and beta, the weighing for the KL divergence loss term. During the training process the model with the lowest validation loss is saved, with the goal to have a resulting model that can predict the structural behavior of a random bitmap sample as good as possible. For every dataset the training process has to be done separately.

4.3. Gradient Descent in latent space

To find an optimized solution when generating new samples the workflow in figure 4.4 is followed. A model trained for a specific dataset is loaded. The gradient descent function samples from the latent space and after decoding and selecting a performance function, optimizes for this function. The resulting predictions can then be denormalized to the real metrics, so they can be compared with a GH calculation.

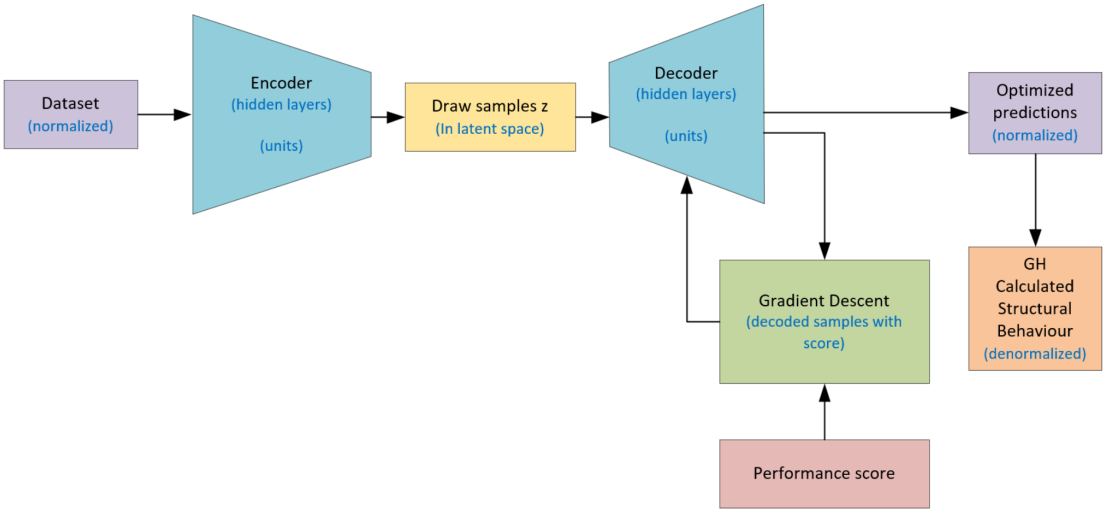


Figure 4.4: VAE workflow with gradient descent **check if arrows are correct**

The `performance_function` takes a decoded sample and the mode that should be used, defining a specific performance function from a number of options. With the performance metrics different combinations of scores can be defined. To combine multiple metrics into one score, the metrics are kept normalized during the calculation of the score. From all the defined performance functions one can be selected every time the optimizer runs.

Denormalization is performed with the `denormalize` function using stored min/max values from the normalization.

The `generate_withoptimizer` function takes the number of samples, the number of steps, the learning rate and the performance function as arguments. The number of samples is set to 1, but could be increased to `n` to initialize a latent matrix `z` with shape `n, latent_dim`. Decoding the samples returns a shape `[n, input_dim]`. When more than 1 sample is used a shared optimization process using batched gradient descent needs to be applied to improve all `n` samples in parallel. This method allows more design candidates in one run.

The number of steps is the number of optimization iterations. It determines how long the optimization will run. More steps gives more time to converge and could potentially lead to to better results. Too many steps could lead to overfitting.

The learning rate controls the size of each gradient step. A larger learning rate results in faster convergence but gives a higher risk of overshooting or instability. A smaller learning rate is slower but a safer way to run the optimizer. The learning rate is set to $lr = 0.01$ by standard and can be altered.

An early stopping mechanism is implemented as well to stop the optimizer after a number of epochs without improvement, considering a certain threshold.

After running the `generate_withoptimizer` function the `track_and_export` function can be used to plot and save the results. In this function the performance metrics are denormalized and the now continuous bitmap values are clipped to 0 or 1.

The resulting sample is not necessarily the best performing one, it can also be under or overpredicting the structural behaviour the most. The only way to know this and filter these results out is to run the samples in GH. Two options are considered to overcome this issue. The first one is to use the parallel

optimization of multiple samples to come to a set of design candidates of which the best one can be selected. The other option is to save for example the last 100 samples of the optimization process and calculate the actual performance. Because the difference in the last 100 samples might be small, there is not a lot of improvement in the end, the samples may all contain the same bad predictions. The first option results in more varied designs, with higher changes one of them being correctly predicting the good performance.

Using multiple latent vectors in parallel encourages the diversity in designs, reducing the risk of converging to the same overly optimistic prediction. It increases the change that at least one candidate is close to the true optimal. This method still relies on the predictive accuracy of the decoder. It also requires more post-processing running all samples in GH to evaluate the actual performance. It might also require more computational resources.

4.3.1. Resulting algorithms

The difficulty of controlling the GD algorithm led to different approaches for the optimization process.

The first algorithm is from now on referred to as the random sampling algorithm. Instead of performing gradient descent it draws a large number of samples z and calculates the performance score. This process is very quick meaning 100,000 samples can be evaluated in a few seconds. Only the best sample is saved, the memory is cleared and multiple runs are performed. Because only the best of 100,000 samples is stored, the performance is always optimistic. An evaluation in GH of all best samples should determine the actual performance scores.

The second algorithm is referred to as the GD optimizer. This is the optimizer described in the previous part of this section. A drawback of this method that if it starts with poor starting points the optimization will often not be able to end in a good scoring result. To overcome this during further application, this algorithm was combined with the random sampling algorithm.

The final algorithm is the GD optimizer with initial random sampling, combining the previous two methods. To ensure good starting points first a number of samples z are drawn and only the best scoring one is optimized with the GD optimizer.

Although the The GD optimizer with initial random sampling performs the best, the experiments with the other two algorithms also include important results, and all three are included in the following chapters.

5

Performance score

If we only minimize for mass, a design consisting of only the lightest regular ribbed modules [0000] will be the best. Other objectives and constraints need to be included in this optimization process as well. To implement code based constraints in the performance based objective mass or elastic energy a performance score can be used.

For this optimization problem there are two distinctive categories of constraints; structural performance based and stock constraints. In this chapter the general implementation of the performance score is discussed. In chapter 8 the performance score is extended with stock constraints.

5.1. Structural Performance with penalties

Using penalty functions to deal with constraints is a common approach in optimization problems (10). With modules of all equal rib dimensions the regular rib module has the lowest mass. To produce interesting results the design with only regular rib modules should not satisfy the other constraints otherwise this option is automatically the best.

To combine different performance metrics and improve model training the normalization is applied to all datasets with the following function, resulting in a score between 0, lowest and 1, highest value.

$$normalized\ data = \frac{data - min(data)}{max(data) - min(data)} \tag{5.1}$$

To the mass, penalty values for the code based constraints can be added. Two options are considered, a linear and a squared penalty. The penalty is calculated as the (squared) difference between the calculated value and the defined maximum value. The resulting penalties from multiple constraints can be added with additional weighing factors to tune the optimization process.

Algorithm 1 Linear displacement penalty

if $d < d_{max}$ **then**
  $d_{penalty} = 0$
else
  $d_{penalty} = (d - d_{max})$
end if

The penalties can be added to the normalized values of the mass with a performance function. The scoring can be tuned with using a weight α

$$p_{overal} = mass_{normalized} + \alpha * penalty \tag{5.2}$$

Algorithm 2 Squared displacement penalty

if $d < d_{max}$ **then**
  $d_{penalty} = 0$
else
  $d_{penalty} = (d - d_{max})^2$
end if

5.2. Python Application

In the python script the performance score can be set for both inside and outside the VAE model. Inside the model the performance function can be used to evaluate performance scores when optimizing in latent space. Outside the model the performance can be calculated of samples that are coming from GH.

The performance function needs samples as input and allows you to set a mode, constraints, penalty weight, stock weight and available modules. From the samples to be evaluated the performance metrics are extracted. The mode defines which performance function to select, some examples are: `min_mass`, `min_energy`, `constrained_min_mass_linear`, and `constrained_min_mass_squared` using the linear and squared algorithms above. The constraints that are passed to the function need to be normalized. This can be done with the `normalize_with_reference` function, that uses a `perf.min()` and `perf.max()` saved during the normalization of the dataset as a reference to normalize the constraints. The penalties are computed with `torch.relu(displacement - constraint)`. After the difference between the performance metrics and the constraint is calculated. `torch.relu` filters out the negatives values, so when the constraint is not violated is passes zero. The penalties are first summed and then added with a `penalty_weight` to the performance objective. Multiple different weights for constraints could be implemented by changing the performance functions that are being used. The performance function that is used inside the VAE to optimize, takes the mean of the resulting score to ensure the gradient magnitude stays consistent regardless of how many samples are in the batch.

Due to the normalization the resulting score, if the penalty weights are not too large, are usually between 0 and 1. However the performance objective is minimized, so if a sample is found that is better than the dataset used for the normalization and there are no or small penalties the score will go below zero.

6

Performance score based training

This chapter covers the alternative approach of performance based training that was abandoned at a later stage of this thesis, but still gives a lot of insights in model selection and dataset generation strategies. Performance based training means that the VAE model is trained on the the calculated performance score, a single numeric value. Performance based training was done for two cases, a symmetric 4 module case and a more complicated 13 module floor plan. The dataset generation depends on the values that were collected, which are mass, elastic energy and displacement. The VAE training is done after calculating the score. If another performance score or optimization objective is desired the VAE training and optimization has to be redone but the dataset generation not.

6.1. Performance score

For this chapter another structural model was used also considering a slab, columns and wall, which were removed later to make the structural analysis much faster. Note that this is the only case in which a score is maximized and not minimized.

The mass of the entire model is directly available after assembling the model. The maximum displacement and change in elastic energy are available directly after analyzing the model. Because other measures such as utilization are available after running another Karamba3D component, slowing down computation time for dataset generation, they are left out in this case.

A dataset is generated by running it through the GH script recording the bitmaps together with the outputs from Karamba3D for mass, displacement and elastic energy. Before a performance score can be calculated these results have to be normalized with the following function:

$$normalizeddata = \frac{data - min(data)}{max(data) - min(data)} \tag{6.1}$$

After normalizing the different measures there is a dataset with values between 0 and 1. Because we want all three measures to be minimized and a higher score to be better the performance is simply calculated as:

$$p_{mass} = 1 - mass \tag{6.2}$$

$$p_{elasticenergy} = 1 - elasticenergy \tag{6.3}$$

$$p_{displacement} = 1 - displacement \tag{6.4}$$

$$\tag{6.5}$$

The performance function takes into account the importance of the different structural performance measures and should be correctly weighed to prevent a bias for one measure.

Modules with more poles have more ribs and are heavier. These heavier modules result in lower displacement and elastic energy change, conflicting with the objective to minimize mass. For now some arbitrary weights were selected to test the VAE.

$$p_{overal} = 0.4p_{mass} + 0.4p_{elasticenergy} + 0.2p_{displacement} \tag{6.6}$$

6.2. Case 1: 4 Modules

When including the performance score at the end of the bit representation it results in a 1D tensor with 17 inputs for 4 modules e.g.
$[0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0.450612]$

Case 1 has a total of $2^{16} = 65536$ combinations, making it possible to use brute force to calculate all of them. With a simple python script in GH all 65636 combination of a 16 bit tensor are passed through the structural analysis. The resulting mass, elastic energy and displacement are saved to a csv file. In python the structural performance score formulas are applied to this dataset. Figure 6.1 shows the non normalized mass, displacement, elastic energy and the performance score after normalization and applying the performance score formulas.

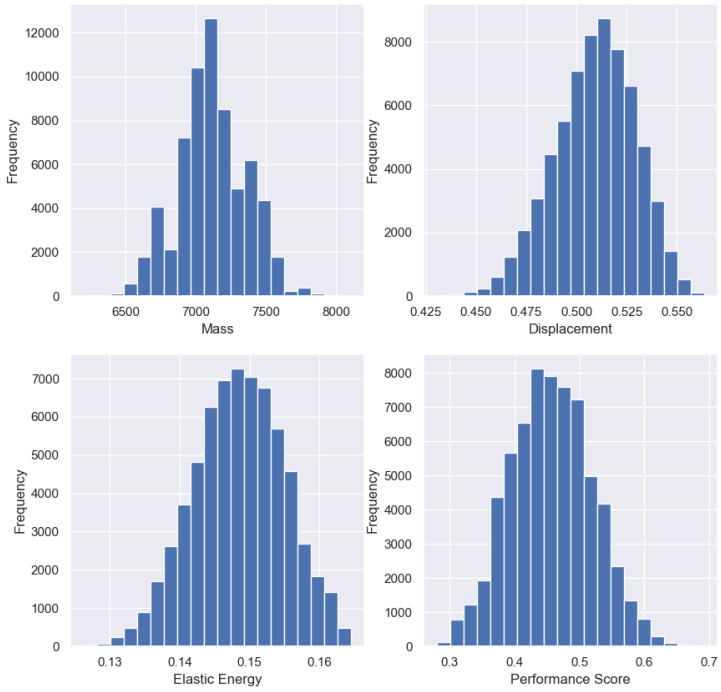


Figure 6.1: Histograms of Case 1: 4 Modules dataset

6.2.1. Hyperparameters

The following hyperparameters are considered for the model selection:

- latent dimensions
- hidden layers
- hidden units
- kld-loss

The VAE is trained to reconstruct these samples from the dataset. One difficulty that occurs is that the reconstruction with the standard VAE setup is not forced to reconstruct integers. For the reconstruction it is possible to clip the values to 0 or 1 afterwards to overcome this.

Table A.1 in Appendix A shows the different inputs for the hyperparameters that were tested. The simplest model tested was with 1 latent dimension, 1 hidden layer and 10 hidden units. All tests were done for 10 epochs. The following was concluded on changing these hyperparameters:

- Changing only the latent dimensions gave the best result for 12 dimensions.
- Increasing the number of hidden layers did not result in a lower validation loss for both 1 and 12 latent dimensions with 10 hidden units.
- With 12 latent dimensions and 1 hidden layer, 200 hidden units gave the best result
- Running the training again with these hyperparamters but instead of 10 epochs, 100 epochs and an early stopping mechanism with a patience of 10, the best validation loss is 0.0114 and training was finished after 34 epochs.
- After removing the best 100 samples the training results in a slightly higher loss as seen in Figure 6.2
- The loss gives an indication how far of the reconstructions are, so 0.011 on a score between 0 and 1 could be good enough to make predictions, a loss of 0.1 might not work. More experiments on a relationship between the value of the loss and the prediction quality are performed in the next chapter.

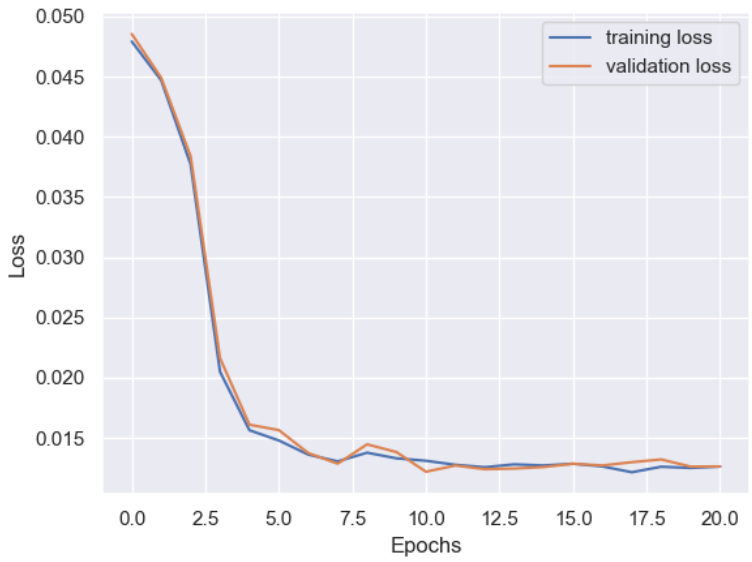


Figure 6.2: Losses after removal of 100 best samples: best validation loss is 0.0122, with early stopping triggered after 21 epochs

6.2.2. Sample generation

When samples are generated by the VAE generate function these new configurations consist of a tensor of floats instead of bits. These can be rounded to bits after the generation, but that does not overcome the problem that the the possible amount of samples to generate are not bound to just $2^{16} = 65536$ anymore. So when a certain amount of samples are generated, with afterwards rounding the floats to bits many duplicate solutions with different performance scores should be generated and many possible configurations could be missed out. Fortunately the generate function can generate 100,000 new samples and sort out the best ones. Because this does not always result in the best solution, a function was written that performs these 100,000 generations, stores the best results, clears the memory, and runs it again.

To validate this method further the 20 best samples were removed from the dataset above to test if the VAE model was able to generate those configuration without having seen them in training. The generate function ran 300 times taking around 2.5 minutes every time saving the best of 100,000 generations. Of the 300 best results, 12 times the best possible solution determined by brute force was found. The calculated performance score of this configuration is 0.6919.

The order in the 300 predictions, the predicted scores and the errors are in Table 6.2. With a average score of 0.6754 and an average error of 0.0165 below the actual score of 0.6919. However, they are all higher than the highest performance score in training of 0.6522.

Table 6.1: Best generations with 4 Modules

Order	Predicted Score	Error
280	0.6653	0.02659
261	0.6662	0.02526
240	0.6684	0.02345
218	0.6702	0.02165
150	0.6745	0.01730
128	0.6757	0.01620
102	0.6776	0.01431
93	0.6778	0.01405
78	0.6792	0.01269
64	0.6809	0.01099
63	0.6811	0.01083
22	0.6875	0.00441

To test it further the training was done again with the removal of the 100 best samples as well. Removing the 100 best samples the highest performance score is 0.6327.

Again 12 correct best samples were found.

Table 6.2: Best generations with 4 Modules

Order	Predicted Score	Error
28	0.6696	0.02233
40	0.6652	0.02670
91	0.6569	0.03499
129	0.6532	0.03866
135	0.6524	0.03945
165	0.6502	0.04171
190	0.6483	0.04358
230	0.6446	0.04728
246	0.6431	0.04880
254	0.6424	0.04944
285	0.6390	0.05288
295	0.6367	0.05517

6.2.3. Conclusion 4 Module case

With the 4 module case the VAE is able to generate the best sample after removing the best 100 known samples. 12 out of the 300 best sample generations generated the best result known by brute force. If the best result is unknown the performance of a number of best samples has to be calculated to validate which are actually correctly predicting a high score and which are over-predicting a non-optimal configuration. This proves a VAE is able to generate samples that are better than the training data.

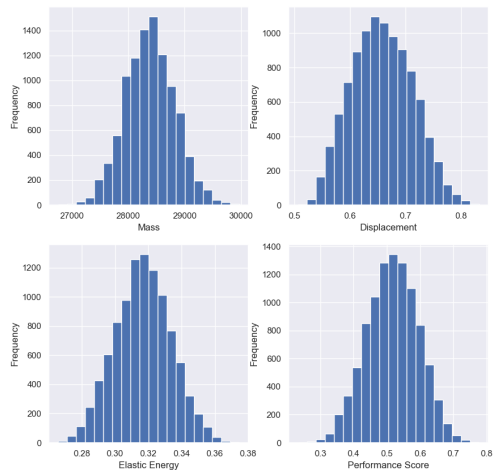
6.3. Case 2: 13 Modules

For the 13 Module case it is not possible anymore to obtain a brute force dataset with all possible configurations as this dataset would contain $16^{13} = 10^{15}$ samples. For this case one of the datasets has to be chosen. The hyperparameters have to be tuned again as well.

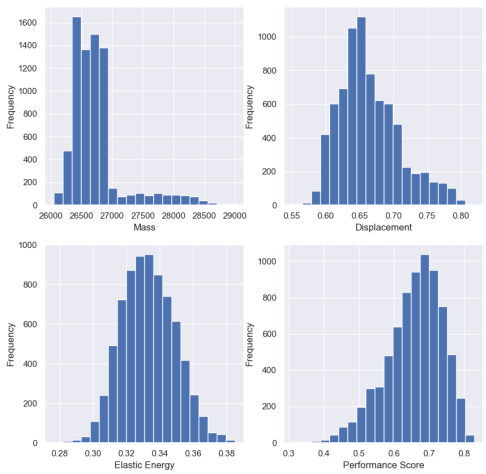
6.3.1. Dataset generation

Different options for dataset generation were considered in the 13 Modules case. The histograms are shown in Figures 6.3 ,6.4 and 6.5.

1. **Random samples** can be used to create a dataset with n completely random chosen configurations of modules. A disadvantage can be that a large part of the samples performs badly and the VAE is not able to generate samples with a high performance score. 17293 random samples were generated with a best performance score of 0.7778, without removal of samples.
2. **Evolutionary algorithm output** can be used to create a dataset saving all the outputs from the generic solver. Because the solver is already trying to optimize the performance score the samples are expected to yield higher scores, so the VAE can be trained with these higher scoring samples. After two runs of the evolutionary algorithm and removal of duplicates, the dataset consists of 7500 samples, with a highest performance score of 0.8284.
3. **Simulated Annealing algorithm output** After two runs of the simulated annealing algorithm and removal of duplicates the dataset consists of 4874 samples, with a highest score of 0.8284
4. **Combination of Evolutionary and Simulated annealing** The Evolutionary and simulated annealing datasets combined consist, after removal of duplicates, of 12014 samples, with a highest preformance score of 0.8284.
5. **Combination of all options** to create the largest dataset with some bias for good performing configurations. After removal of duplicates the All Combined dataset consist of 29307 samples, with a highest performance score of 0.8284.

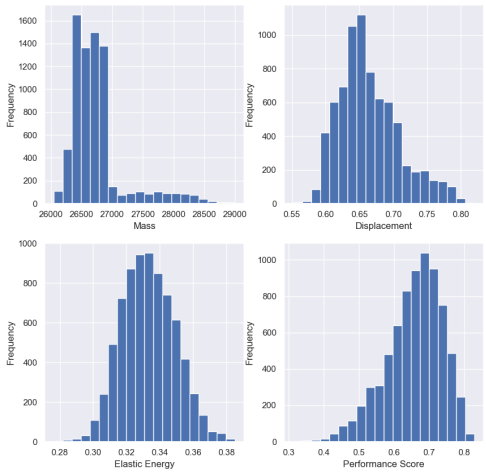


(a) 17293 random samples

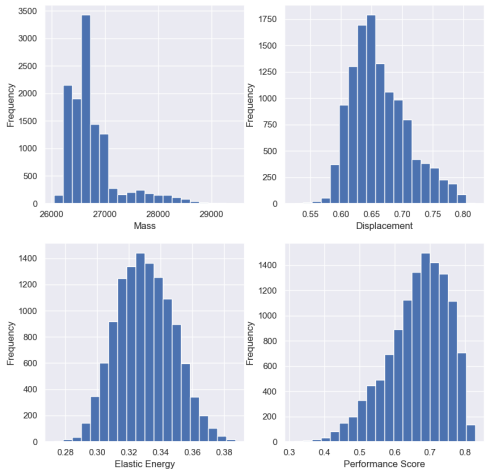


(b) 7500 Evolutionary samples

Figure 6.3: Histograms for 13 Module datasets



(a) 4874 simulated annealing samples



(b) All 12014 generic samples

Figure 6.4: Histograms for 13 Module datasets

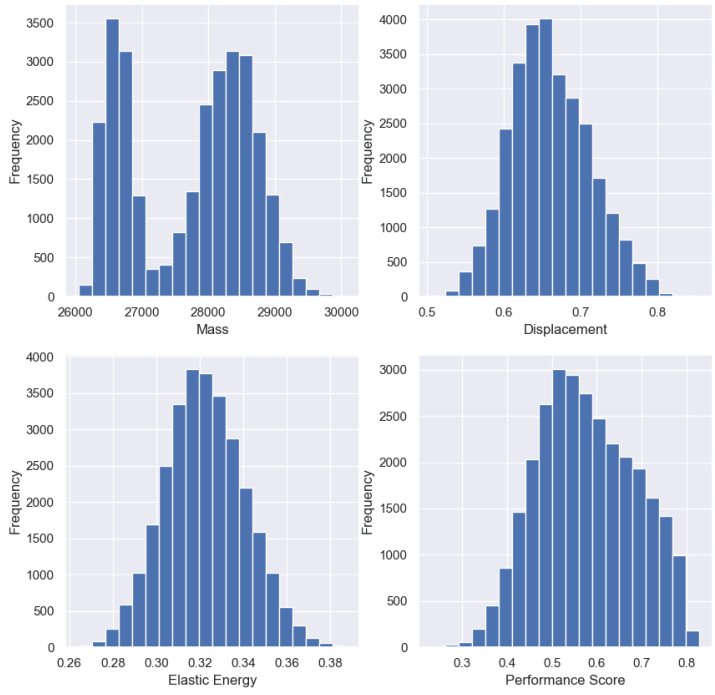


Figure 6.5: Histograms of Case 2: 13 Modules dataset with 29307 samples

6.3.2. Hyperparameter tuning

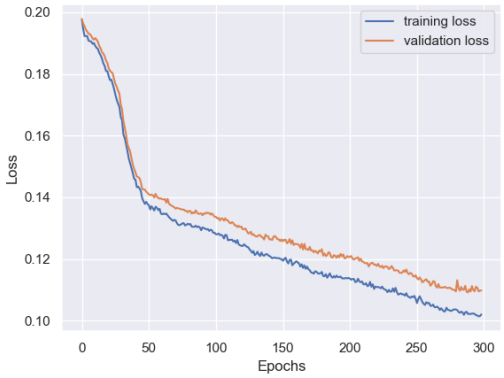
To tune the hyperparameters three datasets were combined; 10,000 random samples, 2601 Evolutionary samples and 4876 simulated annealing samples. These datasets were used to get a first impression of the hyperparamters and were later extended.

Table A.2 and A.3 show the different combinations of hyperparamters explored. A model with 13

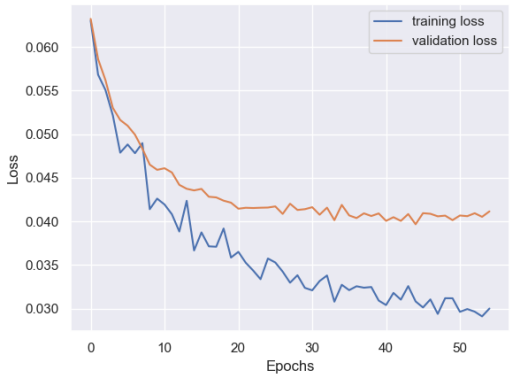
latent dimensions, 1 hidden layer, 300 hidden units and a kld-loss of 5.00e-4 was chosen.

6.3.3. Dataset Selection

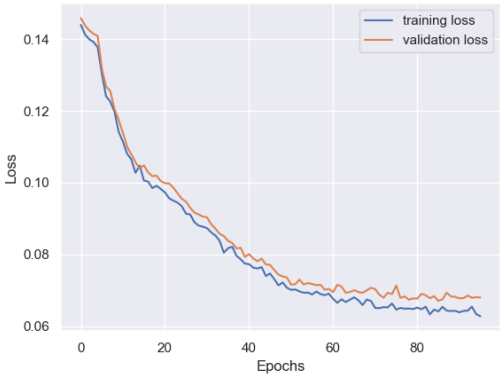
The model selected above was trained with three different datasets. The 17293 Random samples dataset, the 12378 samples Generic combined dataset and the 29671 samples All Combined dataset. The training process is shown in Figure 6.6



(a) Losses using the Random Combined dataset: best validation loss is 0.1, with early stopping triggered after 299 epochs



(b) Losses using the Generic Combined dataset: best validation loss is 0.030, with early stopping triggered after 55 epochs



(c) Losses using the All Combined dataset: best validation loss is 0.067076, with early stopping triggered after 96 epochs

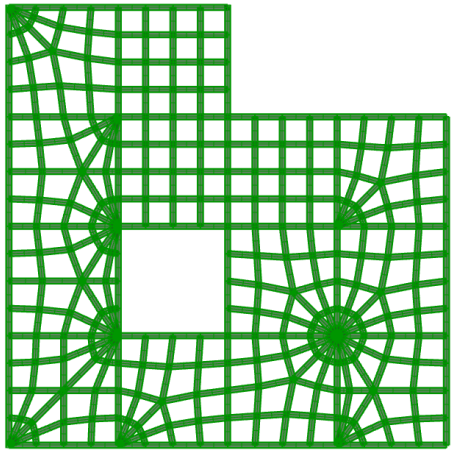
Figure 6.6: Training losses for three different 13 Module datasets

6.3.4. Sample generation

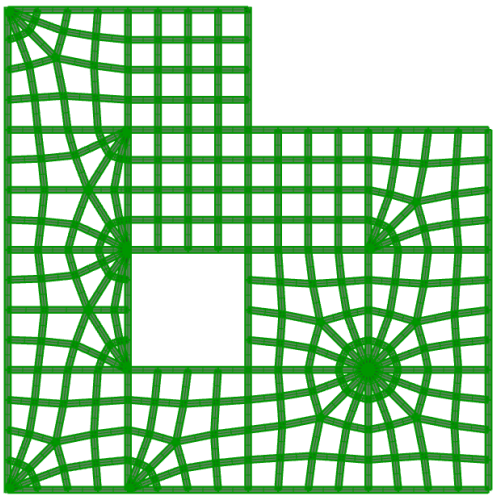
The trained model with the lowest loss, resulting from the Generic Combined dataset, was selected to use for the generation of new samples. From 300 runs of 100,000 generated samples the best one was selected, resulting in 300 configurations with high predicted performance scores.
The 300 bitmaps were loaded in GH to calculate the actual performance score for these configurations. Out of 300 samples 22 have a higher calculated performance score than the dataset. The highest score found was 0.8667, which is also higher than the The manual configuration from the paper (24) that results in a performance score of 0.8415. Although note that in this paper elastic energy was minimized, and now a combination of mass, displacement and elastic energy is optimized. Figure 6.7 shows the best 4 configurations.

Interesting to note is that:

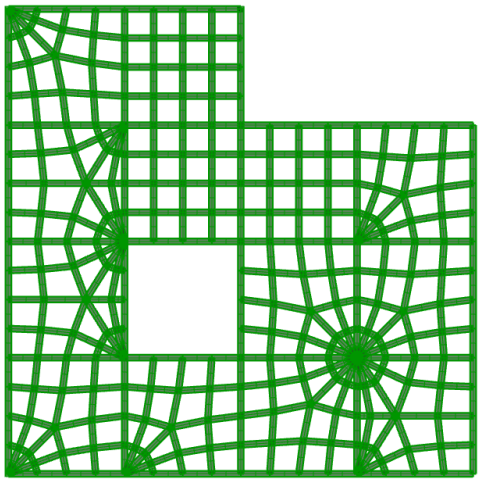
- The second best option was found twice
- The manual configuration was not found
- The best sample from the dataset was not reconstructed and found



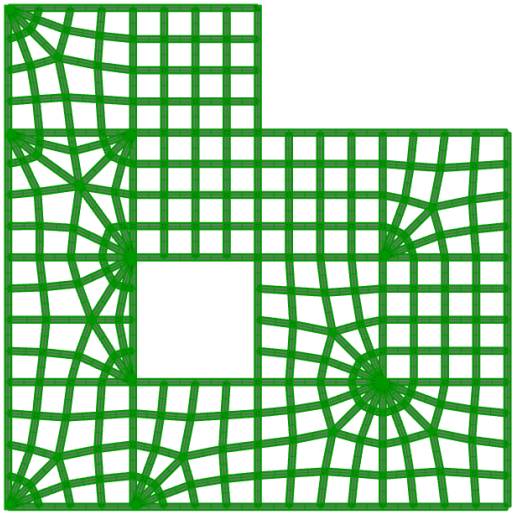
(a) Sample 177 with a score of 0.8667



(b) Sample 211 with a score of 0.8594



(c) Sample 47 with a score of 0.8562



(d) Sample 180 with a score of 0.8478

Figure 6.7: Generated samples with highest performance score

III

Application

7

VAE predicting structural behavior

In this chapter the VAE workflow is tested predicting the structural behavior of the modular ribbed floor system. The prediction capability of the VAE is tested on two aspects, increasing the problem size and increasing the dataset size and/or quality used for training. The datasets are created with random configurations or using generic solvers optimizing for displacement and storing all results. The performance based training from the previous chapter is compared with the metrics based training in this chapter. By the end of this chapter it should be possible to draw conclusions on which model and strategy to use, also including choice of dataset in relation to the problem size.

Hypotheses: The following effects were expected before starting these experiments:

- The dataset size needs to increase for larger problems to result in the same prediction error.
- The prediction error will be the lowest when using brute force to calculate all possible combinations.
- Dataset generation methods using a generic algorithm with a bias for certain designs are expected to work better in VAE training resulting in a lower prediction error.

7.1. Problem Definition

To test the effect of increasing the problem size the structural model is increased in size from 2x2 up to 5x5 modules, see Figure 7.1. The 2x2 problem has 65,536 possible combinations. The order of magnitude of the number of combinations increases for the 3x3 problem to 10^{11}, for the 4x4 to 10^{19} and the 5x5 up to 10^{30}.

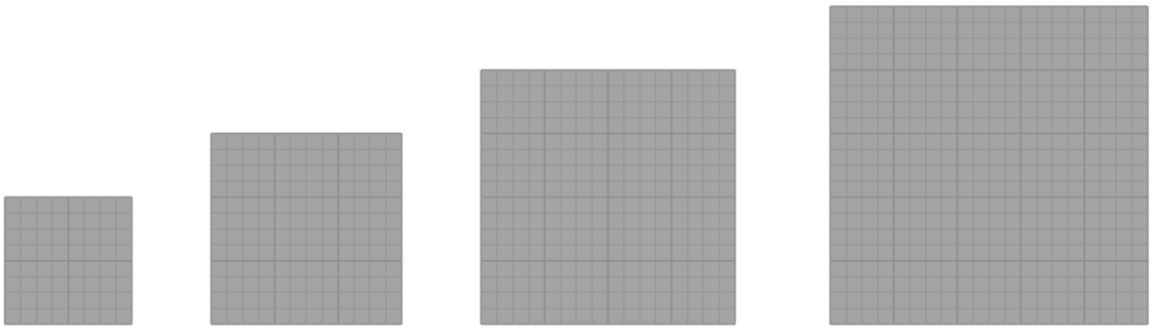


Figure 7.1: floor plans increasing in size from 2x2 to 5x5 modules

For this chapter one cross-section was selected, which is kept the same for all problem sizes. The height is 30 cm and the width 15 cm with a concrete class C30/37.

The same structural model as in previous chapters was used with a $5kN/m^2$ loading and the following load combinations:

$$ULS = 1.35G + 1.5Q \tag{7.1}$$

$$SLS = 1.0G + 1.0Q \tag{7.2}$$

As a result from chapter 3 the following metrics were selected to be interesting for the optimization problem:

- Mass [ton/m^2]
- Maximum Moment [kNm]
- Maximum Shear Force [kN]
- Maximum Displacement [mm]
- Elastic Energy [kNm]

After the selection of the module parameters it is visible that the deflection is the limiting constraint most of the time. Because one cross-section is kept when increasing the problem size, the smaller problems might stay within the constraints and the larger problems might not satisfy any of them. This has no influence on comparing the effect of the problem size and dataset quality and size on the prediction error.

7.2. Dataset creation

For every problem size different datasets are created increasing the number of samples and using different methods to create the datasets. The evolutionary and simulated annealing solvers can be used with a fitness on a specific structural performance measure, or random dataset generation can be used. For the 2x2 case it is also possible to use brute force to create a dataset with all possible configurations. Appendix B shows all the histograms of the datasets. As the displacement is the limiting factor in most cases it is used as fitness for the generic solvers.

From the histograms only, a few things can already be noted:

- When using generic solvers the shear force has two peaks. The lower shear force peak probably corresponds to designs with poles located at the supporting corners and the higher shear force to designs with one or more non poles at supports. In this case the shear force has less ribs to spread over leading to higher forces per rib. This is probably also the reason that the shear force is the most difficult to predict.
- For the 2x2 and 3x3 modules case the two shear force peaks are separated by a region without any samples and when using random samples there are very few samples in the low shear force area resulting in a lack of a second peak, see Figure 7.2a and 7.2b.
- The annealing solver results in a much more skewed histogram for displacement. It has more samples with a lower displacement, which is the optimization target in this case. However, evolutionary datasets seem to lead to better predictions.

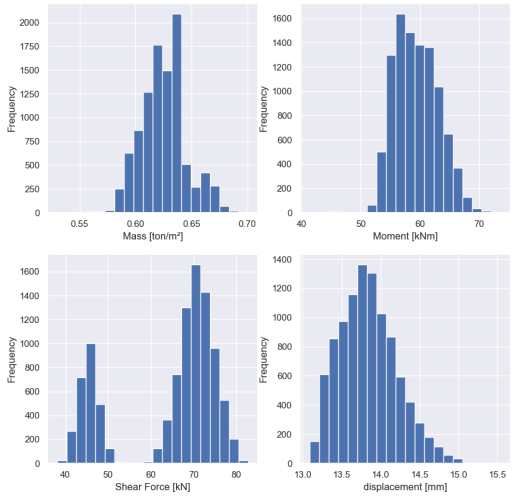
7.3. Model selection and VAE Training

for every dataset the following steps are performed in a python script:

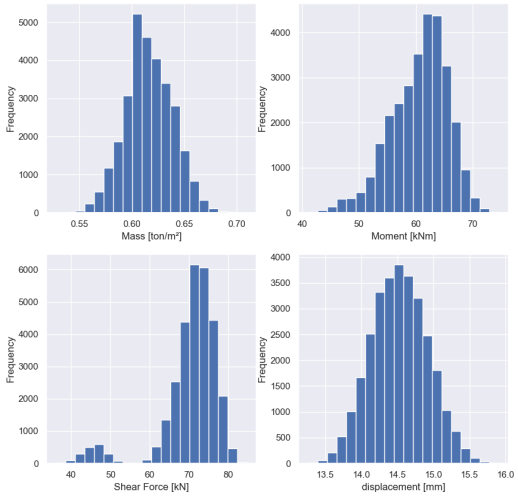
1. Load the dataset csv file
2. Plot histograms for the dataset
3. Normalize the data
4. Train the VAE
5. Generate 100 sample predictions and export
6. Calculate 100 samples in GH
7. Compute the difference between calculation and prediction

7.3.1. Model Selection

For simplicity the same architecture and hyperparamters are selected for all datasets. For both encoder and decoder, unless noted otherwise, an architecture with 1 layer with 200 units is used and 9 latent dimensions and 5e-4 kld-loss. The training is performed for 100 epochs with an early stopping after 10 epochs without improvement.



(a) Histogram 3x3 annealing, 10000 samples



(b) Histogram 3x3 Random, 30000 samples

Figure 7.2

For the datasets with random samples the validation loss is relatively high compared to the generic datasets. To test if a lower validation loss can lead to even better predictions the VAE architecture is reconsidered in this case. Increasing the latent dimensions up to 70 results in the lowest validation loss. The prediction error does not seem to change much compared to 9 latent dimensions with a higher validation loss. Because the training is not slower due to the increased latent dimensions it is kept on 70 for the random datasets.

7.3.2. Prediction quality

To test the prediction quality of all the models trained with different datasets 100 new samples are generated and then verified in GH. The mean absolute error is calculated for all the predicted structural metrics and shown in tables below. The full tables can be found in Appendix B

Table 7.1: Prediction error 2x2 datasets

2x2	Mass	Moment	Shear Force	Displacement	Elastic Energy	v-loss
Evolutionary 10000	2.3%	4.6%	8.9%	1.7%	2.7%	0.013
Annealing 10000	2.1%	4.0%	12.2%	1.8%	2.6%	0.013
Combined 20000	2.4%	5.0%	7.7%	1.6%	2.6%	0.012
Brute Force 65536	2.1%	7.0%	4.6%	1.4%	2.3%	0.011

Table 7.2: Prediction error 3x3 datasets

3x3	Mass	Moment	Shear Force	Displacement	Elastic Energy	v-loss
Evolutionary 10000	1.1%	6.3%	6.2%	0.8%	1.3%	0.020
Annealing 10000	1.5%	4.7%	11.7%	1.1%	1.6%	0.037
Combined 20000	1.6%	5.4%	7.0%	1.0%	1.6%	0.031
Random 30000	1.6%	6.7%	4.2%	1.0%	1.5%	0.063
Random 30000 (70lat)	2.0%	5.1%	5.9%	1.3%	2.0%	0.028

Table 7.3: Prediction error 4x4 datasets

4x4	Mass	Moment	Shear Force	Displacement	Elastic Energy	v-loss
Evolutionary 20000	1.2%	4.0%	6.8%	1.0%	1.3%	0.045*
Annealing 20000	1.5%	5.8%	10.7%	2.6%	2.9%	0.039*
Combined 40000	1.4%	4.6%	8.8%	1.0%	1.4%	0.05*
Random 20000	1.6%	5.3%	5.8%	1.2%	1.8%	0.15
Random 20000 (70lat)	1.3%	5.6%	5.6%	1.2%	1.8%	0.046

*Only the training graph is saved, not the exact validation loss

Table 7.4: Prediction error 5x5 datasets

5x5	Mass	Moment	Shear Force	Displacement	Elastic Energy	v-loss
Evolutionary 8000	0.57%	2.9%	4.1%	0.6%	0.8%	0.046
Random 5000 (70lat)	1.4%	4.1%	6.3%	1.0%	1.5%	0.12
Random 10000 (70lat)	1.0%	4.2%	6.5%	0.8%	1.3%	0.12
Random 20000 (70 lat)	1.2%	5.0%	6.0%	0.9%	1.3%	0.12

On the dataset creation strategy it can be concluded that:
- Lower validation loss does not necessarily mean better predictions.
- Using the Evolutionary solver leads to better prediction of the shear force than the Annealing solver.
- Combining Evolutionary and Annealing solvers does not lead to better predictions.
- Random datasets have about the same prediction quality as datasets created with the Evolutionary solver.
- Increasing the dataset size does not lead to a smaller prediction error, but it is still expected the dataset size matters for the generation of new samples and the optimization, which will be covered in the next section.

On the prediction quality in general it can be concluded that:
- Selecting the right dataset the prediction error of the mass, displacement and elastic energy is around 1-3% and of the moment and shear force is about 5-7%.
- The mass, displacement and elastic energy are better predicted than the moment and shear force. This is probably because the moment and shear force are highly depended on the path the forces can flow towards the supports and the symmetry of the design. If poles are situated at the supports the shear forces are divided over more ribs. If the design is more unsymmetrical higher moments can occur on one edge of the floor compared to the others. The VAE seems to have more difficulty to recognize and predict these cases.

On the problem size can be concluded that:
- The prediction error for the 2x2 case is probably higher because a change of modules has a larger effect on the structural behaviour. Or in other words, with a larger problem there are more ways in which the forces can flow towards the supports making the differences in performance for different designs smaller.
- The validation loss slightly increases when the problem size increases. The 5x5 problem starts to show some overfitting and especially for the random datasets it is needed to increase the latent dimensions up to 70 to keep this overfitting to a minimum.
- Dataset creation takes longer when increasing the problem size. For example, it takes 32 seconds to create 100 samples for the 2x2 the problem and 49 seconds for the 5x5 problem.

7.4. Optimization without constraints

How the gradient descent algorithm works was discussed in the chapter on the optimization workflow. In this section early stopping is turned on with a `patience=100` and a `min_delta=1e-4`. The algorithm is set to a learning rate of 0.01. The choice for the number of steps and the number of samples is discussed. The number of samples used is different throughout this section. This is not only to test how many samples need to be verified with a calculation, but also how many are practical to export to GH. The GH script for example supports a maximum of 254 samples to be imported in the 4x4 case, so

it was decided to use 250 samples. If more samples are wished the script should be changed (which is done in a later stage of this thesis) or the calculation has to run multiple times.

7.4.1. GD number of steps and clipping error

For this experiment the 5x5 Evolutionary dataset is selected because it is a bigger problem and the training results were very good. Different settings for the Gradient descent algorithm are tested. Early stopping is usually executed between the 6000 and 8000 steps. It was noticed that when the algorithm runs to the end until there is no improvement, the predictions become quite bad under-predicting all performance metrics. To confirm this and to find a solution for this issue different settings for the GD algorithm are tested. Table 7.5 increases the number of optimizations steps from 500 to 8000, to see what happens to the optimization of the mass and the prediction of the other metrics. The performance function used is minimization of mass without constraints. Every time the 20 best predicted samples are verified in GH.

Table 7.5: Gradient Descent to minimize mass, 5x5 datasets

Steps	lowest Mass	Mass error	Moment	Shear F.	Displ.	Elastic E.	clipping error
500	0.5703	1.45%	3.19%	6.92%	0.6%	0.92%	0.0973
1000	0.5751	3.3%	4.6%	2.8%	0.42%	0.66%	0.0951
2000	0.5719	3.8%	9.1%	10.3%	0.8%	1.5%	0.0731
4000	0.5669	3.6%	10.5%	16.7%	1.7%	1.7%	0.0608
6000	0.5727	3.6%	13.7%	21.7%	2.3%	2.7%	0.0453
8000	0.5698	3.7%	12.7%	23.5%	1.9%	2.8%	0.0544

Increasing the number of steps increases the prediction error and does not result in a lower mass. The best strategy to find a better performance seems to be to calculate more predicted samples in GH. When following this strategy using less steps makes the optimization much faster. Running time increases from about 3 min for 500 steps with 250 samples to over 35 min for 8000 steps with 250 samples, including early stopping usually triggered around 6000-7000 steps.

Something else that could be happening here is that the GD algorithm tries to optimize the performance by a small amount at the end by slightly changing the input bit values. This is possible because in this process these values are continuous and are only clipped to 0 or 1 later. The algorithm is then wasting a lot of time finetuning the performance to solutions that not really exist. To test this the clipping error is saved, which is the mean difference between the actual predicted values and the bits they are clipped to. Counterintuitively, the clipping error goes down for a larger number of steps, while the prediction error of the structural metrics is going up.

7.4.2. Bechmarking the 2x2 problem with GD

The 2x2 Brute Force dataset can be used to test if the VAE can come up with the best possible solution. When using the complete dataset and only minimizing the mass the VAE struggles to make a sample with all 0's, so the lowest mass is not found, see table 7.6. When minimizing for elastic energy the lowest value is found, see table 7.7. The lowest possible mass is $0.506\ ton/m^2$ for the grid structure (all 0's) and the lowest possible elastic energy is $0.2817\ kNm$.

Table 7.6: Gradient Descent to minimize mass, 2x2 datasets

Steps	lowest Mass	Mass error	Moment	Shear Force	Displacement	Elastic Energy
500	0.5214	4.5%	4.2%	23.3%	1.4%	3.6%
1000	0.5214	6.5%	4.3%	40.5%	2.0%	4.4%

Table 7.7: Gradient Descent to minimize elastic energy, 2x2 datasets

Steps	lowest Energy	Mass error	Moment	Shear Force	Displacement	Elastic Energy
1000	0.2817	5.6%	13.3%	9.0%	4.1%	2.1%

7.5. Optimizing with a performance function

In this section the performance function is integrated in the optimization process and some small experiments are done to get some insights in the best strategy to follow for the optimization. After defining the workflow, models, performance function and constraints the first experiment compares if metrics based or performance score based training results in better predictions using the Random Sampling Method. The second experiment looks into the effect of the stepsize using the GD Method. The third experiment compares the results of the GD Method for the Random, Evolutionary, and Annealing datasets. In the last experiment the GD optimizer is used again for all problem sizes with the best found settings in the previous experiments.

7.5.1. Workflow and training

The python script for metrics based training is altered to perform performance score based training. Now the scores have to be calculated before the training begins. This results in the following workflow:

- Load dataset
- set constraints
- Normalize dataset and constraints
- activate a performance score function
- calculate scores and export a dataframe with the scores added
- Drop the performance metrics columns
- create a training and validation loader
- Train the model
- Load the model
- Select a number of runs and samples and generate
- Make a dataframe with the best scoring samples per run
- Export the predicted scores and the bitmaps
- Calculate strucutral performance of the predicted bitmaps
- Calculate the score and check if there is a new best score

In previous experiments it was noted that one single sample from the dataset can be much better scoring than the rest. In previous experiments sometimes datasets were made by sampling a number of samples from a larger dataset every time the script was run. To make sure the comparison is completely fair in this experiment for every problem a dataset with exactly 20000 random samples without duplicates is made that can be used for both workflows. New models are trained with the new random 20000 datasets for the metrics based training as well. Because in chapter 6 training on performance score with 12 latent dimensions gave the lowest validation loss, that was the starting point here as well. However, changing the number of latent dimensions to higher values for larger problems seems to have a large effect in lowering the score. For the 4x4 case 70 latent dimensions worked well (which is higher than the dimension of the input data) and for the 5x5 case trained on performance metrics as well. For the 5x5 case trained on the score 70, 100 and 120 latent dimensions was tried but for all cases the best scores are almost twice as high as in the dataset. The same latent dimensions are used for the metrics based training. An overview is given in Table 7.8.

Table 7.8: Latent dimensions

Problem size	latent dimensions
2x2	12
3x3	12
4x4	70
5x5	70

For every problem size some fictive constraints are chosen that force to look for solutions with a lower moment and displacement when minimizing mass. The performance score with a 1.0 linear penalty on these constraints is used. The constraints are summarized in Table 7.9

Table 7.9: constraints

Problem size	Moment constraint [kNm]	displacement constraint [mm]
2x2	22	2.5
3x3	55	14
4x4	100	44
5x5	150	112

7.5.2. Performance based versus metric based training with the Random Sampling Method

With metrics based training the model is trained to predict all 5 performance metrics when creating new samples and the performance score is calculated when new samples are generated. With performance score based training the performance score is applied on the dataset and only the bitmap with the score is used to train the model. The disadvantage of the last approach is that it is not possible to compare different performance score functions or penalty values without retraining the model again, which can take up to 10 minutes for larger datasets. The predictions are generated with 200 runs with 100,000 samples each, taking between 1 and 4 minutes to generate, depending on problem size. Table 7.10 shows for every problem size the best 5 scores from generated samples versus the best 5 scores from the dataset.

Table 7.10: Best 5 scores from dataset versus best 5 predictions, trained on scores

2x2 VAE	2x2 data	3x3 VAE	3x3 data	4x4 VAE	4x4 data	5x5 VAE	5x5 data
0.3197	0.3171	0.1017	0.1488	0.2695	0.3410	0.5417	0.3570
0.3197	0.3386	0.1139	0.1894	0.2993	0.3910	0.7065	0.3745
0.3197	0.3452	0.1223	0.2024	0.3140	0.3915	0.7143	0.3792
0.3372	0.3495	0.1226	0.2072	0.3543	0.3961	0.7382	0.3838
0.3372	0.3536	0.1226	0.2211	0.3684	0.4026	0.7408	0.4502

The best results from performance based training are compared to the datasets as well with the results shown in Table 7.11

Table 7.11: Best 5 scores from dataset versus best 5 predictions, trained on performance metrics

2x2 VAE	2x2 data	3x3 VAE	3x3 data	4x4 VAE	4x4 data	5x5 VAE	5x5 data
0.3167	0.3171	-0.1258	0.1488	0.3076	0.3410	0.2903	0.3570
0.3167	0.3386	-0.0604	0.1894	0.3450	0.3910	0.3158	0.3745
0.3167	0.3452	-0.0392	0.2024	0.3539	0.3915	0.3441	0.3792
0.3167	0.3495	-0.0154	0.2072	0.3753	0.3961	0.3648	0.3838
0.3167	0.3536	-0.0077	0.2211	0.3756	0.4026	0.3650	0.4502

The VAE trained on the performance metrics gives better results than the datasets every time but most of the time quite minimal except for the 3x3 case that results in a much better score, being negative as the weight is lower than the lowest weight from the dataset. The performance score trained VAE doesn't work well for the 5x5 module problem. It could be that the VAE trained on performance metrics scales better.

7.5.3. Stepsize of the GD method

The random sampling and GD method are compared by running it for 200 samples with a learning rate of 0.1. The number of steps is increased to see if there is an optimal number steps in this case. 250 steps results in the best found score, being quite close but not beating the random sampling method. The GD optimizer with 250 steps takes about 1 minute to run.

Table 7.12: Best 5 scores from dataset versus best 5 predictions, trained on performance metrics with GD

3x3 data	10 steps	50 steps	100 steps	200 steps	250 steps	300 steps	400 steps
0.1488	0.2395	0.1072	0.0156	-0.0089	-0.0413	-0.0069	0.0228
0.1894	0.2914	0.1781	0.0.0705	0.0119	0.0173	0.0294	0.0443
0.2024	0.3367	0.1833	0.1144	0.0238	0.0249	0.0347	0.0482
0.2072	0.3526	0.1900	0.1283	0.0510	0.0426	0.0608	0.0494
0.2211	0.3610	0.2013	0.1392	0.0763	0.0426	0.0614	0.0535

7.5.4. Different Dataset Generation Strategies

To compare the dataset generation methods a Random, Evolutionary and Annealing dataset for the 4x4 problem of all 20000 samples are compared. They are all trained with 70 latent dimensions for 100 epochs with early stopping with a patience of 10. The VAE results are generated with the GD method using 200 samples, 250 steps and a learning rate of 0.1. The Annealing dataset has the best score, but got the lowest VAE improvement. The Random dataset works much better than the Evolutionary or Annealing dataset. Comparing Table 7.13 with Table 7.11 it is also visible that with using the GD here instead of the random sampling method the VAE predictions are much better for the Random dataset. The clipping errors are very similar for all datasets with 0.058 for the Random 0.046 for the Evolutionary and 0.063 for the Annealing dataset.

Table 7.13: Best 5 scores from dataset versus best 5 predictions

Random data	Random VAE	EVO data	EVO VAE	Annealing data	Annealing VAE
0.3410	0.0777	0.3577	0.2053	0.3133	0.2113
0.3910	0.0857	0.3584	0.2095	0.3293	0.2196
0.3915	0.0926	0.3587	0.2136	0.3521	0.2299
0.3961	0.0937	0.3627	0.2151	0.3564	0.2328
0.4026	0.0947	0.3668	0.2239	0.3583	0.2465

7.5.5. Comparing problem size with Random dataset and tuned GD

To create Table 7.14 the same process is repeated as for Table 7.11, but now with the GD using 200 samples, with 250 steps and a learning rate of 0.1. The results for the 5x5 problem improve by a lot. The scores of the 2x2 problem are still barely improving. Repeating the same workflow with the 2x2 BruteForce dataset of all 65536 samples the best possible score of 0.287694 is found from the dataset, so the result is quite close.

When creating manual designs for the 3x3 case, the grid design scores 0.0050, the 4 corners design -0.1674 and the 8 corners design 0.0466, so only the maual designed 4 corner design scores better than the VAE result and is not found by the VAE.

Table 7.14: Best 5 scores from dataset versus best 5 predictions, after tuning GD optimizer

2x2 VAE	2x2 data	3x3 VAE	3x3 data	4x4 VAE	4x4 data	5x5 VAE	5x5 data
0.3167	0.3171	-0.0413	0.1488	0.0777	0.3410	-0.0162	0.3570
0.3176	0.3386	0.0173	0.1894	0.0857	0.3910	0.0076	0.3745
0.3197	0.3452	0.0249	0.2024	0.0926	0.3915	0.0177	0.3792
0.3202	0.3495	0.0426	0.2072	0.0937	0.3961	0.0579	0.3838
0.3206	0.3536	0.0426	0.2211	0.0947	0.4026	0.0629	0.4502

7.6. Rules of thumb for training

This section aims to make a connection between number of modules and required number of latent dimensions and the number of dataset samples, resulting in rules of thumb. First the number of latent dimensions are altered for the different problem sizes. The best validation loss per problem size is shown in Table 7.15, considering increments of 10 for the latent dimensions. The number of samples is kept on 20,000 random samples. The training is performed with 1 hidden layer, 200 hidden units and a kld-weight of 5e-4. To keep the training time limited the maximum number of epochs is 100 with early stopping with a patience of 10.

Table 7.15: Best latent dimensions per problem size

Problem size	latent dimensions	v-loss
3x3=9	40	0.02708
4x4=16	70	0.04801
5x5=25	120	0.08361

As a rule of thumb for the number of latent dimensions 4-5 times the number of modules is a good starting point to find the optimal.

The possibility of using smaller datasets could be very valuable information as the dataset generation is the most time consuming step in the optimization process. The results using the GD optimizer on the performance scores with 200 samples, 250 steps, learning rate 0.01, penalty weight 1.0 with the constraints from Table 7.9 are shown in Tables 7.16, 7.17 and 7.18.

Table 7.16: Decreasing the dataset size, 3x3 with GD optimizer

20,000 VAE	20,000 data	10,000 VAE	10,000 data	5,000 VAE	5,000 data	2,000 VAE	2,000 data
-0.1548	0.1488	-0.1235	0.2401	-0.1278	0.2447	-0.0904	0.2447
-0.1258	0.1894	-0.0829	0.2443	-0.1077	0.2454	-0.0873	0.2454
-0.1258	0.2024	-0.0779	0.2448	-0.1074	0.2490	-0.0855	0.2772
-0.1132	0.2072	-0.0592	0.2454	-0.0949	0.2589	-0.0842	0.2857
-0.0967	0.2211	-0.0571	0.2457	-0.0855	0.2715	-0.0780	0.3057

Table 7.17: Decreasing the dataset size, 4x4 with GD optimizer

20,000 VAE	20,000 data	10,000 VAE	10,000 data	5,000 VAE	5,000 data	2,000 VAE	2,000 data
0.0803	0.3410	0.0831	0.3697	0.0589	0.3708	0.0812	0.3873
0.0987	0.3910	0.0948	0.3705	0.1367	0.4448	0.0995	0.4701
0.1013	0.3915	0.1027	0.4437	0.1388	0.4732	0.1540	0.5134
0.1107	0.3961	0.1256	0.4732	0.1389	0.4895	0.1546	0.5478
0.1127	0.4026	0.1371	0.4889	0.1420	0.4914	0.1670	0.5537

Table 7.18: Decreasing the dataset size, 5x5 with GD optimizer

20,000 VAE	20,000 data	10,000 VAE	10,000 data	5,000 VAE	5,000 data	2,000 VAE	2,000 data
-0.0299	0.3570	-0.0897	0.3892	-0.0670	0.4591	-0.1066	0.5384
-0.0266	0.3745	-0.0414	0.4643	0.0018	0.4882	-0.0999	0.5994
-0.0156	0.3792	-0.0389	0.4847	0.0821	0.4941	-0.0628	0.6158
-0.0094	0.3838	0.0076	0.5053	0.0900	0.5161	-0.0461	0.6817
-0.0006	0.4404	0.0091	0.5115	0.0919	0.5185	-0.0354	0.6950

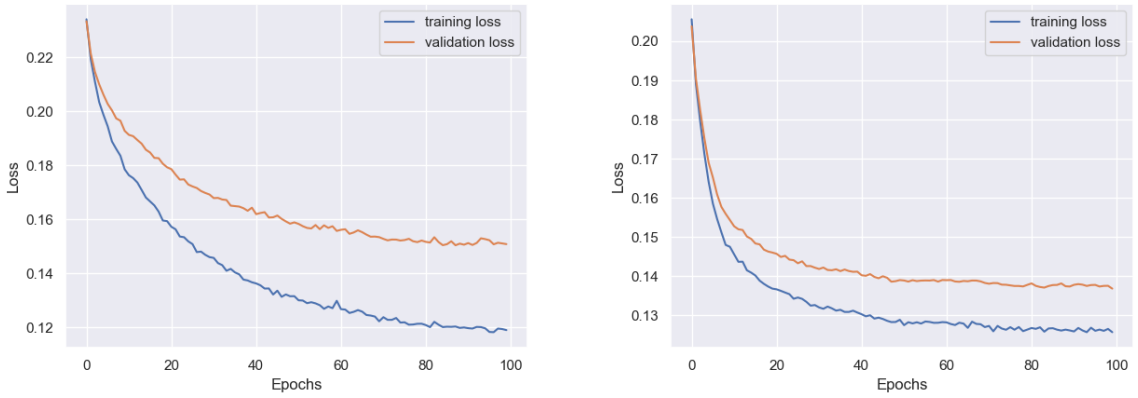
For all datasets used in this section the mean clipping error of 10,000 random generated samples, so not during the optimization, was calculated and is shown in Table 7.19. The error increases when the problem size increases and when the dataset size decreases. Both are in line with the increasing validation loss and the deceasing predictive abilities of the corresponding models.

Table 7.19: dataset size and clipping error

Dataset size	3x3	4x4	5x5
20,000	0.041	0.058	0.067
10,000	0.043	0.059	0.073
5000	0.060	0.071	0.084
2000	0.091	0.072	0.095

7.7. Limits on the problem size

Although still performing well during the training process the VAE started to show some overfitting for the 5x5 module size. The problem size is increased further to seek the limiting size. Increasing the problem size to 6x6 results in 36 modules meaning 10^{43} combinations. Following the rules of thumb 160 latent dimensions was used for the training. Figure 7.3 shows the training process for 4000 and 20,000 random samples is still stable. The increase of the validation loss continues with the increase of problem size, as could be seen before in Table 7.15. However, this increase in validation loss was not a problem for the performance of the VAE for the 5x5 case that is not the case anymore for the 6x6 problem. The clipping error is further increased to 0.115.



(a) Training 6x6 modules with 4000 random samples, val-loss = 0.15042 (b) Training 6x6 modules with 20,000 random samples, val-loss = 0.13684

Figure 7.3

7.7.1. Optimizing the 6x6 problem

The GD optimizer with initial sampling is used with 300 samples, 10,000 initial samples, 400 steps, and a learning rate of 0.01. The results for both the 4000 and 20,000 samples datasets are compared to those of the evolutionary solver in Table 7.20. The increase in performance between the 4000 and the 20,000 samples is rather small. The VAE still outperforms the evolutionary solver when it runs for only 3 minutes. Running it for 10 minutes, the results are quite similar.

Table 7.20: Benchmarking the Evolutionary solver with the 6x6 problem

Samples	Dataset generation	Training	Optimization	Verification	Elastic Energy [kNm]	Elastic Energy [kNm] Dataset
4000 20,000	45 min 3 hours 45 min	43s 11 min	10 min 15 min	3.5 min 3.5 min	6.30 6.27	6.63 6.63
EVO EVO			3 min 10 min		6.66 6.28	

7.7.2. Optimizing the 15x15 problem

Finally a 15x15 problem is tested resulting in 225 modules giving 10^{270} possible combinations. Creating datasets for this problem size takes much more time as well with 8000 samples already taking 3.5 hours. The model is trained with 900 latent dimensions this time and the not very promising result is shown in Figure 7.4. The mean clipping error is now 0.474. Looking at the generated samples near all of them are around 0.5 instead of near 0 or 1 and some of them are around -0.5 strangely enough. Together with the high validation loss and the overfitting training graph, this already indicates that the problem size limits of the model are passed and the model cannot be trusted anymore.

Although the training results do not look very promising and trustworthy some small experiments are performed to see what the behavior of the model is. First the VAE is benchmarked again against

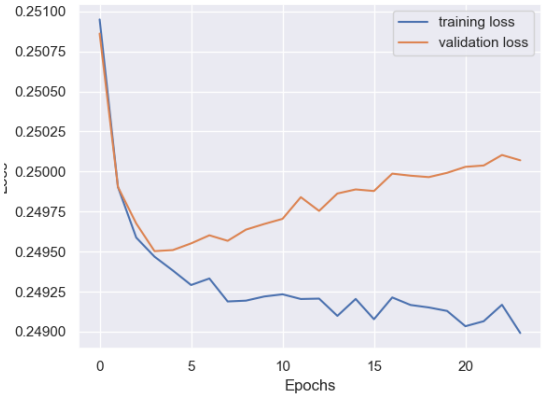


Figure 7.4: Training 15x15 modules with 8000 random samples, v-loss = 0.24950

the evolutionary solver minimizing elastic energy. The GD optimizer with initial sampling is used with 300 samples, 10,000 initial samples, 500 steps, and a learning rate of 0.01. The results are shown in Table 7.21. The VAE still works but is not able to outperform the evolutionary solver anymore.

Table 7.21: Benchmarking the Evolutionary solver with the 15x15 problem

Samples	Dataset generation	Training	Optimization	Verification	Elastic Energy [kNm]	Elastic Energy [kNm] Dataset
2000 8000	53 min 3:30 hours	30s 2 min	10 min 21 min	14 min 14 min	31.15 30.28	31.18 31.18
EVO EVO EVO			25 min 1:15 hours 2:20 hours		30.92 29.97 28.80	

Both solvers are no able to get close to an intuitive well performing design. The design shown in Figure 7.5 only includes poles at the columns and uses the grid modules to stay as light as possible. This results in an elastic energy of 24.47 kNm. The design with only the grid modules results in an elastic energy of 31.46 kNm.

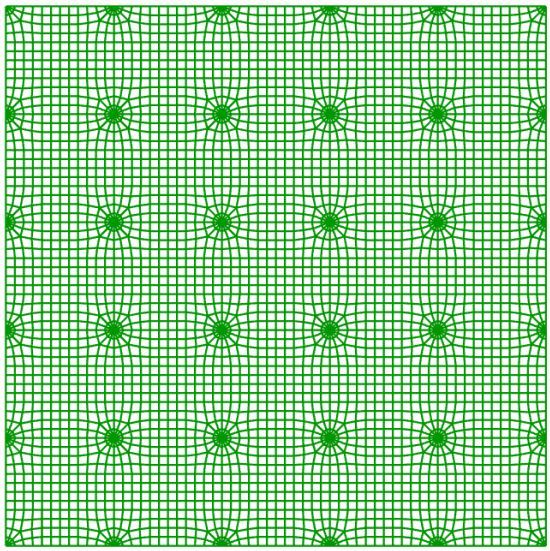
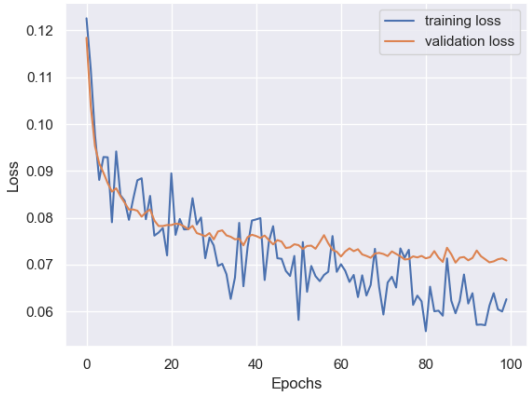


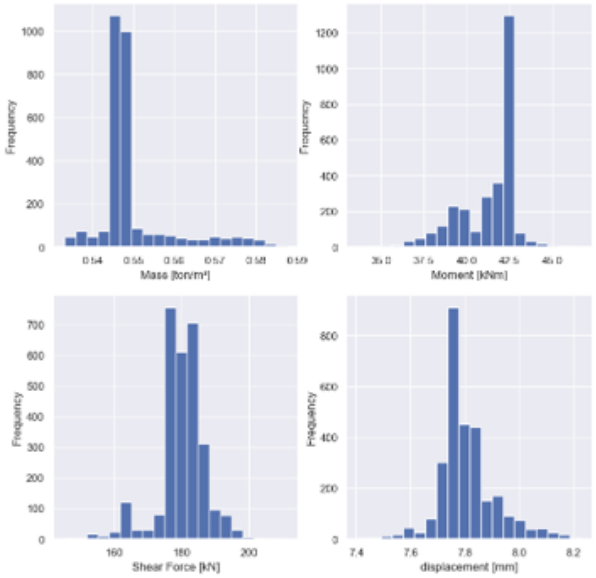
Figure 7.5: Intuitive good 15x15 module design with elastic energy of 24.47 kNm

7.7.3. Train with the EVO solver results

Another possibility is to train the model with the results from the evolutionary solver and look for a small local optimization. 3000 samples from a run of the evolutionary solver are used to train the model. From the histograms in Figure 7.6b can already be seen that the samples are very uniform. The training loss is not as stable as normally the case but the validation loss is. The optimizer is not able to find any improvement in elastic energy, but is very good in generating new samples with a very low clipping error with a 0.0046 average, meaning it is on average 0.0046 away from a 0 or a 1. The generated samples are very accurate predictions with around 5% error for the moment and only between 1 and 2% error for the other performance metrics.



(a) Training 15x15 modules with 3000 EVO samples, v-loss = 0.07043



(b) Histograms 15x15 problem with 3000 EVO samples

7.7.4. The 15x15 problem with stock constraints

The last experiment with the 15x15 module problem is to optimize with stock constraints as will be introduced in chapter 8. The stock from Figure 7.7 is available to use for the optimization problem. Both the VAE and the evolutionary solver are used. Using the dataset with 2000 samples the best score from the dataset is 0.49. The VAE and the evolutionary solver were not able to beat this score.

The optimizer uses 10,000 initial samples, 400 steps and a learning rate of 0.01. To attempt to decrease the optimization both 20 and 300 samples are optimized taking about a minute and 10 minutes respectively. The results are in Table 7.22.

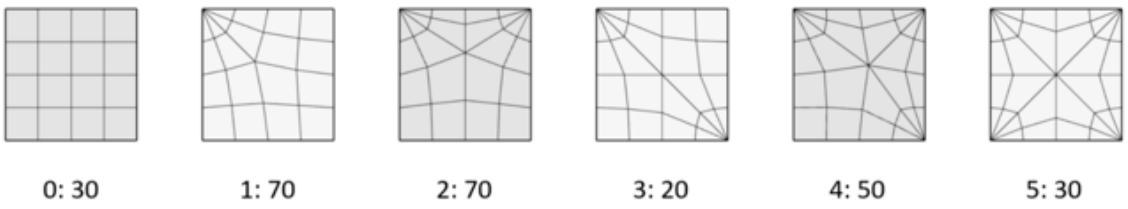


Figure 7.7: The available stock for the 15x15 problem with stock constraints

Table 7.22: Benchmarking the Evolutionary solver with the 15x15 problem with stock constraints

Samples	Elastic Energy [kNm]	Stock violations	Used modules	Score
2000, 20	31.99	8	8 65 54 25 53 20	1.29
2000, 300	31.77	6	11 65 54 23 53 19	0.89
EVO 3 min	33.26	4		0.93
EVO 10 min	31.96	6		0.79

7.8. Conclusions

Metrics based training and performance score based training both work. Metrics based training is preferred because it allows for more flexibility in changing constraints and performance scores.

The random sampling and GD optimizer proof to both be working to generate better samples. However, when running the GD algorithm longer the bitmap does not change anymore but it keeps lowering the predicted score. This happens because the tensor values are clipped to bits after a new sample is generated. So it starts to optimize away from the bits with a predicted score based on non-existing samples. The advantage of the random sampling method is that it is possible to simply keep increasing the number of sample generations to seek for better solutions. The combination of both methods into one optimizer increases the performance further and this optimizer is used in the rest of this thesis.

As the problem size increases the number of latent dimensions should increase as well with 4 to 5 times the number of modules to reach the best validation loss. The validation loss keeps increasing with the problem size and starts to overfit. For smaller problems, 25 modules or less, it is difficult to see a negative effect on the optimization performance of the VAE. For 36 modules the VAE optimizer behaves similar as the evolutionary solver. Increasing the problem size further the training process becomes more unstable, but the predicting qualities of the VAE still remain. However optimizing towards better samples than already in the dataset becomes very difficult.

The dataset creation strategy has a large effect on the performance. At first using generic algorithms to create datasets seemed to lead to better samples in the dataset, so also better generated samples. However, as can be seen in the histograms of the dataset this also gives more irregular distributions of performance metrics, making it more difficult for the VAE to predict and as a result models trained with random datasets perform more consistently well. Using generic solvers also decreases the flexibility of the model, because the dataset generation needs to use a fit on one certain performance metric. When using a random dataset the optimization objectives can quickly be changed using the same trained model. During the remaining chapters the VAE will be trained with random datasets to allow for this flexibility. If one wishes to create a VAE that makes very accurate predictions using a generic solver could still be the preferable choice losing some flexibility as result.

8

Stock constraints

Until now symmetric problems have been shown with intuitive solutions. Although the VAE resulted in better performance in some cases, using engineering sense on these problems could also lead to good solutions. In this chapter stock constraints are introduced leading to extra complexity encouraging the use of the VAE workflow.

8.1. Motivation for stock constraints

As stated earlier in this thesis the motivation to make the ribbed floor system modular is to allow for disassembly and re-usability. One could imagine a building that is no longer wanted, fit for purpose, or in the way of something else, and that therefore must be disassembled. The modules could then be used in another structure. The availability of the modules then depends on the modules that come available from disassembly. If not enough modules are available or the correct number of certain modules is not available, they should be fabricated. This results in a stock constraints problem that balances less efficient designs with available modules with the disadvantages of creating new modules needed for more efficient designs. In this chapter a performance score function is used that balances the stock constraints with the performance in a way that the optimizer behaves stable.

In a real application the performance function should be weighed such that the cost of producing a new module is balanced in terms of cost or embodied carbon compared with the design only using reused modules. This process must take into account the full life cycle of these modules, which can be very complicated and extensive. For example, also the carbon emissions from the transportation should be compared between a new and a reused module. Another difficult question is if the embodied carbon of a module should be accounted for over its entire life span, or only the first project and when reused, the embodied carbon does not have to be accounted for again, but only things like transportation and the erection of the building do. All of this could greatly change the formulation of an optimization problem with stock constraints, of course.

8.2. Workflow

The stock constraints workflow uses the metrics based workflow from the previous chapter. A new stock constraint function is created that is called in the performance function of the VAE. The complete workflow to perform a stock constraints case is as follows:

- Load dataset and previously trained model
- Set raw constraints
- Set stock weight, penalty weight, available modules and output folder
- Normalize constraints
- Set mode to `constrained_min_mass(or energy)_stockconstraints`
- Set performance function with the defined constraints, weights, modules, mode
- Generate new samples and calculate the performance score including the penalty for stock constraints
- Run the GH script for the new samples
- Load and normalize the verification file of the samples

- Calculate the actual performance score

The complete performance score is:

$$score = mass + constraint\_weight * constraint\_penalty + stock\_weight * stock\_penalty \quad (8.1)$$

With the penalty being the difference between the constraints for moment, shear and displacement if above the constraints. The scoring can be tuned by changing the `penalty_weight` and the `stock_weight`. The stock penalty is calculated with a new `check_stock_constraints` function. The function divides the complete bitmap tensor into the 4 bit module pieces, which are then converted into an integer index, from 0 to 15 using the big-endian binary representation. With 4 bits the following formula converts the bits into a integer index:

$$index = 8 * b_3 + 4 * b_2 + 2 * b_1 + 1 * b_0 \quad (8.2)$$

Index	Binary (Bitmap)	Module group
0	[0, 0, 0, 0]	0
1	[0, 0, 0, 1]	1
2	[0, 0, 1, 0]	1
3	[0, 0, 1, 1]	3
4	[0, 1, 0, 0]	1
5	[0, 1, 0, 1]	3
6	[0, 1, 1, 0]	2
7	[0, 1, 1, 1]	4
8	[1, 0, 0, 0]	1
9	[1, 0, 0, 1]	2
10	[1, 0, 1, 0]	3
11	[1, 0, 1, 1]	4
12	[1, 1, 0, 0]	2
13	[1, 1, 0, 1]	4
14	[1, 1, 1, 0]	4
15	[1, 1, 1, 1]	5

Table 8.1: All 16 possible 4-bit module bitmaps and their corresponding indices.

Then it is checked if the used modules are the same as the available modules given when running the function described by Algorithm 3. If one module used is not in the available modules list that is counted as one violation and the total number of violations is summed. This integer penalty value can than be weighed in the calculation of the performance score.

Algorithm 3 Stock Constraints

```
Convert each module's bits to integer indices
Map each of the 16 orientation IDs to a base module ID
Define tensors of zeros for the penalties and used counts
for All samples do Count the occurrence of each base module and save in used counts
    for Each base module do
        if used count > stock limit then
            add penalty
        end if
    end for
end for
```

8.3. Generating samples with random sampling only removing certain modules

The code described in the previous section can take quite some time to run. A faster method that could sometimes be sufficient to solve the problem is to only restrict certain modules and no to count all of them. This method is used in the following example.

As an example the 3x3 Random 20000 dataset is loaded into the python script. The same still a bit arbitrary constraints are used as before with Moment = 55 kNm and Displacement = 14 mm and no constraint or in practice a very high value for the shear force. Now the available modules can be defined and the weights should be able to tune the importance of the module availability, the weight and the constraints. Only the grid module is given as not available. With the random sampling method the best prediction of 100,000 samples is saved 200 times. Note that the score is built up with a structural performance part, which is a prediction and a stock constraints part, which doesn't need any verification. When calculated in GH. The best of these 200 samples are shown in Table 8.2 together with the structural results and if they violate the stock constraint. The lowest mass can be found by making the constraints more "soft", with smaller penalty values. With larger penalty values the constraints are strictly enforced and the mass found is still slightly lower than the best sample from the dataset.

Table 8.2: Stock constraints removing grid module

penalty-, stock- weight	Mass	Moment	Displacement	Score	violations
VAE constraints: 1.0, stock: 1.0	0.5479	52.87	14.17	0.1050	0
Data constraints: 1.0, stock: 1.0	0.5755	55.82	13.75	0.2355	0
VAE constraints: 0.1, stock: 0.1	0.5412	55.80	14.23	0.1102	1
Data constraints: 0.1, stock: 0.1	0.5548	59.62	14.61	0.1198	0
VAE constraints: 0.5, stock: 0.5	0.5548	54.24	13.91	0.0832	0
Data constraints: 0.5, stock: 0.5	0.5754	55.82	13.75	0.2232	0
VAE constraints: 2, stock: 10	0.5566	56.23	13.88	0.1672	0
Data constraints: 2, stock: 10	0.5754	55.82	13.75	0.2600	0
VAE constraints: 10, stock: 10	0.5686	53.88	13.75	0.1684	0
Data constraints: 10, stock: 10	0.5841	54.814	13.93	0.2644	0
VAE constraints: 2, stock: 1	0.5635	54.63	13.79	0.1367	0
Data constraints: 2, stock: 1	0.5755	55.82	13.75	0.2600	0
VAE constraints: 1, stock: 2	0.5635	56.33	13.95	0.1763	0
Data constraints: 1, stock: 2	0.5755	55.82	13.75	0.2355	0

Table 8.3 shows how many of the 200 samples violate the stock constraints for each penalty combination.

Table 8.3: Stock constraints violations out of 200 best samples

penalty-, stock- weight	violations
constraints: 1.0, stock: 1.0	156
constraints: 0.1, stock: 0.1	179
constraints: 0.5, stock: 0.5	156
constraints: 2, stock: 10	138
constraints: 10, stock: 10	140
constraints: 2, stock: 1	140
constraints: 1, stock: 2	151

Only removing one or more modules completely without keeping track of available stock allows for the evolutionary to solve this problem without the need for a scoring mechanism for the stock. The modules that are not available can simply be removed before the evolutionary solver starts not using them at all. When counting the modules used and restricting the use to a certain amount this is not possible and both methods need to use the `check_stock_constraints` function. Now the VAE becomes more relevant and in the next section it is compared with the evolutionary solver.

8.4. Benchmarking the evolutionary solver with stock constraints

To benchmark the evolutionary solver with stock constraints the 13 Module case is used, resulting in about 10^{15} possible combinations. The 10 cm x 20 cm cross-section is selected with C30/37 and still the same loading conditions. The constraints from section 3.4 are used with $M_{Rd} = 27.8\ kNm$, $V_{Rd,max} = 92.1\ kN$ and a maximum displacement of $d = 12\ mm$. The VAE is trained with 30,000 random generated samples to support the optimization for both mass or elastic energy, instead of a dataset specifically for one of the two. During training again a shallow model with only one layer, 200 hidden units and a kld-weight of $5e-4$ was used. During training the number of latent dimensions was varied from 40 up to 80 and 60 latent dimensions was selected because it resulted in the lowest validation loss, see Figure 8.1. To lower the time required for the whole process of the VAE, random subsets of the dataset are used with a lower amount of samples.

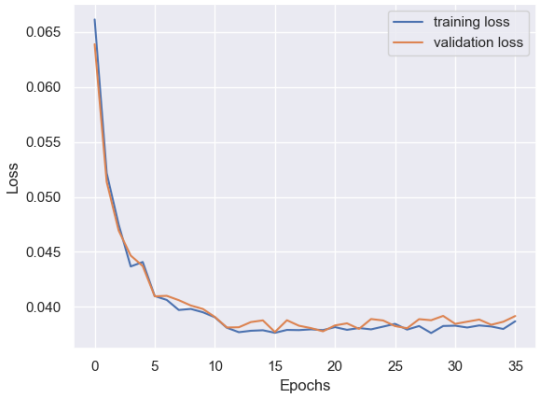


Figure 8.1: Training progress, 13 Modules, 30,000 random samples, best v-loss: 0.03772

Every module that is used and not available is counted and the total is multiplied with the `stock_weight` and added to the score. The linear penalty function is used adding the value above the normalized constraints to the score function multiplied with a `constraint_penalty_weight`. For this case the elastic energy is minimized making the complete performance function:

$$score = energy + constraint\_weight * constraint\_penalty + stock\_weight * stock\_penalty \tag{8.3}$$

8.4.1. Comparing different methods to the dataset

First both the GD optimizer and the Random sampling method are compared to the dataset and after that the combination of the GD optimizer with initial sampling is made. For the first case the stock from Figure 8.2 is used.

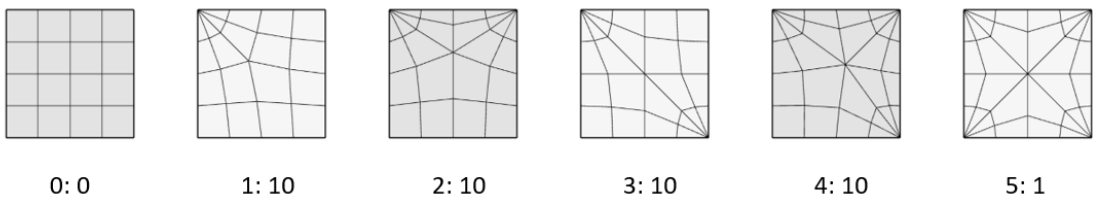
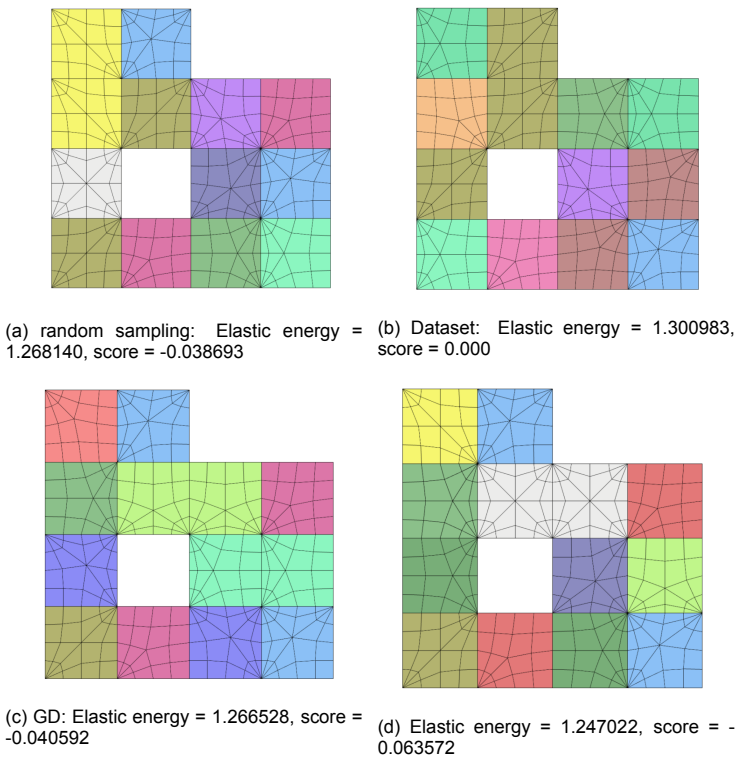


Figure 8.2: Stock with the module id followed by the number of available modules

In this case the best score from the dataset is 0. This means that the sample with the lowest elastic

energy in the dataset, fulfills the constraints of the performance metrics and of the stock constraints. This result is already very good and will be difficult to beat, because there is no information in the dataset how to make a configuration with a lower elastic energy. Both the GD and random sampling optimizer are used. With the GD optimizer 300 samples were generated optimizing them for 400 steps with a learning rate of 0.005. The resulting floor plans with the elastic energy and there score are shown in Figure 8.3.

Both optimizers beat the dataset score and lower the elastic energy by about 3 percent, fulfilling all constraints. Figure 8.3d already uses the combined optimizer with initial sampling and lowers the score and elastic energy even further. In the next section this method is compared with the evolutionary solver



(a) random sampling: Elastic energy = 1.268140, score = -0.038693

(b) Dataset: Elastic energy = 1.300983, score = 0.000

(c) GD: Elastic energy = 1.266528, score = -0.040592

(d) Elastic energy = 1.247022, score = -0.063572

Figure 8.3

8.4.2. Benchmark results

Because the modules with one pole and two poles next to each other are used frequently those are restricted more in the problem that will be compared with the evolutionary solver. Figure 8.4 shows the available stock. The best sample from the dataset does not have a score of 0 anymore, meaning that the sample with the lowest elastic energy does not fulfill the constraints and does not result in the lowest score. This makes it more interesting to look for a better solution because now the VAE should be able to come up with lower elastic energy samples and optimize towards those that also fulfill the stock constraints.

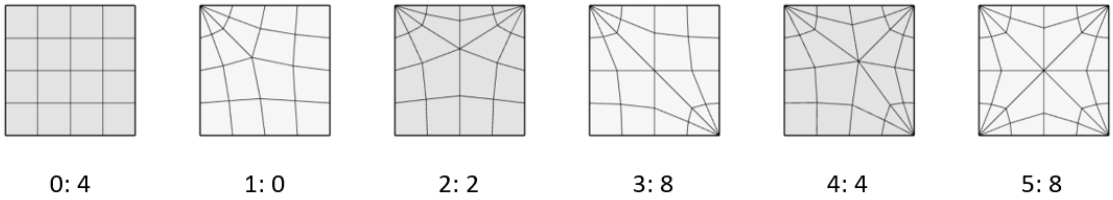
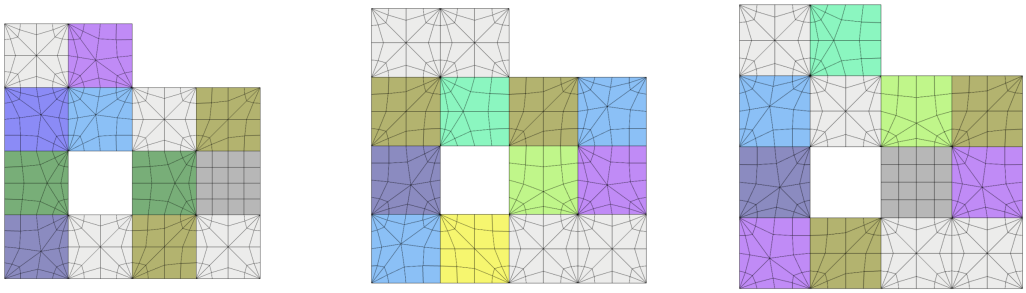


Figure 8.4: Available stock per module

The random sampling method runs again to generate 300 best samples out of 10,000 each time. It finds a solution sticking to the constraints of the performance metrics and of the stock. The GD optimizer also finds a solution that is better than the dataset, but not as good as the random sampling result, see Figure 8.5



(a) random sampling: Elastic energy = 1.427588, score = 0.149155

(b) Dataset: Elastic energy = 1.456656, score = 0.183401

(c) GD: Elastic energy = 1.436862, score = 0.160081

Figure 8.5

To improve the result the random sampling method and GD method are combined into one new optimizer. First, for every run a large number of initial samples are generated. These are decoded and the score is calculated with the performance score. Then only the best sample of each run is optimized with the same method as the GD optimizer from before. Then the best result of every run is exported to GH and calculated. Due to the large number of initial samples only samples satisfying the stock constraints from the start are optimized and the others are thrown away. Running this method with 300 runs, 10,000 initial samples per run, optimizing for 400 steps with a learning rate of 0.005, took about 12 minutes and results in the best score and lowest elastic energy as shown in Figure 8.6. For one attempt to improve the result further the number of initial samples was increased to 100,000 taking an hour to run, but the same best result was found as with 10,000 samples.

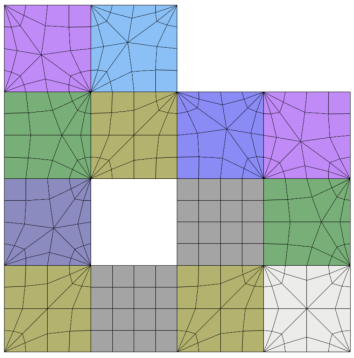


Figure 8.6: Elastic energy = 1.331942, score = 0.136473

The simplicity of the evolutionary solver is that everything is performed in one step. No separate dataset generation, training or verification of results is needed. The algorithm to count the modules used and the stock violations used in the python script is transferred to GH to calculate the performance score. The resulting score can be used as the value to minimize by the Galapagos evolutionary solver in GH.

For the VAE results the GD optimizer with initial sampling is used optimizing the best 300 samples of 10,000 initial samples with 400 steps and a learning rate of 0.01 and no early stopping. The time it takes is mainly dependent on the dataset generation and optimization processes. Because the optimization happens with normalized constraints and metrics, the results depend on the normalization process and can therefore be different when different datasets lead to different normalizations. With 10,000 samples the chances of finding samples with higher or lower elastic energy are higher which and as the minimum value means a zero score the needed elastic energy for this zero score can be lower than with a 1000 sample dataset. To give a better picture of the results the stock violations and the de-normalized elastic energy are included in the results as well. The performance score of the evolutionary solver is calculated with the normalization values of the 10,000 samples dataset.

In earlier chapters often the best 5 samples were shown to get some insight in how depended the optimization process is on randomness/luck. To quantize this more this stock constraint problem was optimized repeatedly with both the VAE and the evolutionary solver. The best results are given together with the mean of 10 runs. The complete results are in Appendix C

Table 8.4 shows the time it takes for both optimization processes and the results. The stock violations average, is the average number of modules violating the stock constraints per run. Running the evolutionary solver for 30 minutes and 3 hours was only done once. Because the VAE optimizer with 2000 samples was behaving well the GD optimizer with initial sampling was also tried with only 100 samples instead of 300 to decrease the optimization time and see if it still works well. Although the evolutionary solver is good in minimizing the elastic energy quickly, running it longer does not result in lower elastic compared to the VAE. It takes more time for the evolutionary solver to produce results that stick to the stock constraints, compared to the VAE only using the available stock except when a much lower elastic energy is found making it worth the extra penalty violating the stock constraints. For this specific problem it can be concluded that the VAE optimizer with initial sampling behaves better independent on how much time is available to run the optimization.

Of course, the evolutionary solver does not require the dataset generation time, but to make a performance score with normalized performance metrics some kind of dataset is still needed to get the normalization values. That makes it difficult to conclude if for a quick and approximate result the evolutionary solver could be faster and easier to use for this problem. But if one wants to find a solution that is a few percent better than that, it is worth to use the VAE. Also if it should be possible to change the stock constraints quickly and frequently due to changing stock the VAE can be the better method to choose as it does not require to redo the dataset generation and training steps in these cases, meaning it can be worth it to invest more time in these processes. This way to invest to gain a small improvement cannot really be done with the evolutionary solver, which does not seem to improve anymore after a certain time. However it seems there is a limit to this ability to invest in a larger dataset as using the full 30,000 sample dataset does not results in any improvement. Note that the score is also higher due

to different normalization values due to more extremes in the dataset.

Table 8.4: Benchmarking the Evolutionary solver with stock constraints

Samples	Dataset generation	Training	Optimization	Verification	Elastic Energy [kNm]	Stock violations average	score average
1000	8 min	14 s	10 min	2 min	1.36	0.4	0.0675
2000	16 min	30 s	10 min	2 min	1.34	0.5	0.0506
2000, 100	16 min	30 s	3 min 45s	1 min	1.36	0.7	0.0844
5000	40 min	1 min	10 min	2 min	1.36	0.6	0.0864
10,000	80 min	3 min	15 min	2 min	1.34	0.3	0.0479
30,000	240 min	3 min	12 min	2 min	1.34	0.6	0.1066
EVO	10 runs		3 min		1.44	2.2	0.3661
EVO	10 runs		10 min		1.38	2.3	0.2489
EVO	1 run		30 min		1.47	0	0.1676
EVO	1 run		3 hours		1.44	0	0.1289

Running both optimizers for 10 minutes the VAE leads to a on average 1.5% lower elastic energy and including the stock penalties a 20% lower score.

8.4.3. Optimizing mass

With the same datasets, trained models and optimizer, the mass can also be optimized with stock constraints. Only this time using the normalized mass the performance score simply changes to:

$$score = mass + constraint\_weight * constraint\_penalty + stock\_weight * stock\_penalty \quad (8.4)$$

This time not a complete sensitivity analysis is performed but a few different settings are tried with the optimizer. With the 2000 sample dataset only 100 samples are optimized in parallel to lower the optimization time as much as possible. The 10,000 dataset is used optimizing 300 samples in parallel to try to get the best result in around 10 minutes, which was tried twice. The EVO solver was used three times running it for 3 minutes and one time for 10 minutes. All the results are in Table 8.5. The results for the evolutionary solver and the VAE optimizer are quite similar and not one method seems to perform consistently better than the other in this specific case.

Table 8.5: Benchmarking the Evolutionary solver with stock constraints

Samples	Penalty weight	Stock weight	Optimization	Mass ton/m^2	Stock violations	score
2000, 100	0.1	0.1	3 min 35s	0.26	2	0.4030
2000, 100	0.1	0.2	3 min 45s	0.27	1	0.5547
10,000, 300, 1	0.1	0.1	13 min 20s	0.26	1	0.4289
10,000, 300, 2	0.1	0.1	15 min 15s	0.26	2	0.4134
EVO			3 min	0.26	2	0.4480
EVO			3 min	0.26	1	0.3150
EVO			3 min	0.26	2	0.5115
EVO			10 min	0.24	5	0.4200

8.5. Optimizing for Embodied Carbon

To incorporate embodied carbon (EC) into the optimization process with stock penalties, reliable data on EC values is required. In this work, the Inventory of Carbon and Energy (ICE) database (9) is used as the primary source for material EC. For processing steps and transportation, emission factors (EF) are applied, expressed as the equivalent kilograms of CO_2 ($\mathrm{kg\ CO_2e}$) emitted per unit of activity.

8.5.1. Embodied Carbon of New and Reused Modules

The EC of producing a new module can be expressed as:

$$EC_{\text{new}} = (\text{material mass} \times \text{material EF}) + \text{processing} + \text{transport}. \tag{8.5}$$

In contrast, reusing an existing module avoids most of the material and processing emissions, but still requires:

$$EC_{\text{reuse}} = \text{inspection} + \text{cleaning} + \text{refurbishment} + \text{transport}. \tag{8.6}$$

The *marginal carbon gap* is defined as the additional embodied carbon incurred when opting for a newly manufactured module instead of reusing one:

$$\text{Marginal Carbon Gap} = EC_{\text{new}} - EC_{\text{reuse}}. \tag{8.7}$$

This marginal gap represents the penalty that should be applied in the performance score when reuse is possible but not selected.

8.5.2. Material and Transport Emission Factors

The EC of concrete elements is strongly influenced by the cementitious content of the mix. Many alternative mixes can achieve the same strength class with different cement replacements, resulting in significantly lower EC values. In Table 8.6 three options from the ICE database (9) of reinforced concrete with approximately the same strength class are compared. The worst scoring mixture uses ordinary Portland cement with a cementitious content of $380\ \mathrm{kg/m^3}$. The PFA (Pulverized Fuel Ash), also known as fly ash and the GGBS (Ground Granulated Blast-Furnace Slag) mixtures both have a strength class of C32/40 according to the database.

Table 8.6: Carbon emission factors for reinforced concrete

OPC	167 $\mathrm{kgCO_2e/ton}$
30% PFA	134 $\mathrm{kgCO_2e/ton}$
70% GGBS	87 $\mathrm{kgCO_2e/ton}$

When integrating the carbon penalty with other design constraints, the EC-based penalty must be normalized to ensure it is weighted appropriately relative to other performance criteria.

For the transport of precast concrete modules, Table 8.7 presents typical emission factors for vehicles capable of carrying up to two tonnes, based on data from (34). In a future reuse-oriented market, detached floor modules will likely be transported to centralized storage and reconditioning facilities. The transport distance for reused elements may be shorter if multiple demolition sites and projects are located within the same region. However, if opportunities for reuse are infrequent, transportation distances to new sites could be substantially longer than for new modules sourced directly from a manufacturing plant.

Table 8.7: Carbon emission factors for transportation

Gasoline	0.344 $\mathrm{kgCO_2e/(t \cdot km)}$
Diesel	0.286 $\mathrm{kgCO_2e/(t \cdot km)}$
EV	0.239 $\mathrm{kgCO_2e/(t \cdot km)}$

In accordance with EN standards, the applied allocation method is the *cut-off approach*, meaning that all production-related emission flows are attributed to the first life cycle of the elements (1).

The EF for the processing, inspection, cleaning and refurbishment steps are difficult to estimate so for this example the very arbitrary estimate of 0.02 $\mathrm{kgCO_2e/ton}$ is used. Different scenarios are assessed with the given EF for the material and the transportation.

8.5.3. scenario Definition

A number of scenarios is set up that varies the material, the transport type and the transport distances for new and reused modules.

Table 8.8: Emission factors and distances for different scenario's

Scenario	Material EF ($kgCO_2e$/ton)	Transport EF ($kgCO_2e$/(t·km))	d_{new} [km]	d_{reuse} [km]
S1	OPC (167)	Gasoline (0.344)	200	50
S2	30% PFA (134	Diesel (0.286)	100	50
S3	30% PFA (134	EV (0.239)	100	50
S4	70% GGBS (87)	EV (0.239)	50	200
S5	70% GGBS (87)	EV (0.239)	50	20

8.5.4. Workflow Integration

Two pathways are possible depending on the optimization scope:

1. **Optimization without structural constraints:** In this case, only embodied carbon is optimized. Random bitmaps can be generated directly in Python, bypassing Grasshopper dataset generation. EC is then computed for each configuration using the module stock and the scenario-specific EF values.
2. **Optimization with structural constraints:** When structural performance must be considered, existing datasets with structural metrics are required. The EC is then integrated within the performance score rather than pre-computed in the dataset. This prevents scenario-dependent datasets and allows flexible evaluation under multiple carbon scenarios.

8.5.5. Optimizing with structural constraints

When structural constraints are included the datasets with the structural performance metrics are needed. The embodied carbon can be either included in the datasets as well calculating the values for every sample before training or it can be calculated inside the performance score as was done until now with the stock constraints. When it is included before training the datasets become scenario dependent, so to prevent that the calculation will be performed in the performance score.

The problem definition from section 8.4 is used and the same 30,000 random samples dataset is used as well. The stock from Figure 8.4 is used.

The EC is calculated with the `calculate_embodied_carbon` function. With a given bitmap, available modules and the EC per ton for both reusing and new modules the total EC can be calculated. It is similar to the stock constraint functions but instead of keeping track of penalties modules violating the stock limits are added to the new modules and the modules withing the stock limits are saved as reused modules. The mass of these modules can than be calculated and multiplied with the EC for both reusing and using new modules coming to a total EC.

The `calc_ec_per_ton` function calculates EC's per ton for new and reused modules. The input for this function is the scenario data that should be used for this specific optimization problem. The scenarios from Table 8.8 are used. For reusing a module 0.02 $\mathrm{kgCO_2e/ton}$ is added to the EC. The maximum and minimum total EC for the different scenario's can be found in Table 8.9.

Table 8.9: Minimum and maximum embodied carbon in dataset for different scenario's

Scenario	EC min ($kgCO_2e$)	EC max ($kgCO_2e$)
S1	148.24	766.57
S2	136.69	533.86
S3	127.32	517.60
S4	266.61	367.77
S5	98.75	327.25

Inside the VAE the `calculate_embodied_carbon` function is used inside the performance function. The total EC is normalized using the minimum and maximum values from Table 8.9. The optimizer minimizes the EC and penalties from the performance constraints are added to the normalized EC with a penalty weight of 0.1.

For the VAE results the GD optimizer with initial sampling is used optimizing the best 300 samples of 10,000 initial samples with 400 steps and a learning rate of 0.01 and no early stopping.

Table 8.10 shows the results from the VAE optimizer versus the 30,000 samples dataset.

Table 8.10: Embodied carbon improvement found with VAE

Scenario	EC dataset ($kgCO_2e$)	EC VAE ($kgCO_2e$)	improvement %
S1	148.24	144.82	2.3
S2	136.69	132.60	3.0
S3	127.32	124.38	2.3
S4	266.61	266.05	0.2
S5	98.75	95.30	3.5

The available reuse stock levels are halved and doubled to gain some further insight. With double the stock the VAE behaves similar. For three of the five scenarios the EC is improved by 1-2%, see Table 8.11. For the other two scenarios the resulting EC is just above the best of the dataset. Halving the stock the VAE struggles to find good solutions and the generated samples are usually far of from the dataset best, see Table 8.12.

Table 8.11: Comparison of EC witch double the stock

Scenario	$EC_{dataset}$ [$kgCO_2e$]	EC_{VAE} [$kgCO_2e$]	Improvement [%]
S1	141.03	141.17	-
S2	130.03	131.68	-
S3	121.12	119.97	0.9
S4	257.03	252.01	2.0
S5	93.94	92.95	1.1

Table 8.12: Comparison of EC with half the stock

Scenario	$EC_{dataset}$ [$kgCO_2e$]	EC_{VAE} [$kgCO_2e$]	Improvement [%]
S1	214.20	259.42	-
S2	180.83	204.44	-
S3	170.40	221.51	-
S4	288.98	288.98	-
S5	125.01	156.20	-

8.5.6. Pareto Front: Embodied Carbon vs Structural Performance

In addition to scenario-based optimization, the problem can be framed as a multi-objective optimization task. Two competing objectives are considered:

- Minimization of embodied carbon (EC), expressed in $\mathrm{kg\ CO_2e}$,
- Minimization of elastic energy, representing the structural performance objective.

These two objectives are inherently conflicting: reducing EC often requires increasing reuse, which may limit the structural performance due to constraints in available stock; conversely, minimizing elastic energy may require selecting new modules with higher cement content, leading to higher EC.

The trade-off between these objectives can be visualized using a Pareto front. Each point in the plot corresponds to a feasible design generated either from the dataset or by the VAE optimizer. The Pareto front highlights the non-dominated solutions, i.e., designs for which no other solution is strictly better in both objectives simultaneously. Figure 8.7 shows the Pareto front for the whole 30,000 samples dataset.

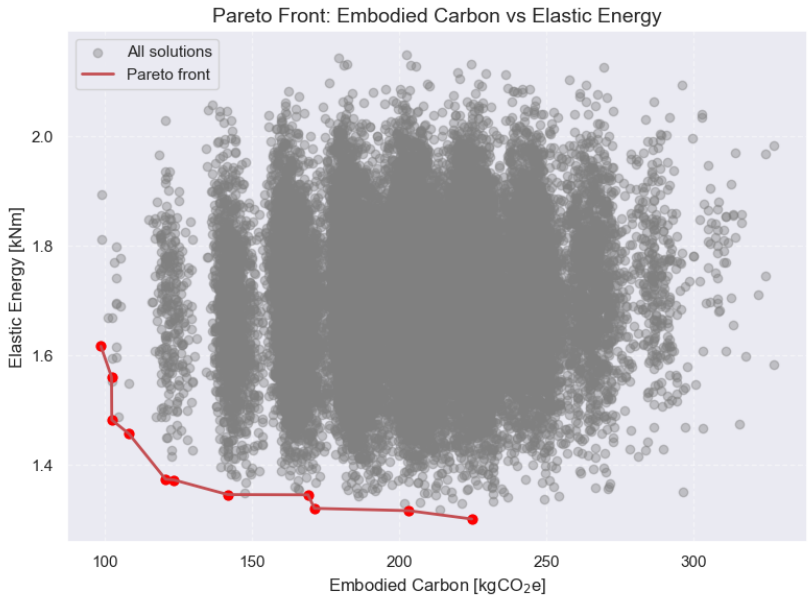


Figure 8.7: Pareto front of Embodied Carbon (EC) vs Elastic Energy. Gray dots represent all sampled solutions, while the red line indicates the Pareto front.

This representation allows decision-makers to select an appropriate compromise depending on project priorities. For example, one might choose a low-carbon solution with a small sacrifice in stiffness, or alternatively, a high-performance solution with slightly higher EC. Such trade-off analysis supports a more transparent and balanced decision-making process compared to single-objective optimization. The visualization can help make an informed decision instead of relying on a hidden scoring mechanism to come to a best solution between conflicting objectives. Figure 8.8 shows the Pareto Front for all 5 scenarios in one figure, which enables to see what the increase of elastic energy or embodied carbon could be when the scenario changes.

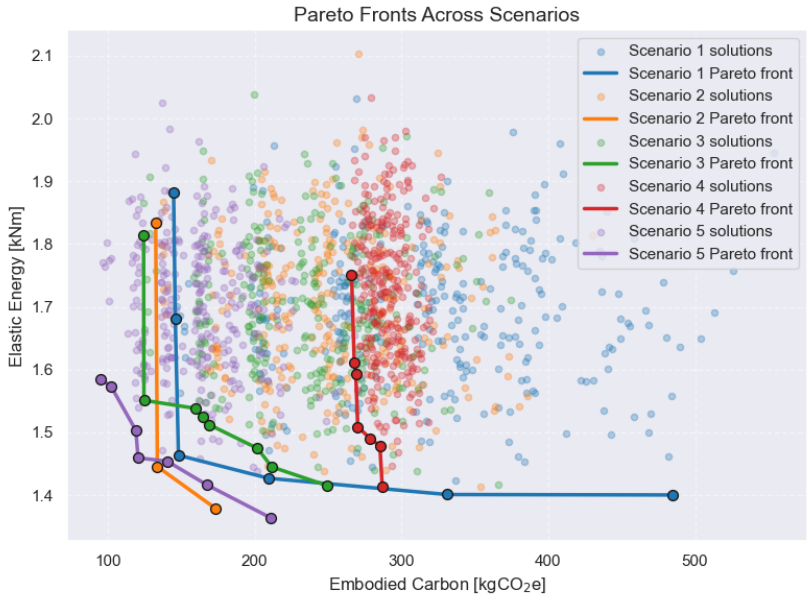


Figure 8.8: Pareto front of Embodied Carbon (EC) vs Elastic Energy. Different colours represent VAE samples for different scenarios

9

Problem Generalization and Extensions

The developed method to solve the combinatorial problem still has some drawbacks that could be overcome by a further generalization. The method now depends on one single cross-section per problem which allows an optimization of mass given this cross section, but choosing a smaller cross section or a mixture of cross sections could of course have a much larger impact. In this chapter some suggestions are made how the inclusion of multiple cross sections in a problem can be made, but is not implemented besides some minor tests.

Maybe the largest question is if the VAE can be generalized so far that it can work for every given floor plan without generating a new dataset every time. A start was made by optimizing the column and wall placement and suggestions are given how this full generalization could potentially work.

9.1. Including multiple cross sections

Until now in every problem one cross section was selected and kept the same throughout the problem. Because of different spans within floor plans it could be more efficient to use multiple different cross sections. Due to simplification in the structural model it is complicated to include this, but some ideas were already considered in chapter 3.

The generation of new samples with the VAE works if the cross section is added as extra bitmap, so 0 or 1 with two cross sections. or [00, 01, 10, 11] with four cross sections. Now the question is when this is practical to use.

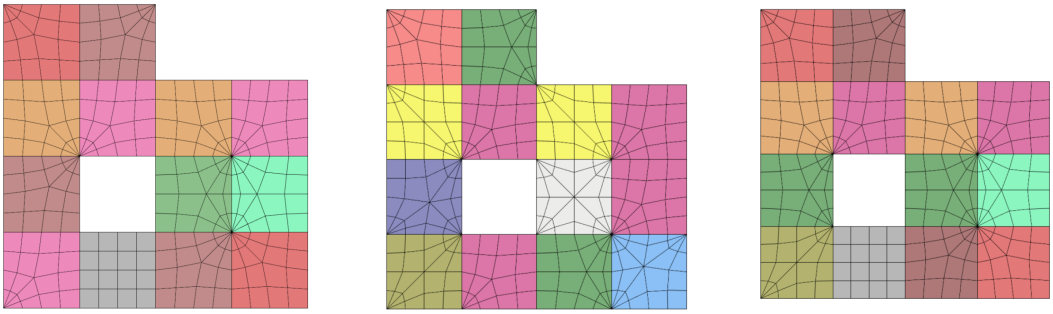
It could make more sense to include multiple cross-sections within one floor. But then difficulties with compatibility arise. What does the jump in cross section do with shear forces? How would this connection work? And what to do with middle and edge beams of modules?

Furthermore, the constraints are now defined as a maximum moment, shear force and/or displacement. The maximum displacement is still valid, but the moment and shear force not. They are given as a numerical value depending on the maximum reinforcement allowed in the cross section. If multiple cross sections are involved one numerical value as constraint does not work anymore. Only the maximum values of the moment and shear force are stored in the dataset and not the location, so with this model it is not possible to know in which cross section these forces occur. Because a change in the model is needed to reach a useful result the inclusion of multiple cross sections is not implemented any further in this thesis.

One lesson learned here that when including more input parameters in the VAE as long as it is represented by binary values the VAE recognizes this and although using continuous values in the generation of samples they stick very close to the binary values with only a small clipping error. When cross sections were for example represented by integers [1, 2, 3, 4] or the cross section in cm [10, 20, 30, 40] the VAE does not understand this when training and generating new samples. So the rule for the use of this VAE when including more input parameters is to use a bitmap, so a combination of 0's and 1's for discrete parameters and for continuous parameters a normalized value between 0 and 1, in this case being the performance metrics.

9.2. Multiple Load Cases

In this section the best results from the paper minimizing the elastic energy of the 13 module floor plan are used to to compare with the VAE. Note that in the paper all modules had equal weight and now they have equal cross sections resulting in modules with more poles to be heavier. The optimizer with initial sampling is used minimizing only for elastic energy with 300 samples, 10,000 initial samples per sample, 400 steps and a learning rate of 0.005. The VAE results in a new best result lowering the elastic energy from $1.283\ kNm$ to $1.242\ kNm$. combining insights from both results some manual configurations were tried and one lowered the elastic energy further to $1.235\ kNm$. The resulting floor plans are in Figure 9.1



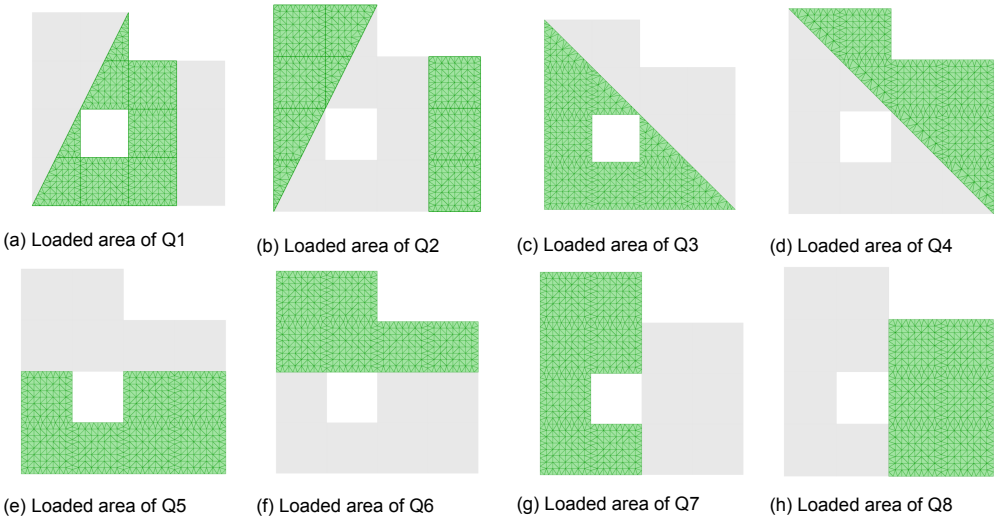
(a) Manual paper: elastic energy: 1.283301 kNm, displacement 10.698659 mm

(b) Best VAE without constraints: elastic energy: 1.241696 kNm, displacement: 9.394274 mm

(c) Manual after VAE: elastic energy: 1.235033 kNm, displacement: 10.047501 mm

Figure 9.1

Like in the paper (24) it can also be assessed what the effect of different load cases is on the elastic energy. The same $Q = 5\ kN/m^2$ is placed on two different areas as seen in Figure 9.2.



(a) Loaded area of Q1

(b) Loaded area of Q2

(c) Loaded area of Q3

(d) Loaded area of Q4

(e) Loaded area of Q5

(f) Loaded area of Q6

(g) Loaded area of Q7

(h) Loaded area of Q8

Figure 9.2

The same load combinations as earlier are used:

$$ULS1 = 1.35 * G + 1.5 * Q1 \tag{9.1}$$

$$ULS2 = 1.35 * G + 1.5 * Q2 \tag{9.2}$$

$$ULS3 = 1.35 * G + 1.5 * Q3 \tag{9.3}$$

$$ULS4 = 1.35 * G + 1.5 * Q4 \tag{9.4}$$

Table 9.1 shows the resulting elastic energy for the three different load cases for the four floor plans, now also including the grid modules design. For the regular load case the manual configuration created after seeing the results of the VAE had the lowest elastic energy, but when considering the asymmetric load cases the configuration from the VAE behaves slightly better.

Table 9.1: Elastic energy for different load cases in kNm

Load case	Manual paper	Best VAE	Manual after VAE	Grid
G + Q	1.283	1.242	**1.235**	1.989
G + Q1	0.931	**0.920**	0.922	1.217
G + Q2	1.046	**0.983**	1.024	1.417
G + Q3	0.994	**0.942**	0.947	1.485
G + Q4	0.526	0.538	**0.520**	0.793
G + Q5	0.848	**0.819**	0.826	1.292
G + Q6	0.642	0.638	**0.622**	0.937
G + Q7	0.924	0.887	**0.878**	1.364
G + Q8	**0.442**	0.466	**0.442**	0.733
Mass [ton/m2]	0.202	0.223	0.206	0.178

9.3. Optimizing column and wall placement

The placement of columns and walls is usually defined by architectural and project requirements, for example demanding an open floor plan. In this section the goal is to create a model that can help design the combination of column and wall placement with the configuration of the floor modules to come to an optimal design. To reach this goal the VAE is trained including random configurations of columns, walls and floor modules. When generating new samples the number and/or location of columns and walls can be guided to restrictions or wishes. The VAE is trained on one dataset with the idea that the scoring can be quickly adapted. The floor plan itself and the cross sections are kept fixed using the 3x3 symmetric floor plan with the 30 cm by 15 cm cross section with C30/37 concrete.

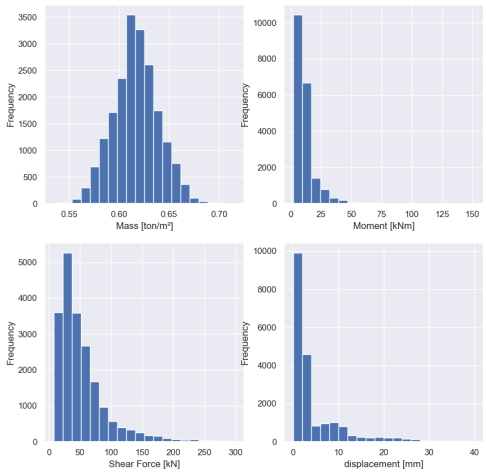
9.3.1. Dataset generation with random columns and walls

In a 3x3 symmetric floor plan there are 16 possible placements of columns and 24 for the walls. Defining the placement of a column or a wall in a bitmap, results in 40 bits with (0) being no column or wall and (1) being a column or a wall. A dataset with random configuration of these 40 bits can easily be generated, but some refinement is needed for the dataset to make sense. In the GH model the first 16 bits are linked to the 16 possible column positions and the last 24 bits to those of the walls.

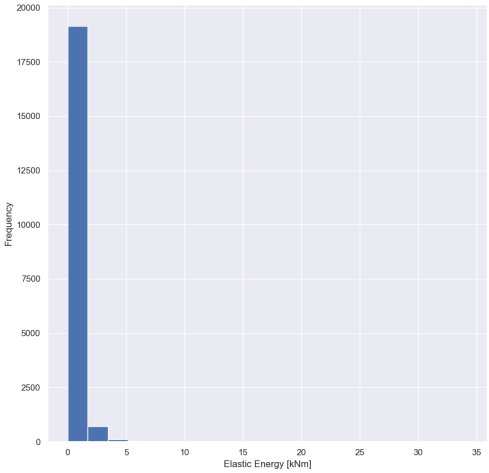
Columns and walls can easily share the same position. This causes an error in Karamba3D stating that two supports are placed in the same location. The easy solution to make the model work is to remove duplicate points before the supports are created, to remove these columns placed in walls, but then the bitmap still includes these columns. A better way to solve this issue is to prevent columns to be at wall locations in the generation process already. This can be done in the python script in GH that generates the random bitmaps, by linking the number of columns to those of walls that correspond and setting the bit of the column to zero if both equal one. The duplicate points still have to be removed afterwards for walls that connect.

A probability of generating a 0 or a 1 is added to the script as well to influence the average amount of columns and walls created. The probability of generating a column or wall is set to 0.2. For 16 column and 24 wall positions this means that on average about 3 columns and 5 walls are generated. With the removal of columns placed in the same position of walls the amount of columns will be lower.

Another issue is that some generated support configurations are just not feasible, for example only columns at one edge of the floor plan, causing very large forces and deformations. These samples can break the VAE, due to normalization issues with extremely large values. Because there is no need to train and generate these kind of samples they can easily be removed before the training process by setting a maximum for all performance metrics and for elastic energy also a minimum value of zero. This results in a 20,000 sample dataset represented by the histograms in Figure 9.3.



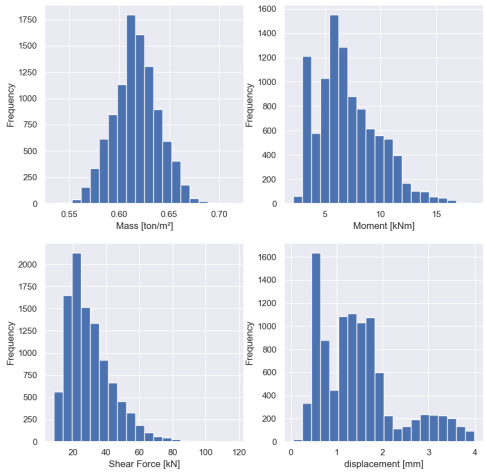
(a)



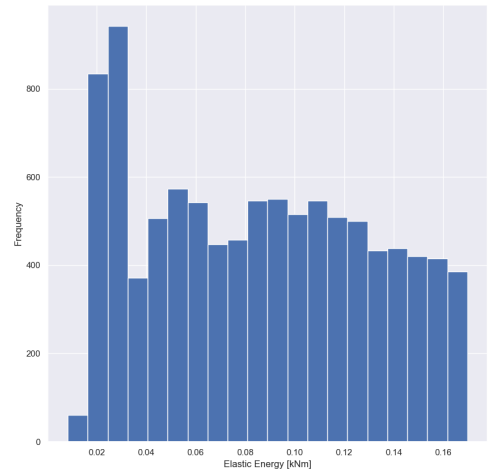
(b)

Figure 9.3

A lot of the samples still contain quite large values for the displacement and elastic energy. Although not visual in the histograms due to the low frequency there are a few samples close to the set maximum that influence mainly the normalization. The dataset can be made smaller removing these samples to see if the performance improves. A 10,000 sample subset from the 20,000 random dataset is made by removing all samples with a displacement larger than 4 mm and after that taking the 10,000 samples with the lowest elastic energy to result in Figure 9.4.



(a)



(b)

Figure 9.4: Histograms from the 10,000 samples case

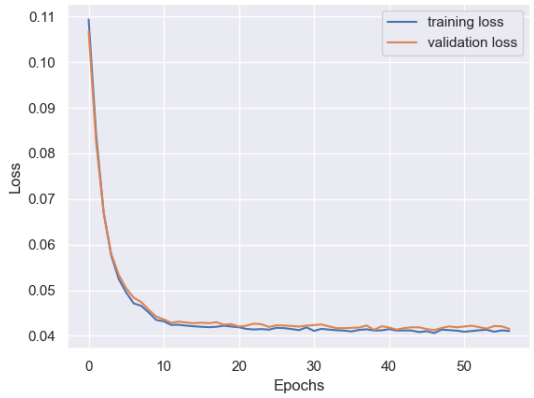
9.3.2. VAE training

For the 3x3 case the 16 bits for the columns, 24 for the walls and 36 for the modules results in a total of 76 bits and 5 performance metrics. First the dataset with 20,000 samples is used to train the model. With the lessons learned from the previous chapters only metrics based training is performed

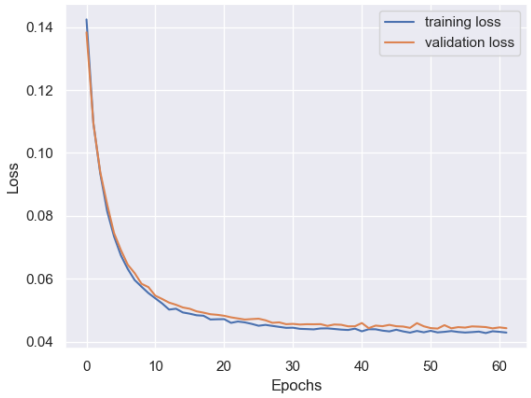
and because the problem is different, a different number of latent dimensions is tried as shown in Table 9.2. For each case 200 hidden units, 200 epochs with patience of 10 and a kld-loss of 5e-4 was used. Figure 9.5a shows the training progress.

Table 9.2: Training the random column and wall placement

Latent dimensions	Training time	Early stopping	best v-loss
30	1 min	18 epochs	0.10227
50	4 min	91 epochs	0.045179
70	2 min	36 epochs	0.041694
75	3 min	57 epochs	0.041260
80	1.5 min	34 epochs	0.041631
120	2 min	28 epochs	0.041907



(a) Training progress 20,000 random samples, best v-loss = 0.04126



(b) Training progress 10,000 random samples, best v-loss = 0.04415

Figure 9.5

The 10,000 sample dataset results in a slightly higher validation loss, see Figure 9.5b

9.3.3. Performance score

Some constraints are set that in practice would come from the project requirements. No constraints are set on the moment, shear force and displacement this time. The elastic energy or mass needs to be minimized and at the same time we want to control the amount of columns and walls. The performance function must be minimizing elastic energy or mass and adding penalties for the amount of columns. Inside the performance function the new modes `columns_and_walls_min_energy` and `columns_and_walls_min_mass` are defined that call the new `count_columns_and_walls` function and sums the elastic energy or mass times a weight with the penalty for the supports times a weight. The `count_columns_and_walls` function simply counts the amount of columns and walls for every sample and returns a penalty. The walls can be given a higher penalty than the columns to penalize these more. Important is to give a maximum amount of columns and/or walls to allow without giving a penalty. Without this, the VAE is pushed to create samples without supports, still predicting good scores although the samples are unfeasible when running them in GH. Furthermore samples with less then 2 supports are given an extra penalty, which is now set to 3.

The random sampling method is used with 200 runs of 10,000 samples to assess how the performance function is behaving for different numbers of columns and walls and different penalties. Table 9.4 shows that the VAE is generating samples that are close to the best dataset result but not yet better performing ones.

The best dataset result has two columns and two walls, see Figure 9.6. Although causing a penalty when restricting the problem to 2 columns and 1 wall, this then still remains the best scoring sample. The score depends on which weights are used. The elastic energy of this sample is 0.018947 kNm.

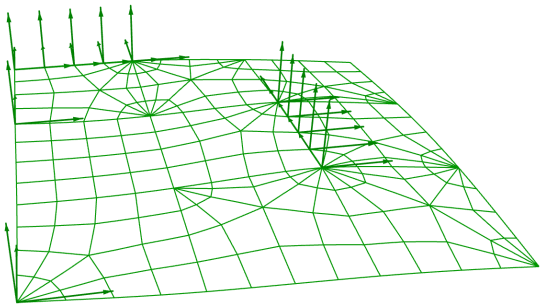


Figure 9.6: Columns, wall and module placement for best performing dataset sample

Table 9.3: Results 20,000 samples dataset

Columns	Walls	Energy weight	support weight	Elastic Energy kNm	Score, dataset
3	2	0.1	0.1	0.258529	0.000733, 0.000032
3	2	1.0	1.0	0.270114	0.007666, 0.000321
2	1	0.1	0.1	0.438340	0.001259, 0.000032
2	1	1.0	1.0	0.418008	0.011992, 0.000321

9.3.4. GD optimizer

The GD optimizer with initial sampling is used to see if it is suitable to generate and optimize better than yet known samples. Because the samples with insufficient supports are removed from the training process, the VAE has no information on the problems this causes and the GD is encouraged to optimize towards samples with insufficient supports. Because including these invalid samples causes problems with normalization and training they are still left out and the problem is dealt with in the generation and optimization process. When generating initial samples for the GD a check is done if there are minimal two supports and if not the samples are removed. In the optimization process an extra early stopping criteria is enforced if a samples has less than two supports.

The optimizer is set to use 10,000 initial samples, optimize 200 samples with 400 steps and a learning rate of 0.005, taking between 2 and 3 minutes to run. Increasing the learning rate to 0.01 or 0.1 results in the early stopping for less than two supports being triggered very quickly and results in samples with higher elastic energy and scores.

Table 9.4: Results 20,000 samples dataset

Columns	Walls	Energy weight	support weight	Elastic Energy kNm	Score, dataset
3	2	0.1	0.1	0.189829	0.000532, 0.000032
3	2	1.0	1.0	0.258529	0.007327, 0.000321
2	1	0.1	0.1	0.296106	0.000843, 0.000032
2	1	1.0	1.0	0.325058	0.009273, 0.000321

A VAE workflow was successfully constructed that can generate new samples including the placement of columns and walls. The optimizer with initial sampling results in better samples than the random sampling method, but does not optimize to better results than the dataset yet. For this thesis no more time is invested to research this specific problem further, but some conceptual ideas for a more overall generalization are given.

9.4. VAE optimizer independent of floor plan

With the workflow in place for the optimization of column and wall placement the step to a complete problem generalization could be made to train a VAE independent of the floor plan that is wished. This

model could then be used to optimize every floor plan in certain bounds removing the need for training the model for different problems. Of course this leads to a larger amount of input data for the VAE and as was seen with trying a very large 225 module problem this would likely lead to overfitting using the given architecture. That is also why no attempts are made to try this within this thesis but ideas are shared as recommendations for future research.

A grid size should be selected which will be the maximum size the floor plan can be. Within this grid modules can be placed or not, which has to be encoded in the bitmap.

Infeasible designs have to be prevented or removed by implementing rules in the generation process. For example that every module should touch at least one other module.

The VAE optimizer could be pushed towards the wished floor plan by penalizing every module that should not be in the floor plan or is missing. The same can be done for the amount or location of columns and walls.

Because this generalization of the problem leads to so many more possible solutions it would be possible that it is already difficult to generate a corresponding and if the optimizer is then still able to also optimize the module configuration.

10 Conclusions

The main research question of this thesis was:

> "How can the combinatorial problem of modular ribbed floor systems be optimized using the latest optimization techniques, including generative machine learning?"

To address this, a number of sub-questions were posed that are answered below.

1. Which structural analysis objectives need to be optimized?
The performance based objectives mass and elastic energy can be optimized with the maximum moment, shear force and displacement being used as code based constraints.

2. Can an existing optimization workflow be adapted to optimize these objectives?
The idea to create the datasets by running the Karamba solver in Grasshopper came from previous thesis studies. The VAE methodology was adapted from the DSAIE course. Literature on using the bitmap as data representation of such a 3D geometry with structural information encoded into it was not found, meaning that to test different VAE architectures was pure trial and error. A shallow 1 layer model was already effective and adding more layers did not prove to be gaining any improvements. The number of latent dimensions most suitable for the training process appeared to be dependent on the problem size. Suitable values for other hyperparameters of the VAE were selected but did not appear to results in large differences between different problems and datasets.

3. What are the advantages and disadvantages of different optimization workflows?
The choice of methodology to solve this optimization problem partly came from the proposed type of input data representation and the optimization objectives. Using a Variational Autoencoder or Reinforcement Learning were both state-of-the-art methods that were successfully used for other structural optimization problems. The main advantage of using the VAE over RL was the prior knowledge available, and when a first script was working the RL method was abandoned.

4. Can a heuristic or generic method be used to create an initial dataset with configurations?
As dataset generation strategies the generic methods available in Grasshopper, being the evolutionary solver and simulated annealing solver are compared with random generated samples. It was expected that using generic solvers to create datasets would lead to better results than randomly created ones. Datasets created with the evolutionary solver can have very good predictive abilities and often result in lower validation losses than with random generated samples. The annealing solver and combined datasets of the two solvers lead to a larger prediction error in the shear force. When optimizing with these different datasets it becomes clear that the model trained with the random generated data is the best in optimizing below the results available in the dataset.

5. How many samples should a dataset have to sufficiently train the model
Increasing the number of samples decreases the clipping error, depending on the problem size on average from 0.9 for 2000 samples to 0.6 for 20,000 samples. Minimizing the elastic energy with constraints for the symmetric problems the 2,000 and 20,000 samples datasets produced similar results, with about half of the time either one outperforming the other.

6. How is the performance of a configuration evaluated?
The performance of a configuration is evaluated as the lowest score, if within the normalized dataset boundaries between 0 and 1 or otherwise slightly above or underneath it. The performance function defines which performance objective is used for this score. Other structural performance metrics can be included in the score as constraints allowing different functions to be set with different penalty functions and weights to tune how hard or soft constraints are penalized and thus enforced.

7. Can stock constraints be included in the optimization method?
Stock constraints were successfully implemented optimizing a performance score that minimizes mass or elastic energy adding penalties for using not available stock. These penalties can be used to enforce the constraints aiming to only use the available penalties, but could also act as the cost of producing a new module and therefore find a trade-off between more efficient designs with extra modules and less efficient designs with only available modules.

As the motivation for this floor system was the decrease of embodied carbon this was also set as an optimization objective, including the use of new and reused modules and the transportation. Although improvements were minor 1-3% and not always stable for every number of available modules, further work could make this a stable optimizer.

8. Can the chosen method be used to generate better configurations than solvers in GH?
The VAE workflow was found to produce better results in the 13 module case optimizing elastic energy than the evolutionary solver. It was quicker in finding solutions that do not violate the stock constraints and found designs with a lower elastic energy overall. Running both optimizers for 10 minutes the VAE leads to a on average 1.5% lower elastic energy and including the stock penalties a 20% lower score. Still, the VAE does not have the ability to follow some simple insights. For example when minimizing mass, why would the best result include the heaviest module when you can also fulfill the constraints without it? With some insight the best found configuration can often be improved manually.

9. Do the generated configurations only use the assigned modules?
The fact that the VAE architecture only allows for continuous values to be generated leads to the clipping of these continuous values to the desired bits. Often the generated values are very close to 0 or 1. As was expected this clipping error increases when the problem size increases and the validation loss of the model increases as well. When using the Gradient Descent optimizer the moment that the number of optimization steps does lead to larger prediction errors the clipping error was expected to increase as well. It would make sense that during ongoing optimization the VAE would try to make samples that are further away from the given bits to seek for extra improvement. However, in this case the clipping error decreases and the optimized mass stays about the same.

10. What kind of surrogate optimizer is best to include based on the optimization objectives?
Two different optimization strategies were considered, one drawing a large amount of initial samples from the VAE and the other optimizing a sample in latent space. A better but still quick working optimizer was set up by combining these two allowing to use a large amount of initial samples and only optimizing the best one in latent space.

11. What are the limits on the use of the method?
The validation loss keeps increasing with the problem size and starts to overfit. For smaller problems, 25 modules or less, it is difficult to see a negative effect on the optimization performance of the VAE. For 36 modules the VAE optimizer behaves similar as the evolutionary solver. Increasing the problem size further the training process becomes more unstable, but the predicting qualities of the VAE still remain. However, optimizing towards better samples than already in the dataset becomes very difficult.

12. Can the trained model optimize other floor plans?
No, but this could be possible by further generalizing the method. For example, the combinatorial problem can be further generalized with learning the VAE for example the placement of columns, walls, or different cross sections and implementing this in the workflow as long as it is represented by a bit input as well.

13. How can the method be used by others?
The method could be implemented in a GH workflow. All numerical experiments were now performed with the structural analysis and dataset generation in GH and the VAE training and optimization in Python in Visual studio Code, coupling the two with csv files with the datasets and optimization results. This was separated to allow for quicker and more stable editing and running of the python code compared to the Rhino python environment. The final python script could be split in two and implemented in GH. One part allows you to feed it with data, choose the training settings, train the model and save it. The other part lets you select a trained model, define all the optimization objectives and optimizer settings, and runs the optimizer. The last step would still be to calculate the actual performance of the optimized samples with the structural analysis in GH.

Contribution to Structural Optimization

The resulting optimizer can contribute most towards structural optimization when problems become more complex, when for example including stock constraints, including multiple load cases, or performing a multiobjective optimization with embodied carbon and elastic energy. In these cases it is difficult to find solutions by hand with only structural insight. For small problems structural insight sometimes leads to better designs than the VAE optimizer. This indicates that it understands the the structural behavior in a different way than engineers do. Combining the VAE designs with human knowledge on how the forces flow through a floor can lead to further improvements. The VAE optimizer in this case acts as a design tool that gives some almost optimized starting designs. The structural engineer is then able to tweak those designs to find an even better solution or include more design requirements or wishes the VAE was not able to implement.

11

Discussion and Future work

Structural simplifications

In this thesis the main focus was on the optimization workflow itself, and as noted at the scope definition and the introduction of the structural model, simplifications were made. The largest simplification is that the whole ribbed floor structure is modeled as completely fixed ribs, which is of course completely against the idea of a modular structure. Because the connections of a modular floor system are a whole research subject by itself, this simplification is left in place and this subject is for further research. A bolted or otherwise connected system would lead to a different structural model as a whole. These connections would be subject to shear forces and moments that need additional code checks. The whole structure would become less stiff leading to larger displacements.

Discrete vs. continuous VAE

Although discussed, a discrete VAE was not implemented but clipping the generated values of the samples to bits was used. When a sample is optimized and one or more values of a bitmap are not close to 0 or 1 but around 0.5 this could mean the VAE is steering towards a combination of modules. One could investigate if in these cases the it would make sense to use a rib design that is in between the two modules. A discrete VAE is not used entirely in this thesis so the performance of such a method is still unknown. It was decided not to use it because this simpler method was working and not investigating it further allowed for more time for numerical experiments. However it could be possible that a discrete VAE performs better if the clipping errors are not something to deal with anymore. Implementing a discrete VAE was seen as complicated, because the input data has a discrete, the bitmap, and a continuous part, the performance metrics. The coupling of these two parts with a different VAE architecture will be a challenge, that was now avoided.

Prediction errors

With almost all datasets it was possible to train the VAE model with prediction errors below 10% for all performance metrics. For mass, elastic energy and displacement the VAE often could make predictions for new configurations with an average 1-2% error. For the moment and the shear force this was usually in the 5-10% error range, with sometimes predicting the shear force much worse: 15-20% error. This indicates that the moment and shear force are more difficult to learn from the dataset, which makes sense. The relationship between the module bitmaps and the mass is the easiest to understand and learn. When the VAE learns that certain combinations of 0's and 1's lead to a certain mass, it can easily predict the mass on a new configuration as that is the only factor influencing it. With the displacement and the elastic energy the relation already becomes more complicated as the location of the modules now also matters in some extend, such as placing more ribs towards a column or wall and less in the middle of a floor span. The moment and shear force however, is something largely influenced by local effects and therefore more difficult to predict. Placing a module with a pole at a column increases the amount of ribs from 2 to 5, decreasing the maximum shear force often by more than 50%, depending on the design. So, if the VAE misses the relationship that a module with a pole or not (0 or 1) is located at a column it can easily make an prediction error of over 50%. As the location of supports is not included

in the bitmap in any way (in the non generalized model at least) and the only information on the shear force the VAE receives is the maximum shear force of the whole floor, it is not that surprising that this often goes wrong and actually impressive that the VAE reaches a shear force prediction error below 10% for so many datasets.

The maximum moment on the other hand is mostly depended on the continuity within the rib design. If two or more modules connect with both being poles or not, the moment usually transfers smoothly. If for example on the edge of a floor, one module ends with a pole and the other not, this can lead to a large jump in the moment larger than the moment would be in the middle of the floor with only grid modules. Something else that happens is that columns or walls in the middle of continuous floors can lead to large hogging moments. The module configurations around these supports have a large influence in creating extreme values for these hogging moments that often also result in the maximum moment of the entire floor. When making module configurations manually one could already see that the design looks weird, or structurally nonlogical, but the VAE is not able to see this and relies on data only. If one small change of one bit can lead to such a peak in the maximum moment this also makes it more difficult to predict.

Possible solutions that might decrease the prediction error of the moment and the shear force and get more insight on the local effects are to include the moment and shear force of all ribs or nodes to the dataset, or to add the supports to the datasets as is done in the further generalization of the problem.

Normalization

The optimization process is dependent on the normalization of the datasets. As was discussed with the results of the stock constraints benchmark with the evolutionary solver in chapter 8, this can make it difficult to compare performance scores. The score depends on the minimum and maximum values of the mass or elastic energy and lower generated values result in a negative performance score. This makes it difficult to compare the performance score between the results generated from two different datasets and is also why the elastic energy or mass and the number of stock violations was reported.

Cross-sections

The multiple cross section problem is still something to deal with. When minimizing mass a problem has to be very close to the constraints for the optimization to make sense. The improvement in for example displacement by finding a better configuration is often quite small, so a grid solution should just be above this constraint for another configuration to fulfill it when minimizing mass.

Module optimization

The modules itself could still be further optimized. The module geometry exists of straight lines between nodes, although curved ribs could be more efficient. The curvature of ribs would be more complicated to model, leading to more elements and thus a longer computation time. Furthermore, the curvature of ribs leads to additional moments caused by the curved reinforcement, leading to additional code checks.

Another possible module optimization could be to change the cross-section within modules. A more efficient rib design would require an increase in cross-section height in the middle of floor spans to increase the stiffness and thus decrease the deflection. Or modules could be further optimized considering supports. Every module optimization that tackles a local structural problem will cause a trade-off between structural efficiency and mass-production and re-usability, by creating extra modules or making their use too specific.

Future work

In the last chapter a first step towards a generalization of the model was made by including the placement of columns and walls. The generalization could be taken further in training a larger model for more floor plans at once. The input dimensions within this model have to stay the same, but with for example creating a bitmap for not using a module different floor plans within a certain grid can be possible. If you can specify what your floor plan is in the optimizer it could maybe generate configurations for this floor plan. The following steps should be taken to perform this complete generalization:

- The inclusion of a module should be included. This could for example be done with including separate bits for including a module in that location of the floor (1) or not (0). Then the bitmaps for the specific modules can be added, and then the bits for the columns and walls.
- Dataset generation should enforce that all modules are connected and the supports are feasible. Rules have to be written in the script that generates the random bitmap samples, so that these samples are excluded before the structural analysis runs and breaks on these unfeasible samples. The model probably has to be large and varied enough to work.
- A different architecture might be needed to overcome the overfitting of the model. As the larger problem sizes already showed overfitting and the random column and wall placement model was not able to find improvements it is expected that the current architecture will not work, or at least will not generate better samples when optimizing. Maybe implementing a discrete approach for generating the bits could work.
- Just like the stock constraints similar functions can be used for the exact location of modules and walls to enforce a specific floor plan design.

The advantages of this complete problem generalization would be that the dataset generation and training steps of the workflow only have to be performed once and this model only has to be downloaded and loaded by anyone that wants to use it.

The expected disadvantages are that the prediction accuracy will be lower and that the optimization step takes longer and leads to less improvement than with a dedicated problem description and dataset.

Besides the problem generalization other contributions in future work could be to:

- Look at other materials. This is a very small change in the model. Only the material, the cross sections and constraints have to be altered. Datasets have to be generated again of course, but could all be done with control of the parameters in GH without changing the workflow.
- Develop this optimization method further into a design tool in GH that can give quicker and better results than the already implemented tools, without the need to perform so many steps as are now needed.

A

Hyperparameter optimization

Table A.1: VAE Hyperparameter tuning

Latent dimensions	Hidden layers	Hidden Units]	kld weight	val-loss
1	1	10	5e-04	0.194275
5	1	10	5e-04	0.139817
8	1	10	5e-04	0.110614
9	1	10	5e-04	0.122681
10	1	10	5e-04	0.121501
11	1	10	5e-04	0.123745
12	1	10	5e-04	**0.108412**
13	1	10	5e-04	0.109473
14	1	10	5e-04	0.122932
15	1	10	5e-04	0.107895
16	1	10	5e-04	0.122117
17	1	10	5e-04	0.122027
18	1	10	5e-04	0.140689
16	1	10	1	0.235572
12	2	10	5e-04	0.138333
12	5	10	5e-04	0.151289
12	1	100	5e-04	0.013624
12	1	150	5e-04	0.012475
12	1	180	5e-04	0.012889
12	1	200	5e-04	**0.012369**
12	1	220	5e-04	0.012749
12	1	250	5e-04	0.012740
12	1	300	5e-04	0.013744
12	2	100	5e-04	0.012924
12	3	100	5e-04	0.014973
12	5	100	5e-04	0.024666
1	1	10	5e-04	0.194275
1	2	10	5e-04	0.193491
1	3	10	5e-04	0.193748
1	4	10	5e-04	0.197853

Table A.2: VAE Hyperparameter tuning 13 Modules, 10 epochs

Latent dimensions	Hidden layers	Hidden Units]	kld weight	val-loss
1	1	10	5e-04	0.191867
5	1	10	5e-04	0.181720
8	1	10	5e-04	0.177527
9	1	10	5e-04	0.178251
10	1	10	5e-04	0.174662
11	1	10	5e-04	0.174027
12	1	10	5e-04	0.174921
13	1	10	5e-04	**0.172009**
14	1	10	5e-04	0.173989
15	1	10	5e-04	0.175810
16	1	10	5e-04	0.173495
17	1	10	5e-04	0.175149
18	1	10	5e-04	0.175456
12	1	100	5e-04	0.140774
13	1	100	5e-04	0.137901
13	1	200	5e-04	0.136302
13	2	200	5e-04	0.134024

Table A.3: VAE Hyperparameter tuning 13 Modules, 50 epochs

Latent dimensions	Hidden layers	Hidden Units]	kld weight	val-loss
1	1	10	5e-04	0.188156
13	1	10	5e-04	**0.162463**
13	2	10	5e-04	0.168605
13	3	10	5e-04	0.175633
1	1	100	5e-04	0.185763
13	1	100	5e-04	0.111289
13	1	200	5e-04	0.100471
13	1	300	5e-04	0.100400
13	2	10	5e-04	
13	2	100	5e-04	
13	2	200	5e-04	

B

VAE predicting structural behavior

Table B.1: Prediction error 2x2 datasets

2x2	Mass	Moment	Shear Force	Displacement	Elastic Energy	v-loss
Evolutionary 10000	2.3%	4.6%	8.9%	1.7%	2.7%	0.013
Annealing 10000	2.1%	4.0%	12.2%	1.8%	2.6%	0.013
Combined 10000	2.1%	5.4%	7.7%	1.8%	3.1%	0.012
Combined 20000	2.4%	5.0%	7.7%	1.6%	2.6%	0.012
Brute Force 65536	2.1%	7.0%	4.6%	1.4%	2.3%	0.011

Table B.2: Prediction error 3x3 datasets

3x3	Mass	Moment	Shear Force	Displacement	Elastic Energy	v-loss
Evolutionary 5000	1.0%	6.4%	4.7%	1.0%	1.4%	0.025
Evolutionary 10000	1.1%	6.3%	6.2%	0.8%	1.3%	0.020
Annealing 10000	1.5%	4.7%	11.7%	1.1%	1.6%	0.037
Combined 10000	1.4%	5.7%	8.3%	1.2%	1.7%	0.039
Combined 20000	1.6%	5.4%	7.0%	1.0%	1.6%	0.031
Random 30000	1.6%	6.7%	4.2%	1.0%	1.5%	0.063
Random 2000(70lat)	2.0%	6.3%	6.6%	1.5%	2.3%	0.031
Random 5000 (70lat)	1.8%	7.1%	6.3%	1.4%	2.0%	0.028
Random 30000 (70lat)	2.0%	5.1%	5.9%	1.3%	2.0%	0.028

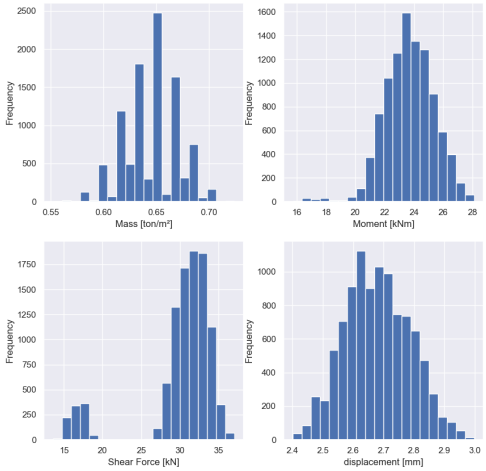
Table B.3: Prediction error 4x4 datasets

4x4	Mass	Moment	Shear Force	Displacement	Elastic Energy	v-loss
Evolutionary 20000	1.2%	4.0%	6.8%	1.0%	1.3%	0.045*
Annealing 20000	1.5%	5.8%	10.7%	2.6%	2.9%	0.039*
Combined 10000	1.4%	4.3%	10.4%	1.4%	1.7%	0.057
Combined 20000	1.4%	4.7%	9.9%	1.0%	1.3%	0.06*
Combined 30000	1.2%	4.5%	9.1%	1.1%	1.5%	0.05*
Combined 40000	1.4%	4.6%	8.8%	1.0%	1.4%	0.05*
Shear EVO 10000	1.0%	6.9%	8.8%	0.6%	1.0%	0.036
Random 10000	1.5%	5.5%	4.7%	1.1%	1.7%	0.18
Random 20000	1.6%	5.3%	5.8%	1.2%	1.8%	0.15
Random 20000 (70lat)	1.3%	5.6%	5.6%	1.2%	1.8%	0.046

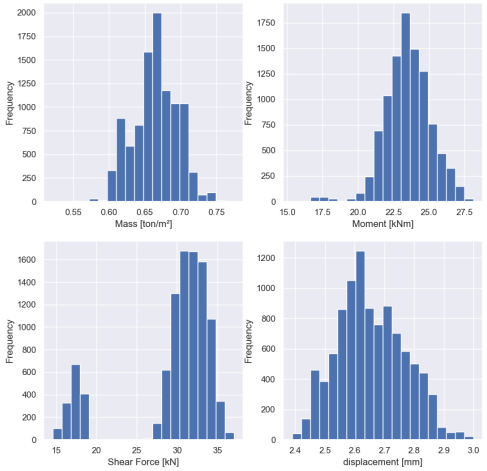
*Only the training graph is saved, not the exact validation loss

Table B.4: Prediction error 5x5 datasets

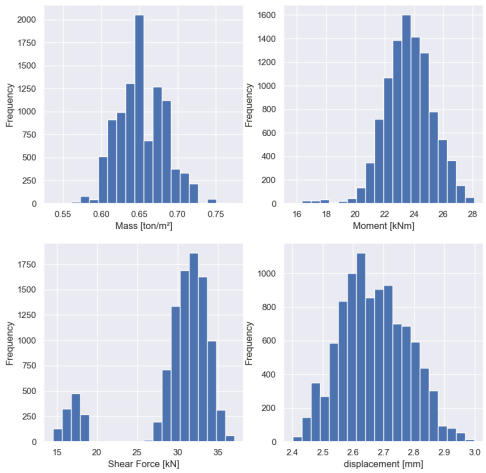
5x5	Mass	Moment	Shear Force	Displacement	Elastic Energy	v-loss
Evolutionary 8000	0.57%	2.9%	4.1%	0.6%	0.8%	0.046
Random 5000 (70lat)	1.4%	4.1%	6.3%	1.0%	1.5&	0.12
Random 10000 (70lat)	1.0%	4.2%	6.5%	0.8%	1.3&	0.12
Random 20000 (70 lat)	1.2%	5.0%	6.0%	0.9%	1.3%	0.12



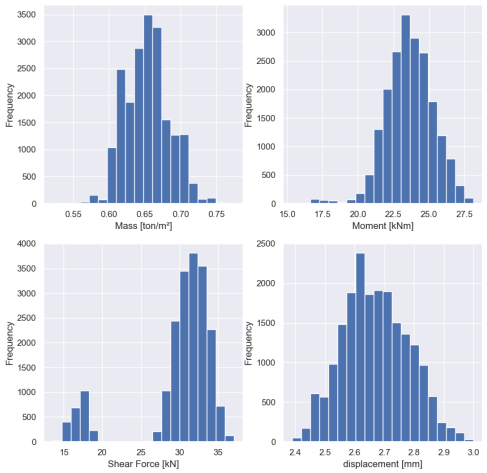
(a) Histogram 2x2, Evolutionary, 10000 samples (own work)



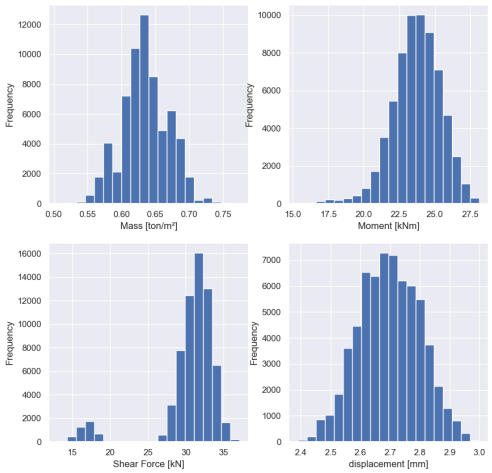
(b) Histogram 2x2, Annealing, 10000 samples (own work)



(c) Histogram 2x2, generic combined, 10000 samples (own work)

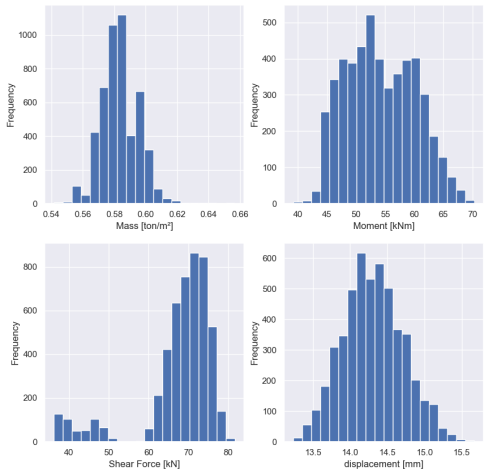


(d) Histogram 2x2, generic combined, 20000 samples (own work)

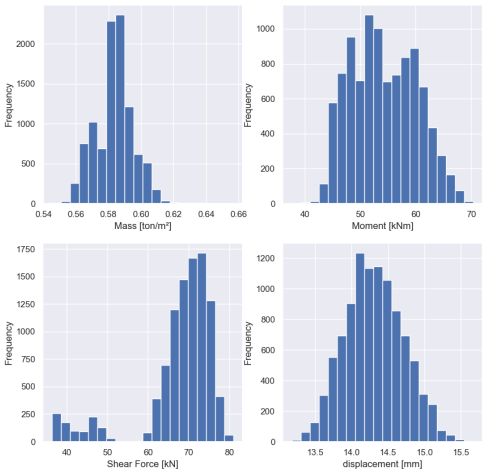


(e) Histogram 2x2, brute force, 65536 samples (own work)

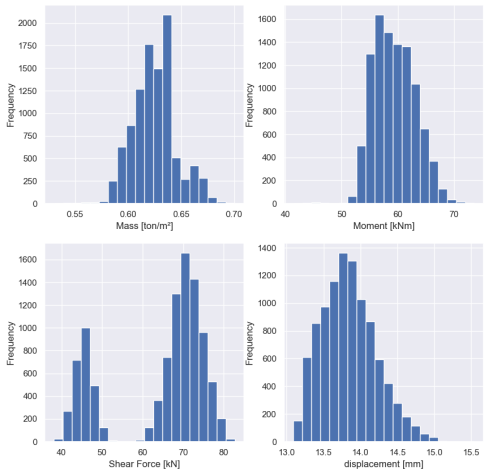
Figure B.1: Histograms of 2x2 case



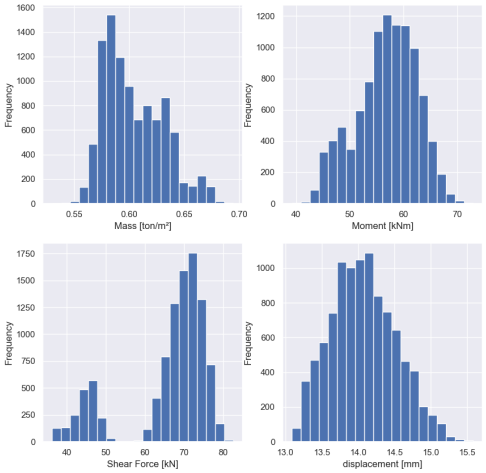
(a) Histogram 3x3, evolutionary, 5000 samples (own work)



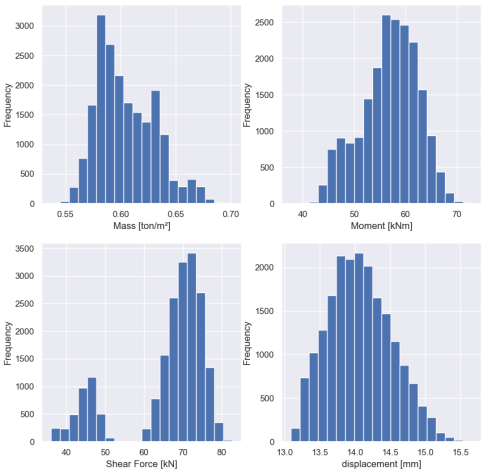
(b) Histogram 3x3, evolutionary, 10000 samples (own work)



(c) Histogram 3x3, annealing, 10000 samples (own work)

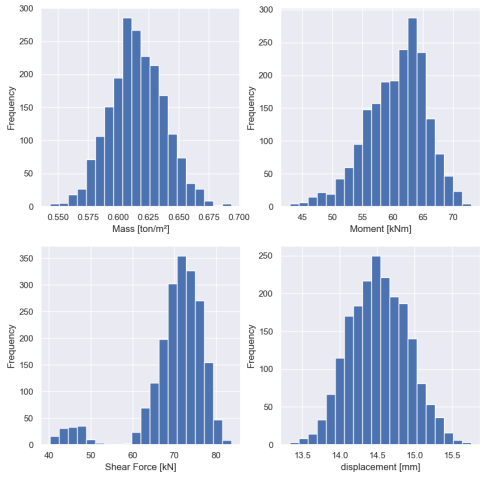


(d) Histogram 3x3, generic combined, 10000 samples (own work)

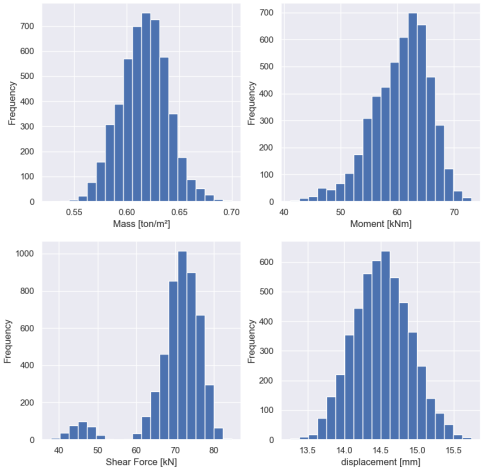


(e) Histogram 3x3, generic combined, 20000 samples (own work)

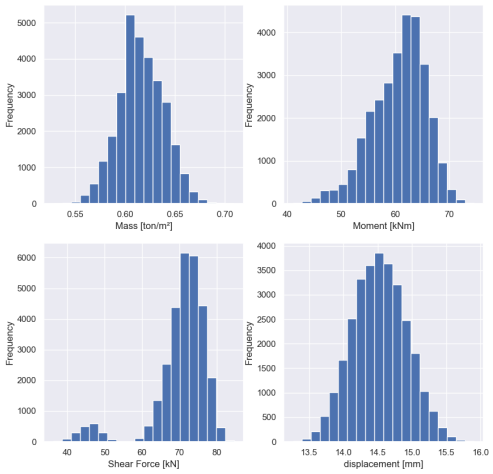
Figure B.2: Histograms of 3x3 case with generic datasets



(a) Histogram 3x3, Random, 2000 samples (own work)

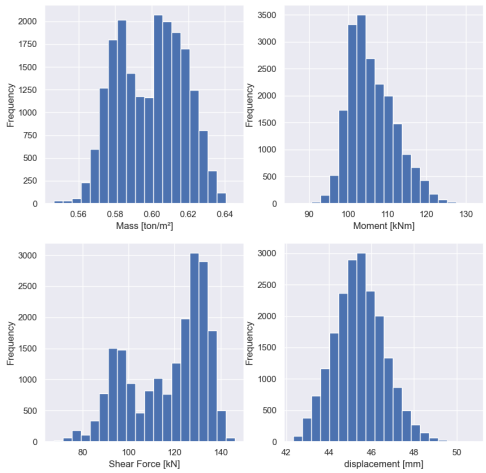


(b) Histogram 3x3, Random, 5000 samples (own work)

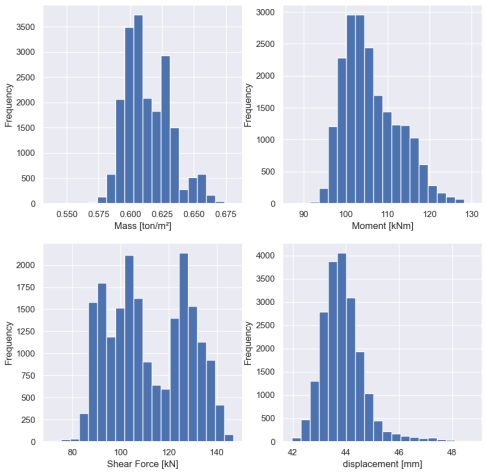


(c) Histogram 3x3, Random, 30000 samples (own work)

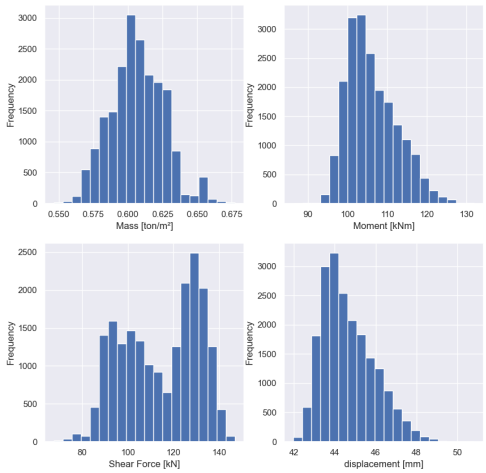
Figure B.3: Histograms of 3x3 case with random datasets



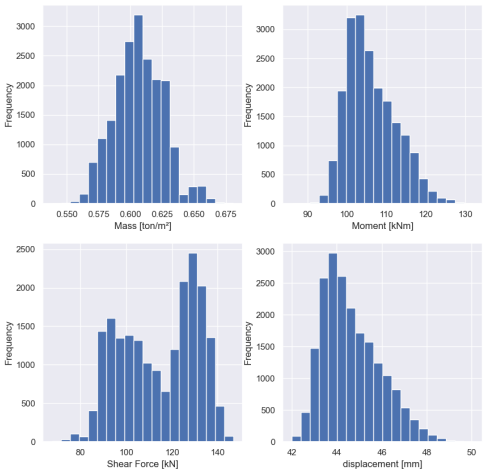
(a) Histogram 4x4, evolutionary, 20000 samples (own work)



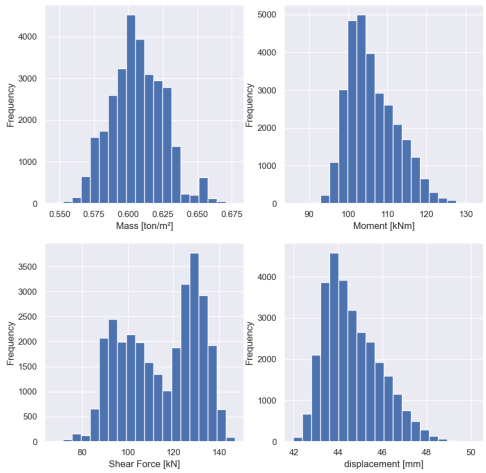
(b) Histogram 4x4, annealing, 20000 samples (own work)



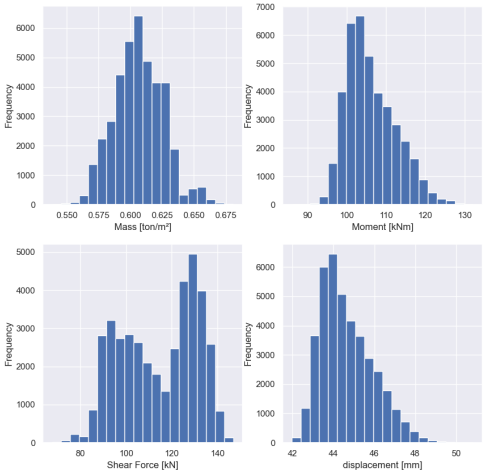
(c) Histogram 4x4, generic combined, 10000 samples (own work)



(d) Histogram 4x4, generic combined, 20000 samples (own work)

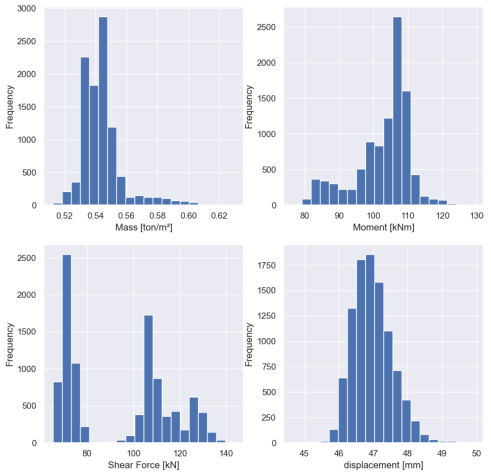


(e) Histogram 4x4, generic combined, 30000 samples (own work)

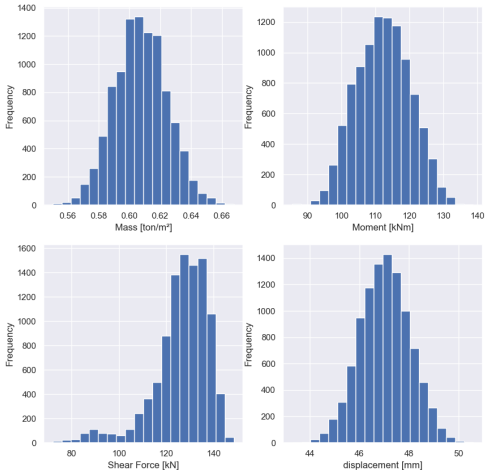


(f) Histogram 4x4, generic combined, 40000 samples (own work)

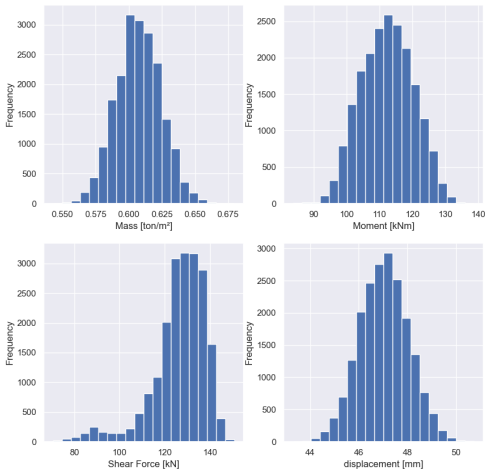
Figure B.4: Histograms of 4x4 case combining evolutionary and annealing



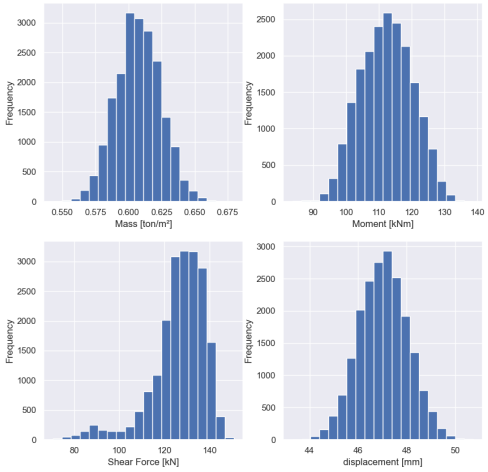
(a) Histogram 4x4, evolutionary shear, 10000 samples (own work)



(b) Histogram 4x4, random, 10000 samples (own work)

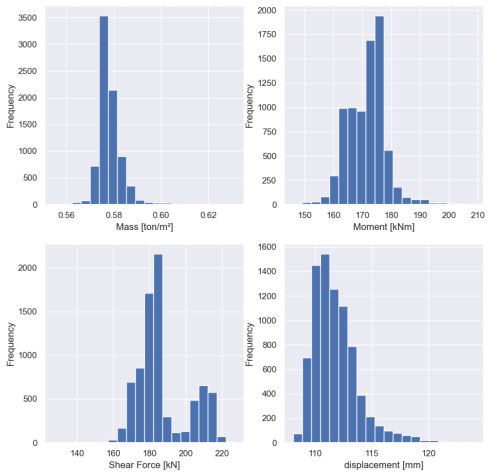


(c) Histogram 4x4, random, 20000 samples (own work)

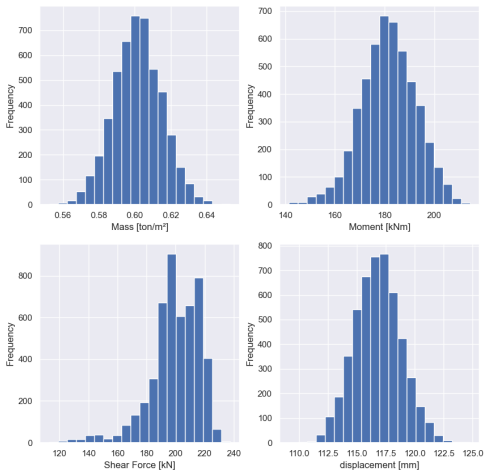


(d) Histogram 4x4, random, 70 latent dimensions, 20000 samples (own work)

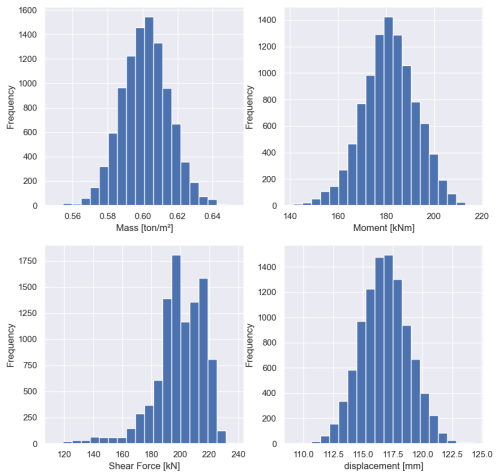
Figure B.5: Histograms of 4x4 case combining evolutionary and annealing



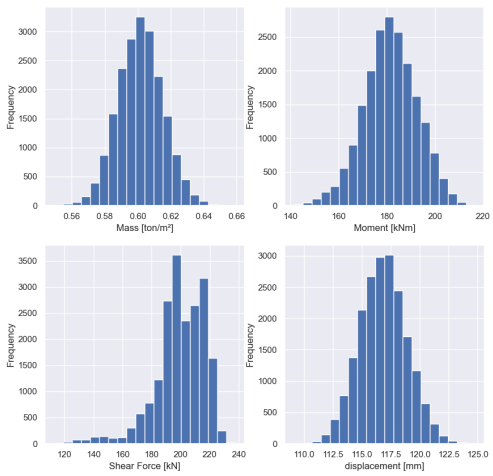
(a) Histogram 5x5, evolutionary, 8000 samples (own work)



(b) Histogram 5x5, random, 5000 samples (own work)

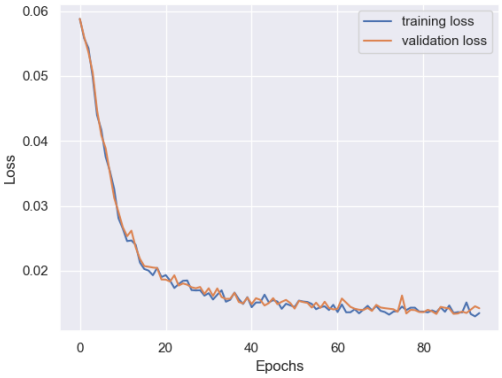


(c) Histogram 5x5, random, 10000 samples (own work)

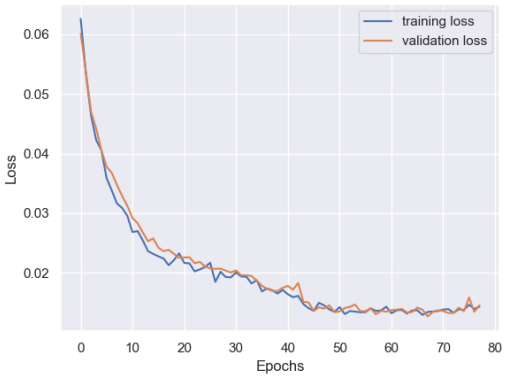


(d) Histogram 5x5, random, 20000 samples (own work)

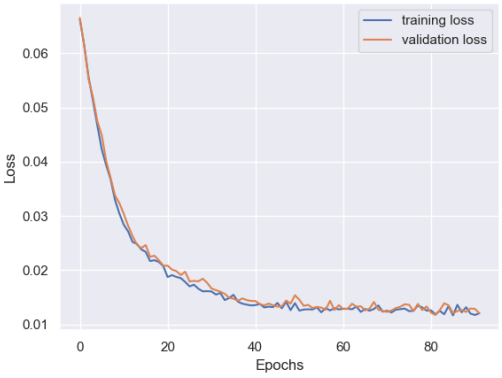
Figure B.6: Histograms of 5x5 case



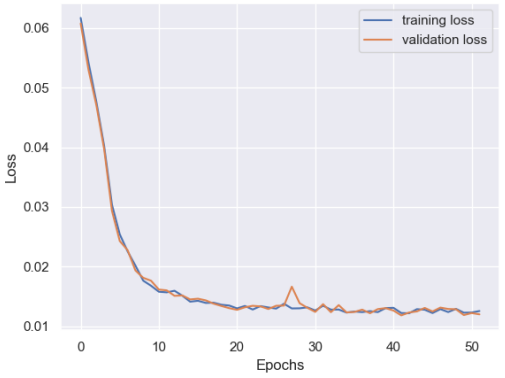
(a) Evolutionary, 10000 samples, best v-loss: 0.01336



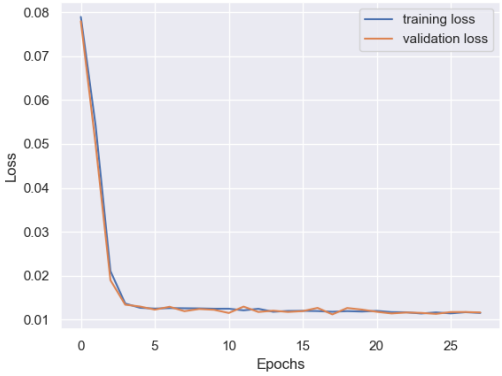
(b) Annealing, 10000 samples, best v-loss: 0.01274



(c) Generic combined, 10000 samples, best v-loss: 0.01175

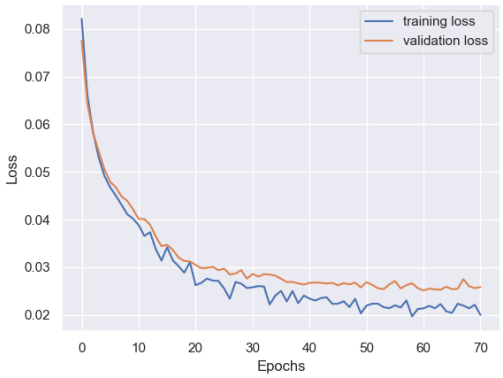


(d) Generic combined, 20000 samples, best v-loss: 0.01180

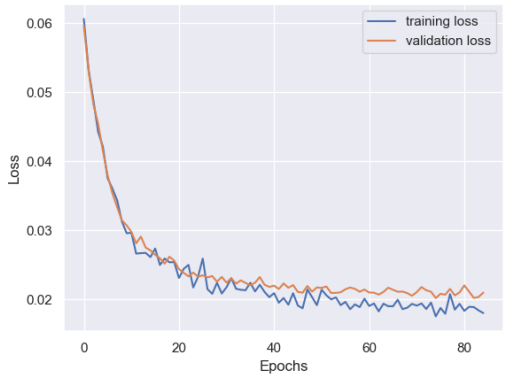


(e) Brute force, 65536 samples, best v-loss: 0.01116

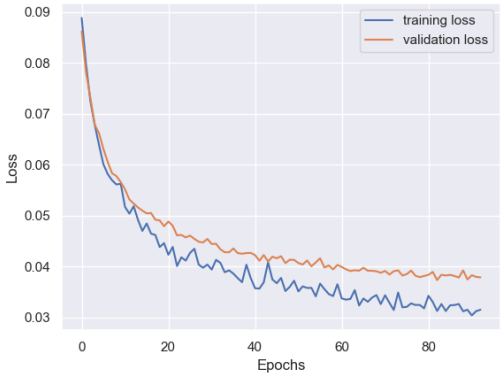
Figure B.7: Trainingprogress of 2x2 case



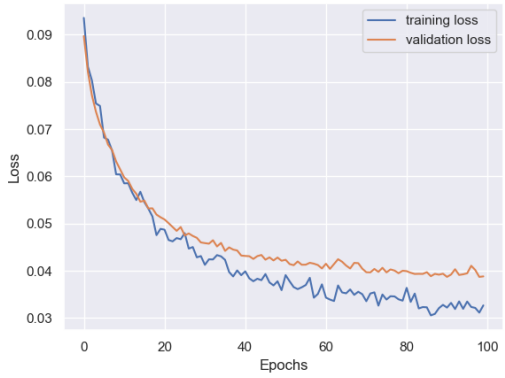
(a) Evolutionary, 5000 samples, best v-loss: 0.02513



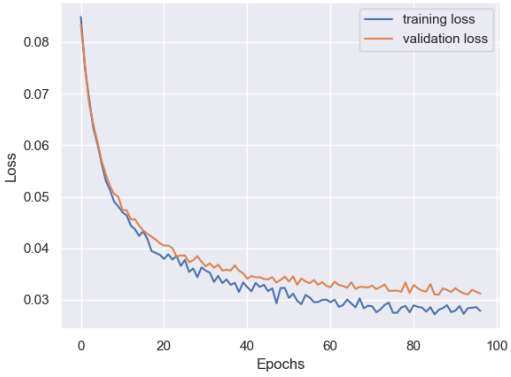
(b) Evolutionary, 10000 samples, best v-loss: 0.02020



(c) Annealing, 10000 samples, best v-loss: 0.03736

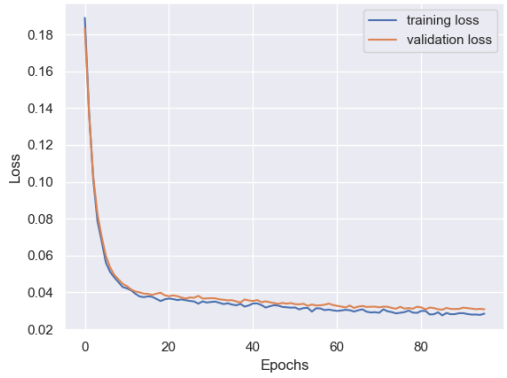


(d) Generic combined, 10000 samples, best v-loss: 0.03871

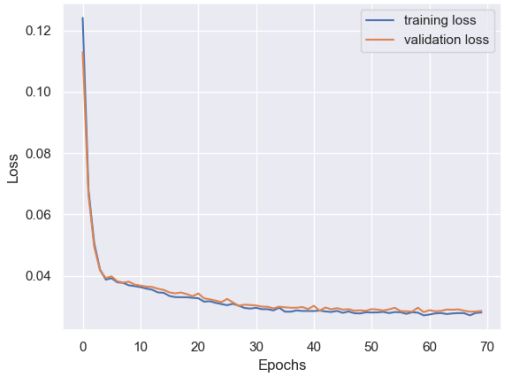


(e) Generic combined, 20000 samples, best v-loss: 0.03102

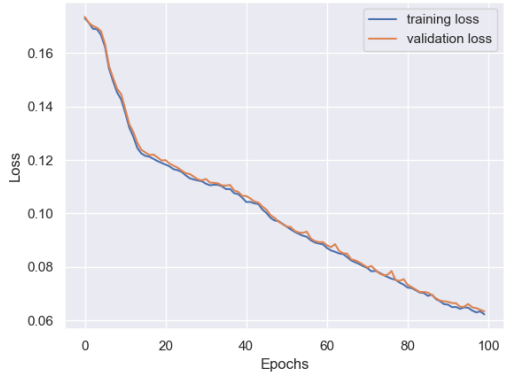
Figure B.8: Trainingprogress of 3x3 case with generic datasets



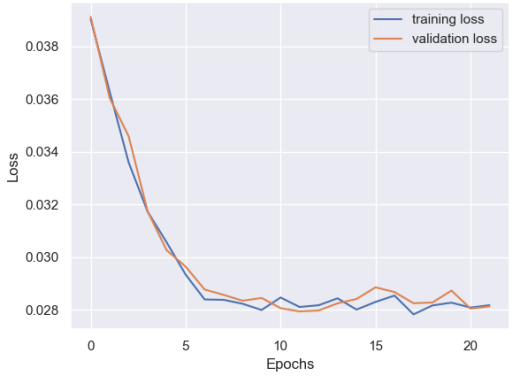
(a) Random, 2000 samples, best v-loss: 0.03063



(b) Random, 5000 samples, best v-loss: 0.02810

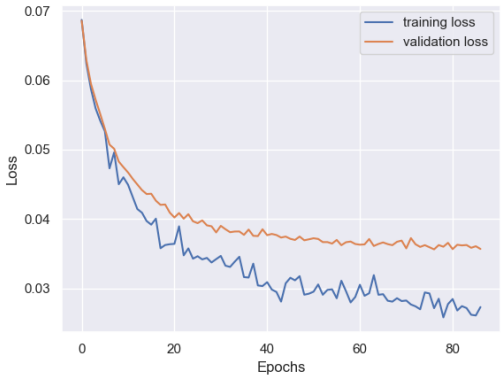


(c) Random, 30000 samples, best v-loss: 0.06334

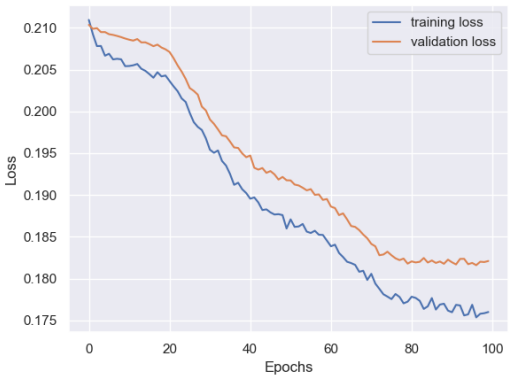


(d) Random 70 latent dimensions, 30000 samples, best v-loss: 0.02794

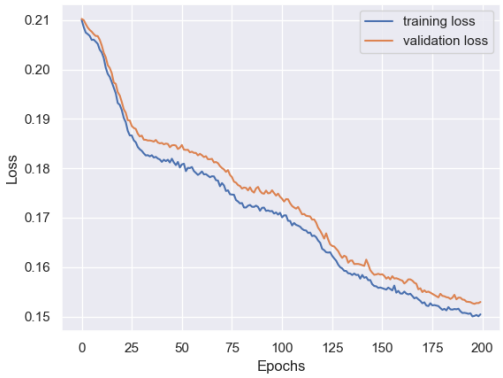
Figure B.9: Trainingprogress of 3x3 case with random datasets



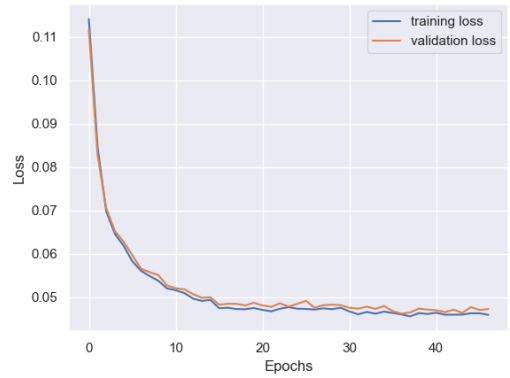
(a) Evolutionary shear, 10000 samples, best v-loss: 0.03563



(b) Random, 10000 samples, best v-loss: 0.18160

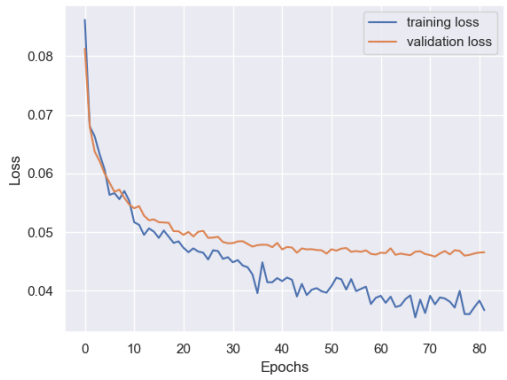


(c) Random, 20000 samples, best v-loss: 0.15256

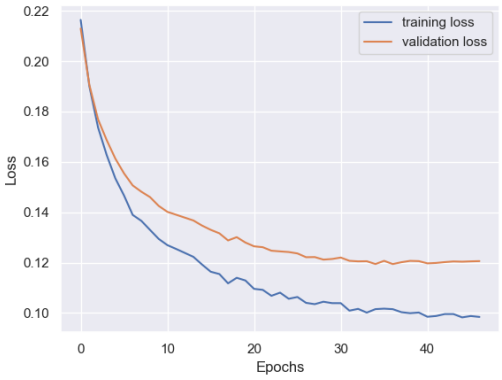


(d) Random 70 latent dimensions, 20000 samples, best v-loss: 0.04617

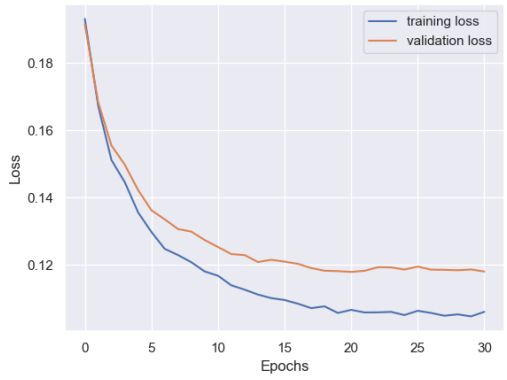
Figure B.10: Trainingprogress of 4x4 case



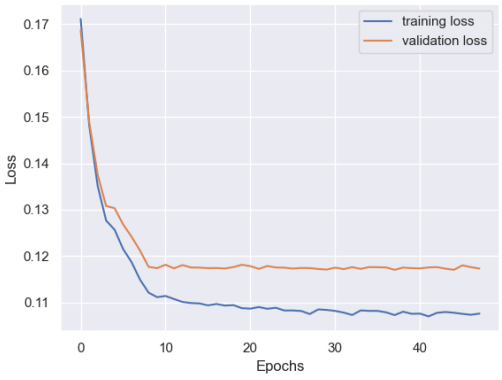
(a) Evolutionary shear, 10000 samples, best v-loss: 0.03563



(b) Random 70 latent dimensions, 5000 samples, best v-loss: 0.11944



(c) Random 70 latent dimensions, 10000 samples, best v-loss: 0.11792



(d) Random 70 latent dimensions, 20000 samples, best v-loss: 0.11706

Figure B.11: Trainingprogress of 5x5 case

C

Benchmarking evolutionary solver

Table C.1: Benchmarking the Evolutionary solver with stock constraints: 1000 samples

Run	Elastic Energy [kNm]	Stock violations	Used modules	Score
1	1.35	0	1 0 2 3 4 3	0.0128
2	1.42	0	2 0 2 5 4 0	0.0996
3	1.35	1	2 0 3 3 3 2	0.1101
4	1.31	0	1 0 2 4 4 2	-0.0340
5	1.42	0	2 0 2 2 4 3	0.0989
6	1.33	1	0 0 3 4 4 2	0.0834
7	1.42	0	3 0 2 3 3 2	0.0976
8	1.31	1	0 0 3 4 4 2	0.0635
9	1.38	0	1 0 2 5 3 2	0.0460
10	1.34	1	2 1 2 2 4 2	0.0967
average	1.36	0.4		0.0675

Table C.2: Benchmarking the Evolutionary solver with stock constraints: 2000 samples

Run	Elastic Energy [kNm]	Stock violations	Used modules	Score
1	1.36	0	3 0 1 4 4 1	0.0309
2	1.34	1	0 1 2 5 4 1	0.0954
3	1.36	1	2 0 3 4 4 0	0.1258
4	1.32	0	1 0 2 3 4 3	-0.0286
5	1.30	1	1 0 3 2 4 3	0.0528
6	1.31	1	0 1 2 4 3 3	0.0552
7	1.36	0	1 0 2 3 4 3	0.0309
8	1.32	1	0 1 1 5 4 2	0.0891
9	1.40	0	2 0 2 4 3 2	0.0755
10	1.32	0	0 0 2 5 4 2	-0.0206
average	1.34	0.5		0.0506

Table C.3: Benchmarking the Evolutionary solver with stock constraints: 2000 samples, with 100 samples optimization

Run	Elastic Energy [kNm]	Stock violations	Used modules	Score
1	1.36	0	3 0 1 4 4 1	0.0309
2	1.29	1	1 1 1 4 4 2	0.0335
3	1.37	0	2 0 2 4 4 1	0.0355
4	1.34	1	0 0 2 5 5 1	0.0967
5	1.34	1	0 1 2 5 4 1	0.0954
6	1.38	1	1 0 1 5 5 1	0.1508
7	1.36	2	1 0 1 4 6 1	0.2222
8	1.45	0	2 0 2 2 4 3	0.1397
9	1.36	1	2 0 3 4 4 0	0.1258
10	1.35	0	0 0 1 6 4 2	0.0092
average	1.36	0.7		0.0844

Table C.4: Benchmarking the Evolutionary solver with stock constraints: 5000 samples

Run	Elastic Energy [kNm]	Stock violations	Used modules	Score
1	1.36	0	3 0 2 3 4 1	0.0335
2	1.38	0	1 0 1 5 4 2	0.0642
3	1.37	0	2 0 2 3 4 2	0.0479
4	1.38	0	3 0 2 2 3 3	0.0625
5	1.33	1	1 1 2 4 3 2	0.104
6	1.33	1	3 0 3 3 4 0	0.1041
7	1.36	1	3 0 3 3 4 0	0.1379
8	1.32	1	0 1 2 4 4 2	0.0855
9	1.33	1	2 1 2 2 4 2	0.1045
10	1.35	1	0 1 2 4 4 2	0.1203
average	1.36	0.6		0.0864

Table C.5: Benchmarking the Evolutionary solver with stock constraints: 10000 samples

Run	Elastic Energy [kNm]	Stock violations	Used modules	Score
1	1.36	0	1 0 2 7 2 1	0.0383
2	1.31	0	0 0 2 6 4 1	-0.0180
3	1.35	0	2 0 2 4 4 1	0.0257
4	1.32	1	1 0 0 5 5 2	0.0893
5	1.34	0	3 0 1 3 4 2	0.0170
6	1.27	1	1 1 2 4 4 1	0.0278
7	1.34	0	3 0 2 3 3 2	0.0173
8	1.37	0	1 0 2 6 3 1	0.0504
9	1.35	1	1 1 2 3 4 2	0.1218
10	1.42	0	0 0 2 5 4 2	0.1093
average	1.34	0.3		0.0479

Table C.6: Benchmarking the Evolutionary solver with stock constraints: 30000 samples

Run	Elastic Energy [kNm]	Stock violations	Used modules	Score
1	1.34	1	1 1 4 4 4 2	0.1463
2	1.34	0	2 0 0 6 3 2	0.0510
3	1.40	0	4 0 2 3 4 0	0.1148
4	1.32	0	0 0 2 5 4 2	0.0194
5	1.31	1	2 1 2 3 4 1	0.1112
6	1.39	0	1 0 2 5 3 2	0.1009
7	1.34	1	0 0 2 5 5 1	0.1447
8	1.37	1	0 0 3 3 4 3	0.1779
9	1.29	1	1 1 2 5 4 0	0.0910
10	1.31	1	2 0 3 1 4 3	0.1087
average	1.34	0.6		0.1066

Table C.7: Benchmarking the Evolutionary solver with stock constraints: EVO 10 min

Run	Elastic Energy [kNm]	Stock violations	Used modules	Score
1	1.42	2	1 1 3 4 4 0	0.3056
2	1.35	2	4 1 3 2 3 0	0.2140
3	1.39	2	1 2 2 7 1 0	0.2666
4	1.42	2	4 1 3 4 1 0	0.3032
5	1.38	0	3 0 2 4 4 0	0.0567
6	1.30	4	0 2 4 5 2 0	0.3426
7	1.37	2	1 1 3 5 3 0	0.2410
8	1.30	4	0 4 2 4 3 0	0.3458
9	1.30	4	0 4 2 4 3 0	0.3458
10	1.31	1	1 1 2 5 4 0	0.0673
average	1.35	2.3		0.2489

Table C.8: Benchmarking the Evolutionary solver with stock constraints: EVO 3 min

Run	Elastic Energy [kNm]	Stock violations	Used modules	Score
1	1.43	2	1 2 2 4 4 0	0.3146
2	1.40	1	1 0 3 6 3 0	0.1742
3	1.65	2	2 2 2 3 4 0	0.6013
4	1.38	3	0 2 3 6 2 0	0.3529
5	1.48	2	4 2 2 2 3 0	0.3782
6	1.41	3	0 3 2 6 2 0	0.3917
7	1.36	1	0 1 2 6 4 0	0.1269
8	1.42	3	1 2 3 4 3 0	0.4087
9	1.41	3	2 0 5 5 1 0	0.3946
10	1.47	2	2 1 3 4 3 0	0.3661
average	1.44	2.2		0.3509

D

Structural analysis and verification

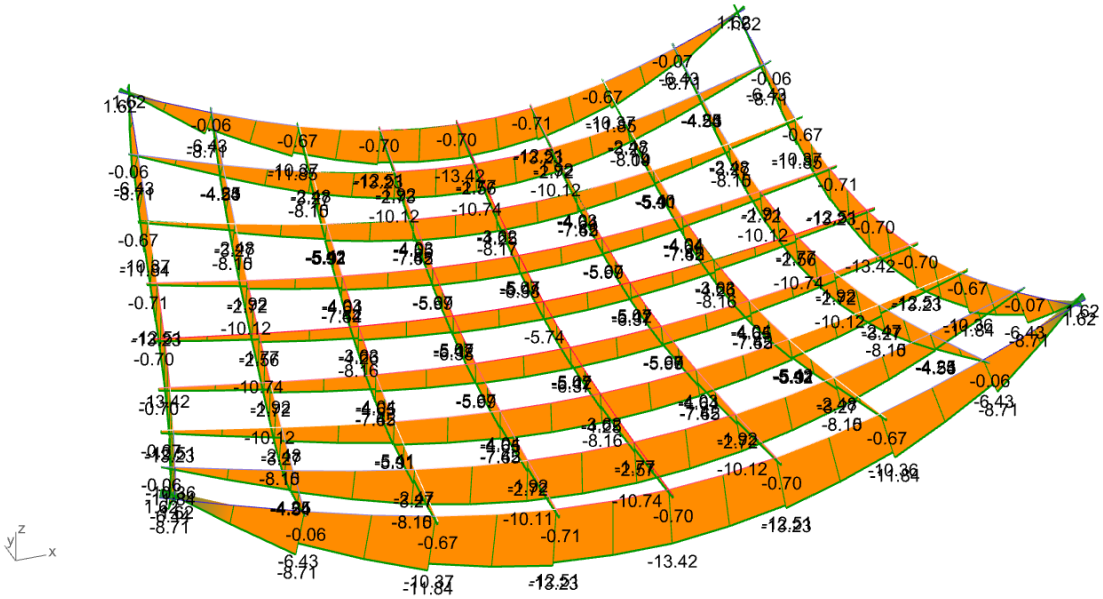


Figure D.1: M_y for ULS with 5 kN/m^2 distributed load

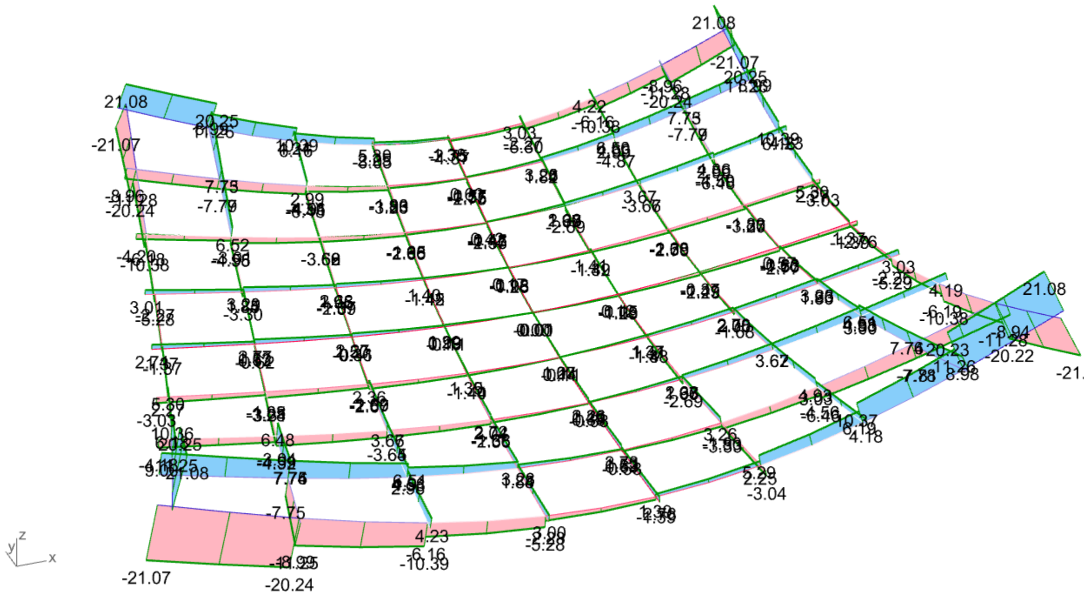


Figure D.2: V_z for ULS with $5\ kN/m^2$ distributed load

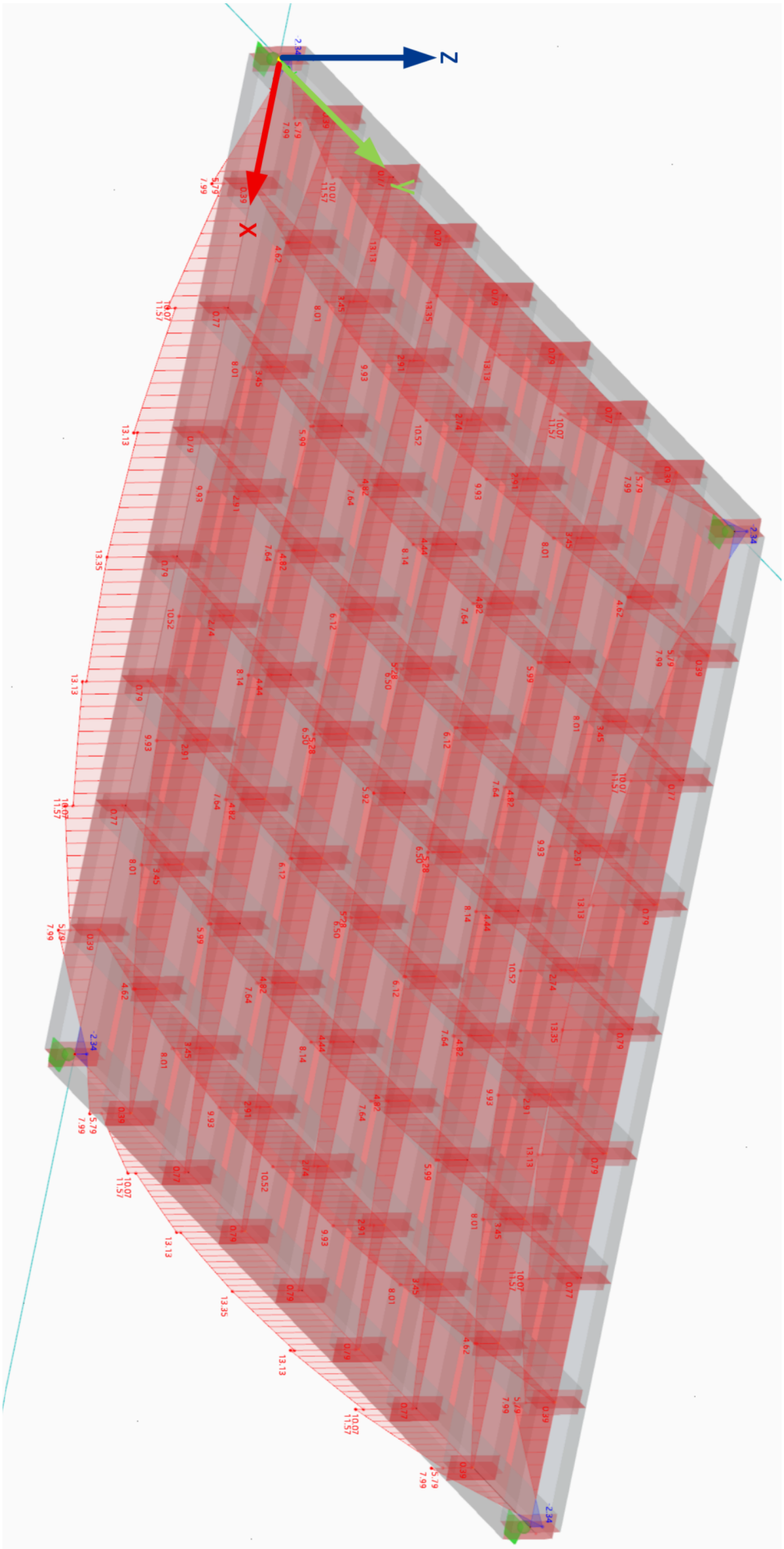


Figure D.3: M_y for ULS with 5 kN/m^2 distributed load

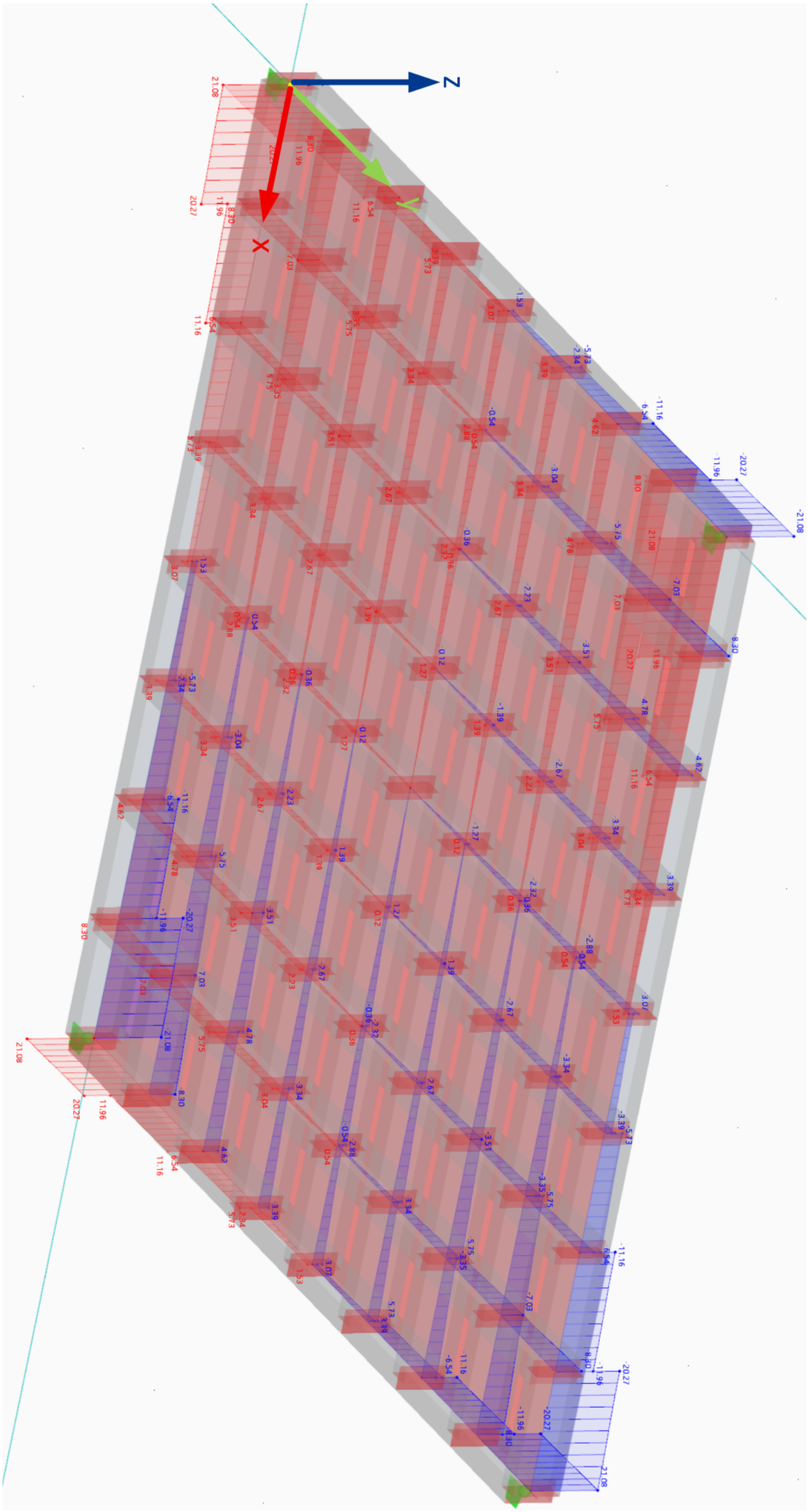


Figure D.4: V_z for ULS with 5 kN/m^2 distributed load

Bibliography

[1] A. Al-Najjar and T. Malmqvist. Embodied carbon saving of reusing concrete elements in new buildings: A swedish pilot study. *Resources, Conservation and Recycling*, 212:107930, 2025.

[2] M. A. Amir, O. Mesh□based topology, shape and sizing optimization of ribbed plates. *Struct Multidisc Optim*, 67(103), 2024.

[3] S. O. Amir, O. Reinforcement layout design for concrete structures based on continuum damage and truss topology optimization. *Struct Multidisc Optim*, 47:157–174, 2013.

[4] T. Barbier. *Performance-based and Eurocode-based design optimization of reinforced concrete ribbed floors*. PhD thesis, KU Leuven, 2023.

[5] Benytech. Waffle slab installation contractor.

[6] J. Burger, T. Huber, E. Lloret-Fritschi, J. Mata-Falcón, F. Gramazio, and M. Kohler. Design and fabrication of optimised ribbed concrete floor slabs using large scale 3d printed formwork. *Automation in Construction*, 144:104599, 2022.

[7] H. K. Dai, Y. An, W. Huang, and C. Chen. Design optimization of floor plan for public housing buildings in hong kong with consideration of natural ventilation, noise, and daylighting. *Building and Environment*, 263:111865, 2024.

[8] B. M. . B. H. Eberhardt, L. C. M. Building design and construction strategies for a circular economy. *architectural engineering and design management*, 18(2):93–113, 2022.

[9] C. Ecology. Inventory of carbon energy (ice) database. version 4.0. [data set].

[10] S. Farahmand-Tabar and N. Sadrekarimi. *Overcoming Constraints: The Critical Role of Penalty Functions as Constraint-Handling Methods in Structural Optimization*, pages 1–26. Springer Nature Singapore, Singapore, 2023.

[11] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, 2018.

[12] A. Halpern, D. Billington, and S. Adriaenssens. The ribbed floor slab systems of pier luigi nervi. *IASS Symposium*, 54:127–136, 09 2013.

[13] A. L. Huxtable. *Pier Luigi Nervi*. Braziller, 1960.

[14] J. L. Jewett and J. V. Carstensen. Topology-optimized design, construction and experimental evaluation of concrete beams. *Automation in Construction*, 102:59–67, 2019.

[15] D. P. Kingma, M. Welling, et al. Auto-encoding variational bayes, 2013.

[16] X. Lin, P. Su, W. Lu, and H. Guo. Modulepacking: A top-down generative design approach for modular key plans. *Journal of Computing in Civil Engineering*, 39(1):04024047, 2025.

[17] G. S. N. D. R. S. V. S. Luong, P. Bayesian optimization with discrete variables. *AI 2019 : Advances in Artificial Intelligence : Proceedings of the 32nd Australian Joint Conference*, 2019.

[18] P. N. M Danschutter de and B. Oostdam. Tijdelijke rechtbank amsterdam (2): tenderfase- warmlopen voor circulariteit. *Bouwen met staal*, 257:12–21, 2017.

[19] J. Ma, M. Gomaa, D. W. Bao, A. Rezaee Javan, and Y. Xie. Printnervi – design and construction of a ribbed floor system in the digital era. *Journal of the International Association for Shell and Spatial Structures*, 63, 09 2022.

[20] G. Mirra and A. Pugnale. Comparison between human-defined and ai-generated design spaces for the optimisation of shell structures. *Structures*, 34:2950–2961, 2021.

[21] G. Mirra and A. Pugnale. Expertise, playfulness and analogical reasoning: three strategies to train artificial intelligence for design applications. *Architecture, Structures and Construction*, 2:111–127, 2022.

[22] S. Oh, Y. Jung, S. Kim, I. Lee, and N. Kang. Deep generative design: Integration of topology optimization and generative models. *Journal of Mechanical Design*, 141:1, 07 2019.

[23] OpenAISpinningUp. https://spinningup.openai.com/en/latest/spinningup/rl$_i ntro.html$.

[24] R. Oval. Nervi puzzle: a topologically reconfigurable modular ribbed floor. *Proceedings of the IASS Symposium 2024 Redefining the Art of Structural Design*, 2024.

[25] L. Regenwetter, A. H. Nobari, and F. Ahmed. Deep generative models in engineering design: A review. *Journal of Mechanical Design*, 144(7), 2022.

[26] Rijksoverheid, 2023. https://www.nederlandcirculairin2050.nl/samenwerking/transitieagendas/transitieagenda-bouw.

[27] D. Rutten. Galapagos: On the logic and limitations of generic solvers. *Architectural Design*, 83(2):132–135, 2013.

[28] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[29] A. Sterrenberg. Deep generative design: A deep learning framework for optimized spatial truss structures with stock constraints. Master's thesis, TU Delft, 2023.

[30] J. Storm. High-dimensional numerical optimization of fiber reinforced polymers with variational autoencoders and bayesian optimization. Master's thesis, TU Delft, 2021.

[31] D. teaching team. Dsaie book, 2024.

[32] A. van den Oord, O. Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. *CoRR*, abs/1711.00937, 2017.

[33] M. Vejrum, P. Jensen. Design and construction of a ribbed concrete slab based on isostatic lines. *ICSA Symposium*, 2022.

[34] H. Wang, L. Zhao, H. Zhang, Y. Qian, Y. Xiang, Z. Luo, and Z. Wang. Carbon emission analysis of precast concrete building construction: A study on component transportation phase using artificial neural network. *Energy and Buildings*, 301:113708, 2023.

[35] J. Whiteley, A. Liew, L. He, and M. Gilbert. Engineering design of optimized reinforced concrete floor grillages. *Structures*, 51:1292–1304, 2023.

[36] W. Zhang, Z. Yang, H. Jiang, S. Nigam, S. Yamakawa, T. Furuhata, K. Shimada, and L. B. Kara. 3d shape synthesis for conceptual design and optimization using variational autoencoders, 2019.

[37] J. ZHU, H. ZHOU, C. WANG, L. ZHOU, S. YUAN, and W. ZHANG. A review of topology optimization for additive manufacturing: Status and challenges. *Chinese Journal of Aeronautics*, 34(1):91–110, 2021.