Multi-Material 3D Printing

Haoyu Wang

Department of Engineering Science

University of Auckland

Supervised by:

Dr. Emilio Calius
Dr. John Cater

September 2016

Abstract

The aim of this project is to identify and overcome the some design difficulties that may arise from 4D printing. Using additive manufacturing (AM) to build complex shapes with different materials and embed prefabricated components with in a single heterogeneous body will allow the fabrication of highly integrated and multi-functional objects that require little or no assembly and minimal tooling. This problem is approached by voxelising specified shapes and performing Genetic Algorithm to find the optimal material assignment. The result of the optimisation can then be printed and using STL files generated using a CAD package. The method is validated numerically using FEM analysis to converge on model geometries to a set of natural frequencies with a error below 5%.

Acknowledgements

Firstly I would like to thank Callaghan Innovation for providing the opportunity for me to work on a great project. I would also like to thank my supervisor Dr. Emilio Calius for the help and assistance he gave me through out the project. I would also like to acknowledge my co-supervisor Dr. John Cater for the contribution and suggestions he gave me while keeping me on track.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Additive manufacturing is a rapidly growing area of research, which has the ability to build complex shapes and internal structures as easily as simple ones. By combining this with the currently developed technology of multi-material printing we are looking at a brand new field of composite design space that require little or no assembly for highly integrated products. This means that 3D printing is no longer limited to prototyping duties but can also be used in production.

1.1 Motivation

The most attractive part of using multi-material 3D printing for production is the theory of creating properties as needed. If we are able to discover a structure using a complex and heterogeneous arrangement of the materials then the theory can be achieved. However, to do this, we need to consider not only the geometry of the object but also the interior material distribution. This report outline the methodology of creating and determining the material arrangements for a property of need by using natural frequency as a case study. By doing so, we show that the process can be applied to any other material properties. As most mechanical behaviour of an object can be approximated using FEM, hence we can manipulate other properties using the as method.

1.2 Project Scope

Having a toolchain would open up a new design space that allows users to choose the properties of an object without modifying the topology. This is also able to overcome some of the major limitations of conventional design theory, yet providing a faster production. This section addresses the currently available technologies that are involved and some of the existing the theoretical approaches for such tasks.

1.3 Additive Manufacturing

Ever since the 1980s, when the first form fabricating objects using a layer by layer technique was introduced, it has been commonly used for rapid prototyping purposes, to showcase objects. [3].

The fabrication process of most AM technologies creates successive cross-sectional layers of an object to directly form a 3D model. These models are often created using a CAD

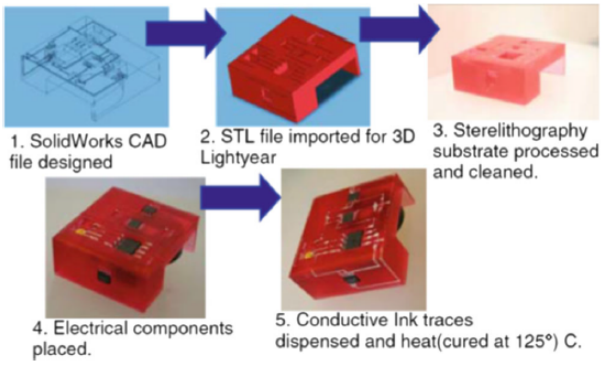


Figure 1.1: 3D printing process of a magnetic flux sensor [1].

(computer-aided design) software which produces an STL file that defines the boundary of the object.

As this technology became more and more advanced for the past decade, the price AM been dropping while allowing the user to create a better and complex object. During this development, there have been many studies on optimising the processes of AM [5], but few concerning about how the recent development have changed the design space of AM. Causing the AM technology not been used to its full potential.

1.4 Multi-Material Printing

In order to produce a well-integrated 3D object can used in application, using one material is often not enough. Hence multi-material 3D printing has been developed to allow fabrication of 3D composites that are composed of two or more different materials. These materials are fused together to provide a large range of material properties. This means the design space is no longer limited to one or two material properties but to all combinations of the available materials. The ability to combine materials to achieve a certain property or functionality can be utilised in many fields. e.g. combing stiff materials and soft materials can achieve different property gradients that cannot be otherwise generated within a single item.

If we can capitalise on these advantages of AM by adopting it in the manufacturing sector to rapidly produce end of use products [4]. It will not only reduce the processing time significantly but more importantly bringing a new approach to design space that is in contrast to the conventional design theory.

1.5 AM Data

The STL (StereoLithography) file has been the standard format in the AM industry since the mid-1980s [6]. STL uses triangle surface norms and vertices to approximate a CAD model.

Which means it can only describe the surface geometry of a 3D object without any information on the model's colour, material and texture, which was sufficient at the time when it was invented. STL files can be broken down into two types: Binary and ASCII. Where the binary files are more common due to their smaller file size [23].

As AM technologies have developed over the years, the ability to fabricate higher fidelity parts, multiple materials, and complex hierarchical structures has made the standard STL files insufficient due to the lack of information it contains [10].

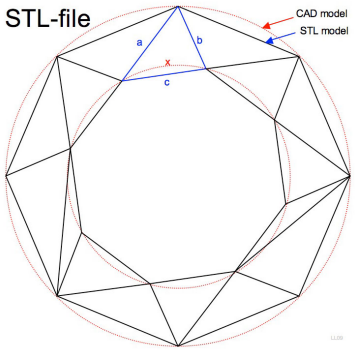


Figure 1.2: Diagram showing how STL file formats are used to approximate a 3D model [7].

1.6 Voxel-Based Design

Due to these limitations, many studies has been done on creating a new data format for 3D printing [25]. The goal of these studies is to allow users to specify not only the shape but also information such as material or the interior structure of the object. One of the common approaches is to use a volumetric representation of the object using voxels, where each voxel is an individual cube that contains information on its own colour and material. An example of this would be FAV that is designed by FUJI XEROX [25].

Another approach that is commonly employed is to use triangular volumes instead of voxels. Each of the triangular volumes are meshed with several vertices, these vertices are shared along different volumes. Some examples of these are 3MF and AMF. Sadly most of the formats mentioned have only been introduced recently which cannot be recognised in older 3D printers.

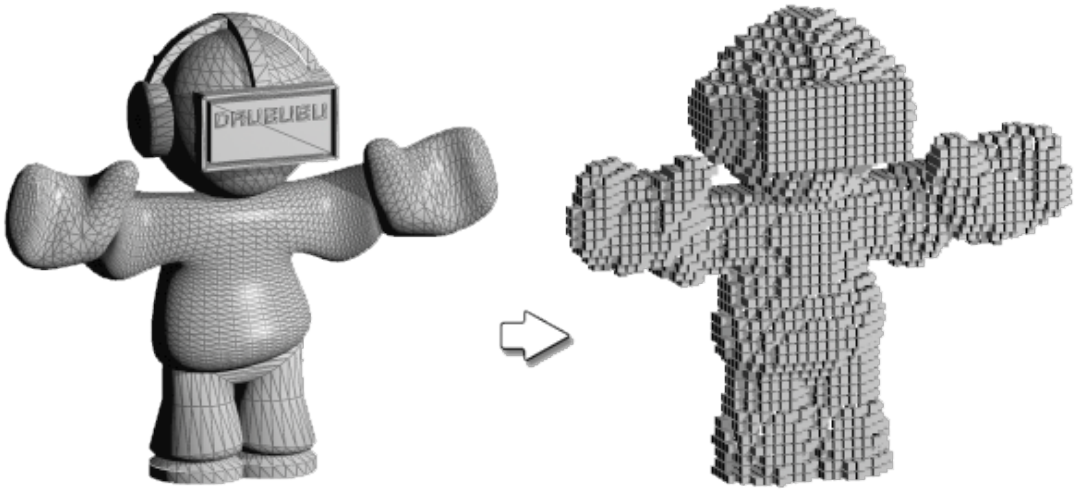


Figure 1.3: A surface model VS a voxel model [8].

Voxelisation has been used for the past three decades, mainly in gaming and medical industries [24]. However, the concept of using it inside CAD packages is rather new. Hence, it is uncommon for CAD and FEM packages to implement the ability to voxelise models. Which means the construction of voxelised models needs to be manually done using a third party package.

One of the idea is to use a combination of voxels and different materials we can modify an object's property without changing the topology [22].

As voxels are the base units of the model, the resolution/size of these voxels is one of the two main factors that determines the accuracy of the results, where the other factor is optimisation method which will be discussed later. In terms of voxel size, by having higher resolution will not necessarily result in a better solution, but having excessive number voxels can cause the optimisation and FEM calculation taking a significantly longer time. The best resolution is often determined by trial and error, running the optimisation several times and comparing the time spent and the convergence rate.

The concept of voxelisation can be done and automated using Grasshopper [32], which is a graphical algorithm editor for Rhinoceros, allowing users to create and automate CAD models by using predefined toolboxes and user defined inputs. There are several packages that are used inside Grasshopper to help populate the voxels. Using a combination of GhPython [31], which is a python compiler in grasshopper and Milkbox plugin [33], we can generate voxels using a user given geometry.

1.7 Optimisation Using Meta-Heuristics

Meta-Heuristic is a technique that is designed to select or find a sufficiently good solution for an optimisation problem. It is normally applied to problems that have a lack or no information on the behaviour and involves a solution set that is too computationally expensive to completely enumerate [11].

Below is a brief description of each of the algorithm.

- Single Objective
  - Genetic Algorithm
    A method for solving optimisation problems based on principles of natural selection [12]. The process tries to replicate biological evolution, where each iteration represents a generation of population. For every iteration, the current generation is mutated and cross-overed based on the ”fitness” of the candidate to reproduce the next generation. After many iterations, the population is hopped to converge to a good solution.
  - Particle Swarm
    unlike Genetic Algorithm, Particle Swarm Optimisation is not a selection algorithm. In PSO the population is made up of many particles, where each particle will move one step at a time through the solution space of the problem for each iteration [13]. The distance and the direction that the particles move is based on its own velocity and the best position of particles.
- Multi-Objective
  - Genetic Algorithm
    This algorithm follows the same concept of Genetic Algorithm. Instead of only having objective value, it can optimise over many. Instead of returning a single optimal solution like the single objective algorithms, it will give a number of alternative solutions lying on the Pareto optimal front [14].

1.8 Finite Element Method

Often to calculate some behaviour of a structure, a system of equations involving the equation of motion needs to be solved. Normally this can be done manually for a simple structure given the necessary parameters and a numerical model. However, things becomes complicated when solving for heterogeneous structures, where the analytical solutions of the system are usually impossible to obtain. To get an approximation of the behaviour of such structures, a numerical time-stepping methods for integration of differential equations needs to be used. A possible technique is the Finite Element Method [16].

The Finite Element Method(FEM) is a numerical technique used to subdivide mathematical models of an object into simpler elements. The response of these elements is then calculated based on the number of degrees of freedom (DOF). The response of the model is approximated using the discrete response of all elements [15].

There are many advantages of using FEM [16]:

- It can be used to predict behaviour of proposed structures, used test and optimise the system.
- It is able to calculate behaviours of structures with dissimilar material properties.
- The characteristics of the model can be changed easily, allowing many numerical experiments to be conducted until a desirable solution is found.

The above properties of FEM make it a convenient tool to use when optimising the structural behaviour of a heterogeneous object.

MATLAB [34] is a popular tool that is due to the freedom of changing the formulation easily. However, it can be difficult to solve large problems due to the slow processing speed. To solve problems that involves more complexity, more specialised programs such as: ANSYS [35], ABAQUS [36] or COMSOL [37] should be used.

1.9 Natural Frequency Control Case Study

Following the idea of Nicolas Cheney et al. [26], we chose natural frequency control as a case study to validate and our method and toolchain.

Figure 1.4: Classical failure caused by natural frequency: Tacoma Narrow Bridge [9].

Background

Being able to freely control the natural frequencies of a material has always been an area of interest in engineering [17]. Every object will exhibit some motion when excited with a periodic load. However at certain frequencies, the object will vibrate at a greater intensity when not subjected to a continuous of repeated external force. This is known as the natural frequencies of the object.

Existing Solutions

Managing structural performance is an crucial but difficult concept to the modern engineering field. The conventional engineering treatments have remained mostly untouched since the early 20^{th} century [17]. In most cases, these harmful vibrations are "directed" away from the fragile part of the structure[19]. This can be achieved by adding external devices called tuned vibration absorbers (TVA) [18]. However, by adding a external device means we are essentially increasing the mass and the complexity of the system structure.

This process has known to be inefficient [18] and fails to take advantage of the large design space made available by 3D printing. Not only the amount of stress can be absorbed by this device is limited but also the parameters and locations of the device are set using intuition and guess-and-check methods. Hence, to handle relatively large vibrations a larger TVA is required, as the frequency gets higher eventually the mass of the devices will become large enough to affect the natural frequency profile of the original material, therefore complicating the problem [18].

Some studies have also involved changing the topology of the structure to achieve a certain set of natural frequencies. Most of the proposed methods are based on a predetermined structure which creates a limited usage [20]. These strategies also involve working with optimisation over a single natural frequency [21].

Finite Element Analysis

Most physical properties are governed by the material composition of the object. In the case of a heterogeneous cantilever beam, it is necessary to determine the matrix representing the object and solve using FEM. In the case of working out natural frequencies of an object, it is necessary to determine the eigenvectors and the eigenvalues of the object. For every degree of freedom there exists a natural frequency, however in most of the applications, the number of nodes needed to perform a precise calculation is far more than the natural frequencies of interest.

Multi-material AM for Natural Frequency Control

Our idea of the new design space involves combining the additive manufacturing technology and multi-material printing technology to create a simple and automated method to construct a composite material which can be easily integrated. There have been several attempts at exploring this design space which gave us new ideas and design theories for this particular field of 3D printing [1].

Taking into account the issues we have described above we would like to create a method that can be used to control multiple natural frequencies. The approach that was mentioned in Cheney's paper is to use a black box optimisation algorithm to find the material assignments to different regions of a given shape. We have combined the ideas of using an optimisation process

and voxelised representation of an object. Placing voxels with different materials at different places on the geometry, then evaluating the difference between the desired frequency and the current assignment frequency and optimising the difference between the two.

1.10 Report Structure

The main sections of the report has been divided into three parts. Chapter 2 describes the general method that was used to optimise any properties. Chapters 3 provide a detailed description of how our optimisation process. Including how the algorithm is chosen and the performance of the chosen algorithm using natural frequency as an example.

Chapter 2

Overall Methodology

2.1 Procedure

The method involves analysing an user input object geometry, and returning multiple STL files containing the inclusions and matrices for the same geometry, each with a specific properties which then can be 3D printed together. The reason for using having to use multiple STL files to represent the object is due to the fact that STL files do not contain any information of the material. Hence a new STL file is needed to be generated for each new material added. To perform such task the following four steps were used.

1. **Voxelisation**
   The user will input a desired geometry and a voxel size. For example, this would be a pre-existing shape that is used in an engineering solution. For testing purposes, this is a simple cantilever beam that has the dimensions of $1mm \times 10mm \times 40mm$. The object geometry will be voxelised.

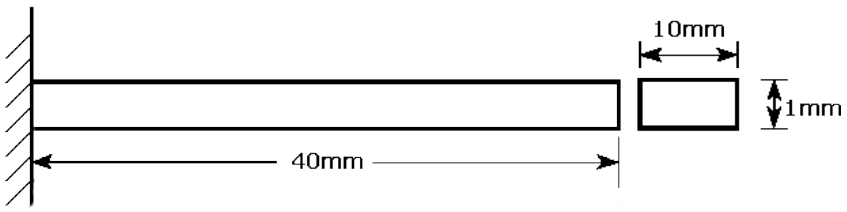


Figure 2.1: Illustration of simple cantilever beam that is used for testing.

2. **Property Calculation**
   Next, the user will input the properties of available materials and the property distribution they would like to achieve. This could be any property that can be calculated using a FEM solver. For example the stiffness, or the natural frequencies of an object. The materials available depend on the type of the AM technology and the types of materials that is compatible with the machine. To validate the method we will try to optimise the beam described above with a set of natural frequencies. The material we used is listed in later in Table 2.1.

3. **Optimisation**
   The report focuses mainly on this step, as it controls the overall performance of the

toolchain. The optimisation algorithm tries to minimise the difference between calculated material properties of our optimised beams and the target property distributions, by changing the material assignments for each voxel in the a beam.

In order to find the best optimisation method and parameters, this section will be divided into 2 phases. In the first phase, a sensitive analysis is performed on three types of meta-heuristic algorithms to find the optimal parameter set, then performing a simple test to find the best performing algorithm. Phases 2 consists of a more complicate test for the algorithm that we have selected in phase to examine the potential of the algorithm. The details of the phases are described in later chapters.

4. **Visualisation**
   Using the results from the optimisation, a physical model is constructed in a CAD package for visualisation purposes for export for 3D printing.

There have been studies on solving each of the steps separately. However, the combination of these setps to solve a complex problem has not been mentioned before. The previous solutions of each of the steps need to be modified to fit the voxel design. The main challenge is to work out the voxelised solutions for each of these sections based on previous studies and combine these solutions to make a system of solutions that works together to solve the problem. Figure 2.2 shows the workflow of our methodology graphically, showing how an user defined object is process to produce STL files of a model with the same geometry with the desired property distribution.

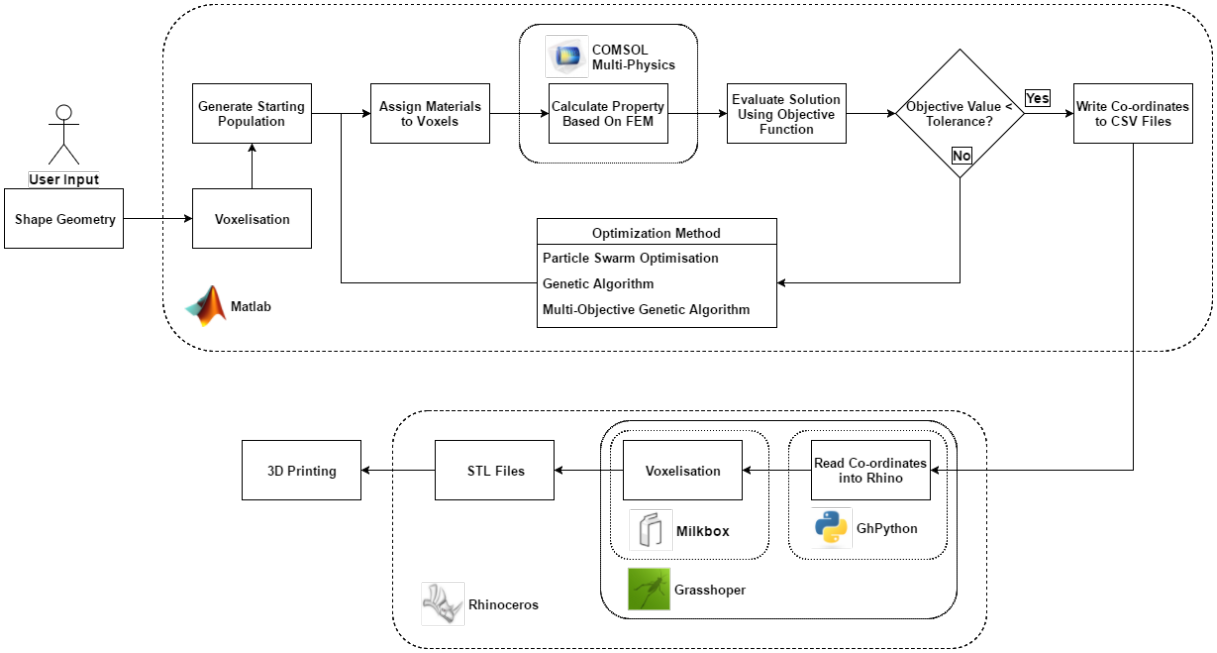


Figure 2.2: Flowchart of the overall procedure showing how a input geometry gets processed to produce output STL files containing the model for each material.

The working environment/packages of each step are also shown in Figure 2.2, an enlarged version can be found in Appendix A at the end of the report. These are selected based on the capability with the environment of the other steps. As all of these sections are interlinked with each other we need to find a set of tools that can be automated easily. Considering the above constraints, COMSOL MultiPhysics v4.3 is selected as the FEM calculator and Matlab 2015b will be used as the optimisation tool. With the COMSOL Matlab Livelink feature, FEM

calculation can be easily setup in Matlab without involving any third party software. Rhinoceros 5 [27] is used to generate the STL files and produce graphical models of the geometries.

2.2 Voxelisation

For our method, voxelisation is required for both visualisation and the FEM calculation. This is essentially the same operation, however two separate scripts need to be written due to the difference in software for each step.

Both of the scripts are written in MATLAB. The script used for COMSOL FEM calculation takes in dimensions of an object and the voxel size as an input. Then returns a COMSOL model file with each voxel constructed as an individual object. Running this script can take up to 2 - 5 minutes using a 16G RAM computer, the speed also depends on the number of voxels and the resolution of the mesh. However, this model file is reusable for calculations of different material assignments. Figure 2.2 shows a voxelised model that is generated using the MATLAB script (Refer to Appendix B).

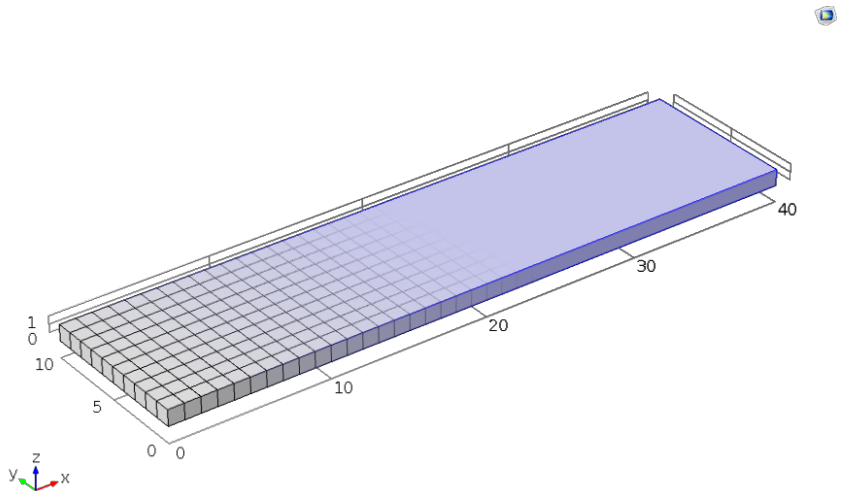


Figure 2.3: Voxelised cantilever beam geometry.

The script for visualisation takes in an array of integers representing the material assigned at each voxel and size of the voxels then returns the corresponding co-ordinates of the voxels for each of the materials as many CSV files. As mentioned before, it is due to the limitations of current 3D model format that numerious CSV files needs to be generated in order to represent all materials. Then the CSV files are picked by by Rhinoceros and STL files can be generated from Rhinoceros.

For testing we have set the voxel size to be 1mm cubes, what this means for the testing sample of the $1mm \times 10mm \times 40mm$ beam is split into 400 voxels where each voxel could potentially have different materials.

2.3 Finite Element Analysis

The materials that we are using for testing are called Vero White and Tango Plus, their properties are listed in Table 2.1:

Table 2.1: Material Properties

	Density (km/m^3)	Yong's modulus (Pa)	Poisson's ratio
Tango+	1120	9.2×10^5	0.48
Vero White	1170	2.3×10^9	0.35

The main problem with using a FEM solver is the computation time. As a meta-heuristics optimisation method is used to solve the problem, it could involve many calculations of natural frequency so it is essential that we minimise the time spend on each calculation. To improve the speed of each calculation, three adjustments were made:

1. The mesh type is changed from Free Tetrahedral to Free Quad, as our voxels are perfect cubes by approximating it using quadrilateral shapes will not only require less number of elements but also increase the accuracy of the calculations.

2. The minimum mesh element size is increased to $1mm$, which results in each of the voxels are approximated using 6 quadrilateral surfaces, 8 nodes and 16 edges.

3. The displacement field of discretisation is changed from quadratic to linear, which involves fewer calculations.

Using test cantilever beam mentioned before, it is constructed from 400 voxels, using the default setting of COMSOL the mesh had 4800 elements which can take up to 20 seconds to solve. After the adjustments are applied there are only 400 elements which take 4-6 seconds to solve. The mesh model before and after the adjustments is shown in Figure 2.4.

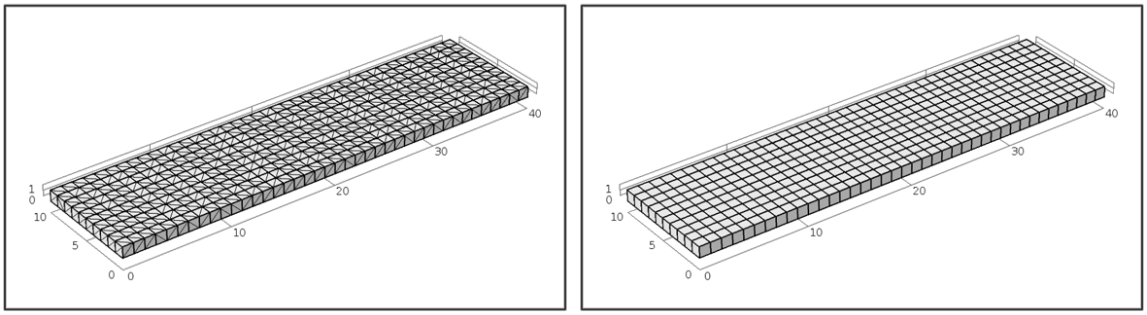


Figure 2.4: Mesh model before and after the mesh changes.

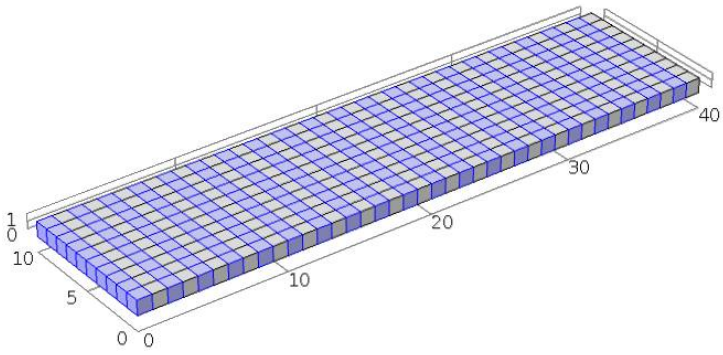


Figure 2.5: A COMSOL model with period structure

However, by making these adjustments it is sacrificing accuracy of the calculation for speed. Using an example with a periodic assignment of the materials on the cantilever beam (shown in Figure 2.5), the difference in the calculations results are recorded and presented in Table 2.2.

Table 2.2: Frequencies Calculated Before and After Mesh Changes

	Natural Frequencies (Hz)					
	1^{th}	2^{th}	3^{th}	4^{th}	5^{th}	6^{th}
Before	5.95	36.98	52.36	102.66	133.28	198.38
After	11.85	73.04	90.87	138.34	199.26	323.03

We can see in Table 2.2 that by changing the discretisation method and mesh model, the frequencies calculated shifts slightly higher than before. Which means this is effectively decreasing the accuracy however for testing purposes as long as the setup remains constant for all samples it is enough to show the validity of the methodology.

2.4 Optimisation

Our problem involves solving for some physical property of a heterogeneous beam, which involves solving multiple nonlinear equations. Due to this, it would be rather difficult to capture the behaviour of the problem while optimising. As our objective function has large objective space and many local minimum, it would be extremely inefficient to implement local search methods. Also, we are not required to find the global minimum of the problem, as we are only looking for a ”good enough” solution.

With these limitations in mind, it is clear that a black box meta-heuristic optimisation algorithm such as GA or PSO would be suitable for this problem. These algorithms are designed to provide a sufficiently good solution, while not requiring any information on the objective function. However, the disadvantages to this type of algorithm is that it can take a long time to converge and is heavily dependent on the starting population.

There are many different meta-heuristic algorithm that has been developed, however, the performance of each of these heavily depend on the problem. We chose three of the more well-known algorithms to compare the performance for each and mainly as a prove of concept.

On the next page we have listed the types of algorithms we will be evaluating and the corresponding packages used to perform these algorithms.

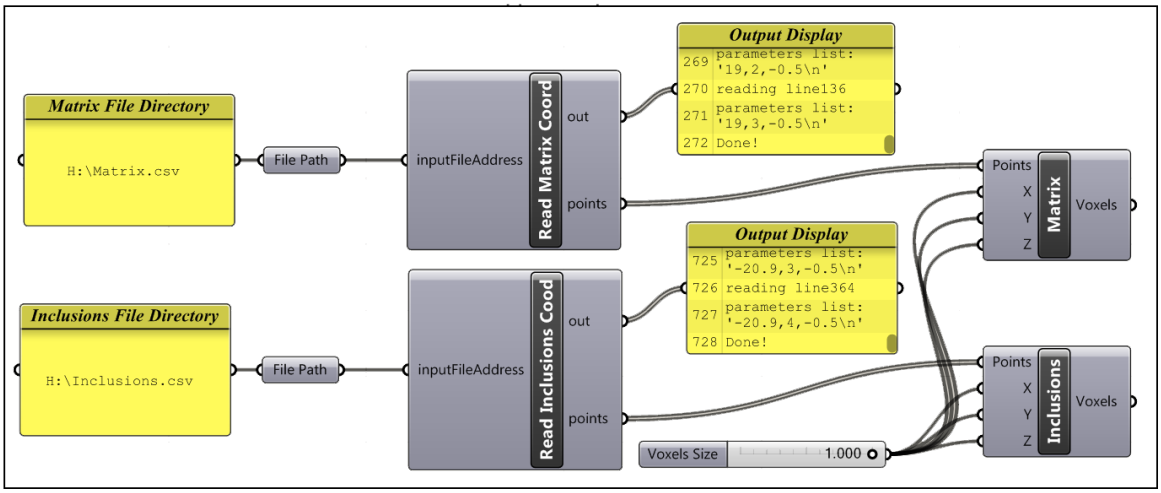
- Single Objective
  - **Genetic Algorithm**
    Adapted from Yarpiz’s code [28].
  - **Particle Swarm**
    Using MATLAB toolbox ”PSOPT” [29].
- Multi-Objective
  - **Genetic Algorithm**
    Using the function ”gamultiobj” from MATLAB optimisation toolbox [30].

The source code used to run each algorithm can be found in appendix B.

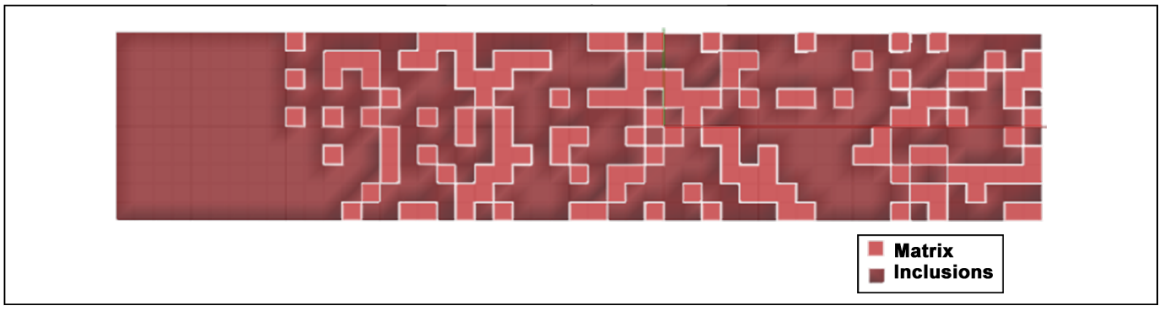
2.5 Visualisation

Due to the lack of voxelisation ability of most CAD software, it is rather difficult to find an option where the voxelisation is done automatically. Using the CSV file that was constructed earlier in the voxelisation step, it is possible to make a voxelisation algorithm in Rhinoceros that generates specific voxel at the given coordinate.

This is done inside Grasshopper, where firstly GhPython[31] is used to read CSV files and converts the coordinates to a point array which can be read by Grasshopper. The points are then passed to a voxel tool contained in the Milkbox plugin [33], this tool creates a cube with a given size at each point which together forms our shape.



A. Grasshopper Script for Voxelisation



B. Visual Output to Rhino

Figure 2.6: Grasshopper script with example Rhinoceros output

This means that the only thing that needs to be completed to generate a voxelised CAD model of the given geometry is run the MATLAB script to construct the CSV files. A snapshot of the grasshopper file and an example of the output is presented in Figure 2.5.

Chapter 3

Optimisation Methodology

In our methodology, optimisation is the core process that controls the accuracy and speed of the procedure. To find the best algorithm and the evaluate the performance of the algorithm, two phases of tests were carried out, one before another.

Phase 1

Phase 1 testing is used to distinguish the best algorithm out of Genetic Algorithm, Particle Swarm Optimisation and Multi-Objective Optimisation. Considering the time consumption of running each of the algorithms, it is inefficient to run the final testing on all three. Therefore a faster and more efficient test is designed to discover the best performing optimisation algorithm.

The test engages on only optimising for the first two frequencies for a periodic beam with a sensitivity analysis for each algorithm. This should allow each algorithm to have their optimal performance. The final objective value and the time taken for the algorithm to converge as the factors of the final decision.

Phase 2

The purpose of this testing phase is to examine the accuracy of our toolchain for a complex problem involving natural frequencies. We try to optimise the material assignment of each voxel of the beam using the Genetic algorithm that we have selected in phase 1 with the optimal parameters. We do this by comparing the difference between the weight least square error of our optimised model to the target model. The algorithm is further tune to fit the purpose of our problem.

3.1 Objective Function Formulation

Before we can set up any optimisation problem, the objective function must be clearly defined to make sure that we are optimising the correct parameters. The objective function is formulated with the idea of minimising the difference of the target frequency and the frequency of the material assignment for that candidate. The function takes in an array of integer variables, each variable represents a voxel and the integer that is assigned to the variable represents the material that is assigned to that voxel. Using the input array we are able to modify the model that we have constructed earlier to the material assignment specified by the array.

The model is then calculated using COMSOL and the result of natural frequencies are returned to MATLAB as an array. Using target frequencies and the frequencies from COMSOL,

we can calculate a weighted sum of the differences between each of the frequencies using the fitness function mentioned previously.

3.2 Starting Populations

Phase 1

The starting population is generated based on the random assignment of materials in the voxels. This is done by generating a random number between 0 and 1 for each of the voxels in the geometry. Based on the number being greater or smaller than 0.5, the corresponding material will be assigned to that voxel. This means the probability of each voxel being material 1 or material 2 is approximately equal.

The size of each generation will be controlled at 20 individual candidates for faster iterations. The starting population is the same for all algorithms for fairness.

Phase 2

Starting population is a crucial key in determining the convergence of these algorithms, with a candidate that is closer to the desired frequency we would be more likely to converge to a better solution with a faster speed. By having a larger spread over a different ratio of materials, we are hoping to find a good candidate to start with. However, there is a trade-off between the size of the population versus the number of iterations. With a large population size it is more likely to pick up traits with better fitness, but taking a longer time to run.

The procedure that was undertaken is to start with a large population size of 40, ranging from a material ratio of zero to one. This provides enough diversity to the population giving it the potential to find a candidate that is relatively close to the target. As we get closer to our target frequency the population size is decreased to 20 for faster iterations.

3.3 Fitness Function

Phase 1

A Fitness function was designed for the single objective optimisation algorithms, the idea is to sum the squared difference of the calculated frequency (f_i^{calc}) and the target frequency (f_i^{target}).

$$\min f(x) = \Sigma_{i=1}^{2}(f_i^{\text{target}} - f_i^{\text{calc}})^2 \tag{3.1}$$

The multi-objective genetic algorithm would have 2 objectives each being the difference of the 2 target frequencies.

Phase 2

Unlike the fitness function for phase 1, the fitness function for phase 2 focuses on the first 6 natural frequencies. The fitness function is adapted from Nicolas' paper [26], which evaluates the error in matching the target frequencies and takes a weighted sum of least squares approach to problem.

$$\min f(x) = \Sigma_{i=1}^{n}(\frac{n-i+1}{n})err(i)^2 \tag{3.2}$$

Where n is the number of frequencies being optimised, and $err(i)$ is the error in matching the i^{th} target frequency. The weightings of each frequency is defined by $\frac{n-i+1}{n}$. For example if the problem involves optimising over 6 frequencies, using the weight calculation mentioned the error in matching the first frequency will be $\frac{6-1+1}{6} = 1$ whereas the error of the 4^{th} frequency would be $\frac{6-4+1}{6} = 0.5$. This means that as we go further with increasing order, it would be less penalising to have a larger error. This is so that the method focus to get the primary frequencies right, rather than trying to converge with many objectives.

3.4 Target Frequencies

Phase 1

In theory, the results of the test should not be affected by the target frequencies as the algorithms have the same starting population and given the same number amount of iterations to converge. However, using the frequencies with a known structure such as a periodic beam, can still be beneficial as it can provide information on how different the solution structure is to the known structure.

For this reason, the natural frequencies of a periodic beam has been selected. Due to the way that the initial population is established, it would have a high chance to have a good candidate in the intial population, that lies closely to our target solution and hopefully be a guidance to the algorithms.

Phase 2

In order to test performance and the ability to converge to different frequency sets, 5 sets of frequencies are "randomly" generated. These sets of frequencies must exist within the range of the material properties. By using the following formula is used to generate the target frequencies:

$$f_n = \frac{f_n^{\text{stiff}} - f_n^{\text{soft}}}{z} * (n + \mathcal{N}(0, 0.25^2)) + f_n^{\text{soft}} \tag{3.3}$$

f_n^{stiff} and f_n^{soft} are the natural frequency of a beam that is fully constructed from the stiff and soft material respectively. Where z is the total order of natural frequencies that are being evaluated, n is the frequency number that is being generated. Thus for our problem $z = 6$ as we are evaluating up to the 6^{th} natural frequency, which means $n = 1, 2, ..., 6$.

Equation 3.3 uses the range between the principle material frequencies to work out a linear spread of the frequencies and generates the target frequencies by applying a random change based on a number from a normal distribution. This ensures our target frequencies will be a good spread between the feasible range while having some randomness to it. This also allows us to examine the performance of the algorithm with different assignments.

3.5 Stopping and Restarting Condition

Phase 1

The algorithm is only stopped if the maximum number of iteration is reached and no restarts are implemented. This means each algorithm only have one chance to converge to the best solution.

Phase 2

There are two different stopping condition of the algorithms, the first one is if the algorithm has reached a tolerance, meaning that it is close enough to the target that was set by the user. The other condition is that if the best cost of the algorithm has been stalling for 10 or more iterations, the algorithm will restart with a new set of starting points. Each run the algorithm is allowed to restart n times before terminating, where the best solution of all restarts will be the output.

3.6 Cross-overs, Mutation and Selection Methods

As this only applied to genetic algorithm, the same settings has been used for both phase 1 and phase 2. Unlike conventional genetic algorithms, instead of sticking with one crossover method the algorithm used is adapted to have 3 different crossover methods available to choose from: single point, double point and uniform. Whenever 2 chromosomes are about to crossover, one of the 3 methods will be selected based on a roulette wheel selection. The probability of each of the methods being selected is 0.1, 0.2 and 0.7 respectively.

3.7 Test Specifications

Phase 1

When running a metaheuristics algorithm, the solution quality is heavily influenced by parameters such as: the rate of crossover, the rate of mutation and number of generations. Many studies have been carried out to find the best performing parameters for different algorithms. However most of these came to a conclusion that the optimal parameters depend on the question and the environment.

The idea of sensitivity analysis is to find the best performing parameter by iterating through different parameters while fixing the others to see their interaction with the solution quality. By comparing the final cost of the evaluation it is possible to select parameters.

Hence, some parameters that are known to be sensitive to the algorithms are tested. These are:

- Genetic Algorithm (GA)
  - Cross-over rate
    This factor controls the amount of candidates inside the population that will crossover and produce the next generation.
  - Mutation rate
    This factor shows the percentage of the population that will be mutated and passed on to the next generation.
- Particle Swarm Optimisation (PSO)
  - Cognitive Attraction
    Controls the acceleration of a particle towards its own best position. This represents the awareness of the particle.
  - Social Attraction
    Controls the acceleration caused by the best position of the swarm.

The parameters for each algorithm are listed in Table 3.1, where p_c is the probability of crossover, p_m is the probability of mutation, c_1 is the Cognitive Attraction, and c_2 is the Social Attraction factor.

Table 3.1: Parameters Values

Methods	Parameters	Values
GA	p_c	0.6, 0.7, 0.8, 0.9
	p_m	0.1, 0.2, 0.3, 0.4
PSO	c_1	0.5, 1.0, 1.5
	c_2	0.5, 1.0, 1.5
MOGA	p_c	0.6, 0.7, 0.8, 0.9
	p_m	$1 - p_c$

Each of the combinations of parameters is tested for each algorithms with a maximum number of iterations of 50. No restarts are allowed and at the end of the 50 iterations, the best objective value is recorded.

By starting with the same population, eliminates the chance of the difference in objective value being affected by the starting population. As a result of this, objective values will give an indication of the performance of the algorithms with the specified parameters.

After the best parameters are obtained, each of the algorithms is run one more time with 100 iteration. This should give the algorithm an opportunity to generate a better solution.The final objective value is used as an indication of the performance of each algorithm.

Phase 2

To test the potential of the selected algorithm from phase 1, 5 sets of target frequencies are generated using Equation 3.3. However, to fully examine the performance of the algorithm, other than the randomly generated frequencies it is ideal to also test the convergence of some known structures. Therefore the natural frequencies of a periodic beam are added to the list of target frequencies, by being able to converge to such a structure or different structure with similar properties would be desirable. The target frequencies are presented in Table 3.2 below.

Table 3.2: Natural Frequencies of Target Sets

n	Full Soft	Full Stiff	Set 1	Set 2	Set 3	Set 4	Set 5	Periodic
1	4.579	183.6	35.85	56.86	94.02	134.23	162.12	11.85
2	23.002	1134.6	217.12	347.58	578.24	827.89	1001.0	73.04
3	28.443	1144.7	223.38	354.38	586.01	836.70	1010.5	90.88
4	28.853	1366.3	262.41	419.38	696.90	997.27	1205.5	138.34
5	73.382	3203.7	620.04	987.41	1637.0	2340.0	2827.4	199.26
6	79.891	3557.4	687.18	1095.3	1816.9	2597.9	3139.4	323.03

3.8 Overall Procedure

In Figure 3.1, the above sections are organised into a flowchart.

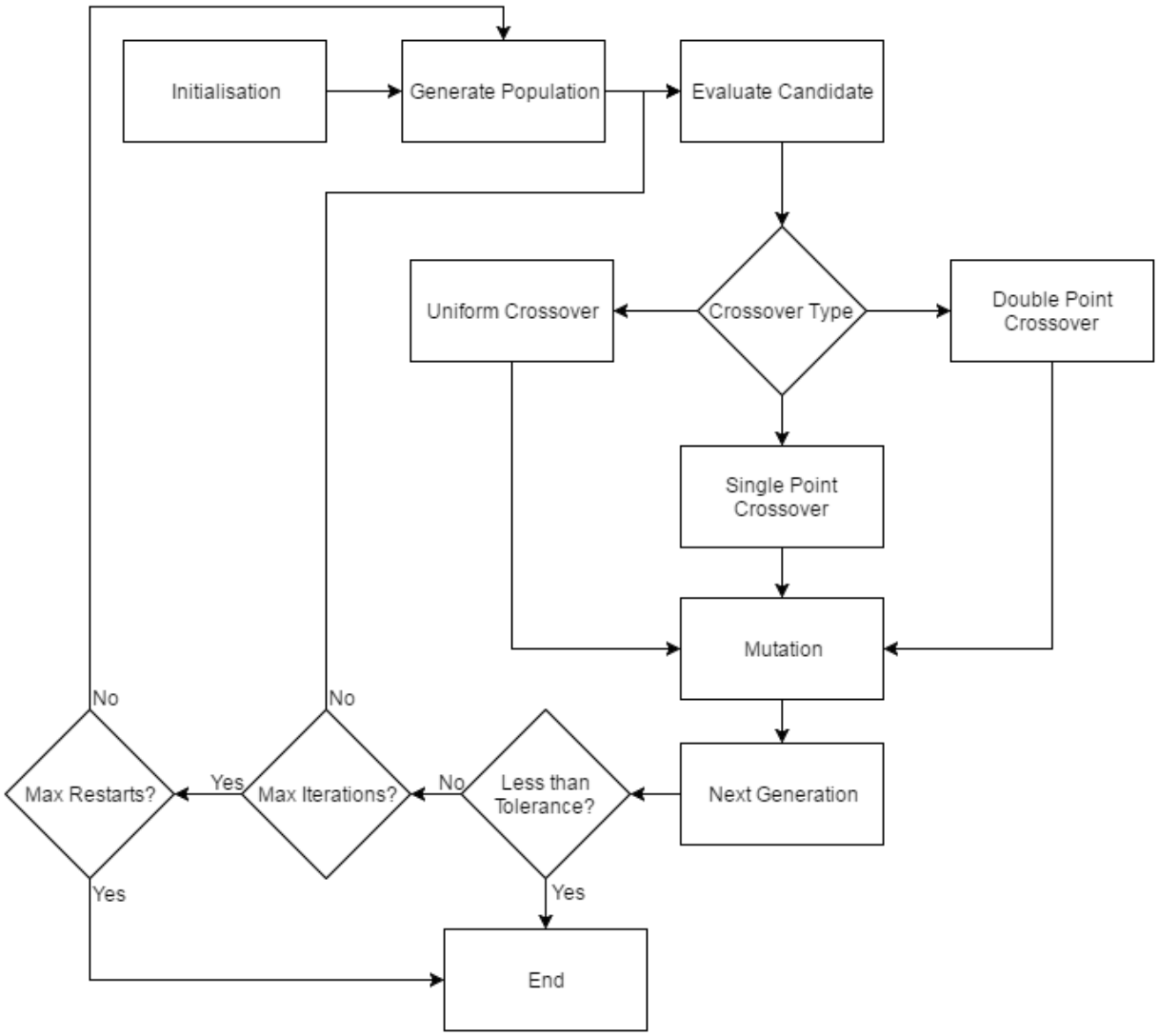


Figure 3.1: Flowchart of adapted Genetic Algorithm.

Chapter 4

Phase 1 Results

This section presents the results of the testing completed. Due to the complexity of the problem, it is hard to distinguish an exact relationship for each of the parameters to the objective value. However, the analysis is still practically important when using metaheuristic algorithms as it reveals the influence of parameters. This is especially true for large size problems, by having the correct parameters it can result in a better solution as well as a faster solution time.

The objective values are calculated using the fitness function (Equation 3.1) define in the previous section, which we are trying to minimise.

4.1 Particle Swarm

Table 4.1 shows the results of PSO after 50 iteration using combinations of Cognitive Attraction (c_1) and Social Attraction (c_2).

Table 4.1: PSO Sensitivity Testing Results

Cognitive Attraction	Social Attraction	Obj. Function Value
2.0	2.0	1893.1
	1.0	1802.5
	0.5	1382.6
1.0	2.0	1893.1
	1.0	1482.5
	0.5	1844.3
0.5	2.0	1612.8
	1.0	1601.8
	0.5	2001.0

The best objective solution appears at $c_1 = 2$ and $c_2 = 0.5$. These parameter values allows the algorithm to keep track of current best location while exploring the rest of the feature space.

4.2 Genetic Algorithm

The results in Table 4.2 represent the objective value after 50 iterations of the Genetic Algorithm using the different parameters.

It was observed that changing the parameters for GA, had a pronounced effect on the best objective value. For example, by changing the mutation rate, from 0.4 to 0.3 and leaving the cross-over rate at 0.6, the objective value experiences a decrease of more than 50%.

Table 4.2: GA Sensitivity Testing Results

Cross-over Rate	Mutation Rate	Obj. Function Value
0.9	0.4	1152.0
	0.3	1088.9
	0.2	1342.4
	0.1	1747.0
0.8	0.4	950.06
	0.3	1015.1
	0.2	1050.5
	0.1	1160.0
0.7	0.4	969.83
	0.3	715.14
	0.2	1014.6
	0.1	1448.8
0.6	0.4	706.59
	0.3	872.52
	0.2	874.27
	0.1	1363.0

The best solution is given using $p_c = 0.6$ and $p_m = 0.4$, with an objective value of 706.59. Due to the high mutation rate, it gives the population a fair amount of randomness, allowing the population to have a higher chance of escaping the local minimum. This is extremely useful for the frequency problem, due to the nonlinear nature of the problem causing the object function to contain a large amount of local minimas. Hence by have the ability to escape these minimas gives the algorithm a better convergence rate than the others.

4.3 Multi-Objective Genetic Algorithm

Table 4.3 shows the results of changing the crossover rate for the multi-objective genetic algorithm. The mutation rate is set to be $1 - p_c$.

Table 4.3: MOGA Sensitivity Testing Results

Cross-over Rate	Objective		Obj. Function Value
	$x_1 = f_1^{\text{calc}} - f_1^{\text{target}}$	$x_2 = f_2^{\text{calc}} - f_2^{\text{target}}$	
0.9	51.39	4.126	2658.2
0.8	54.65	23.97	3562.4
0.7	43.38	2.263	1887.4
0.6	40.48	35.36	2889.3

4.4 Objective Values and Time Consumption

Starting Population

The starting population is generated randomly and kept constant for all algorithms. To have a graphical interpretation of the starting populations, 5 of the 20 candidates are selected and generated using CAD and shown in Figure 4.1.

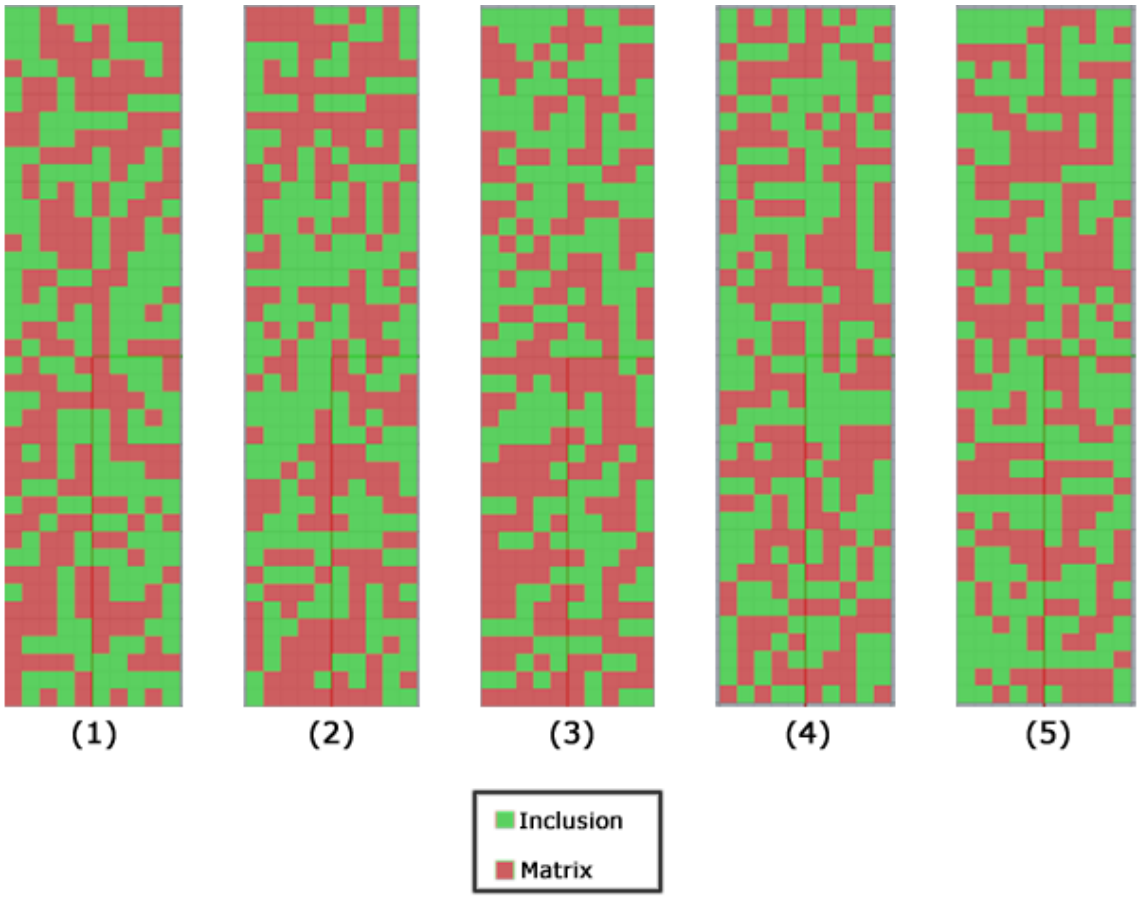


Figure 4.1: Five candidates of the starting population.

Candidate (5) had the best objective value of 2828.5 for the starting generation, and acted as a starting point for the algorithms.

Results

Once the best parameters for each algorithm were found, each was ran for another 100 iterations. The results are presented in Table 4.4.

Table 4.4: Final Objective Testing

Algorithms	Objective		Obj. Func. Value	Time Taken (mins)
	$f_1^{\text{calc}} - f_1^{\text{target}}$	$f_2^{\text{calc}} - f_2^{\text{target}}$		
GA	26.09	6.31	648.50	70.62
PSO	42.25	4.05	1801.5	81.52
MOGA	43.06	1.87	1857.3	106.99

Overall the best performing algorithm is the Genetic Algorithm, with the final objective value of 648.5 which is more than 50% lower than the next best algorithm. In terms of time consumption, Genetic Algorithm and Particle Swarm both take an hour and 15 minutes to solve whereas for MOGA takes two hours to solve. However, the time consumptions of these algorithms are dependent on the speed of the computer that it is running on. Our simulations are run on a 16 RAM Dual Core computer.

Using the plot of Genetic Algorithm objective against iterations as an example in Figure 4.2. It can be observed that from iteration 60 to iteration 90 there has been no improvement of

the objective value. This suggests most of the candidates in the population has converged to the current best solution, which means the candidates are breeding with identical solution causing the next generation to have the same candidates as before. At iteration 92 the best objective started improving again due to random mutation of the candidate.

Hence, the algorithm became inefficient once the candidates in the population became similar as the only way of improving the objective value is to wait for a good mutation to appear which is dependent on chance. To have a more efficient algorithm, restarts should be implemented where every time the objective values stop improving for a certain number of iterations.

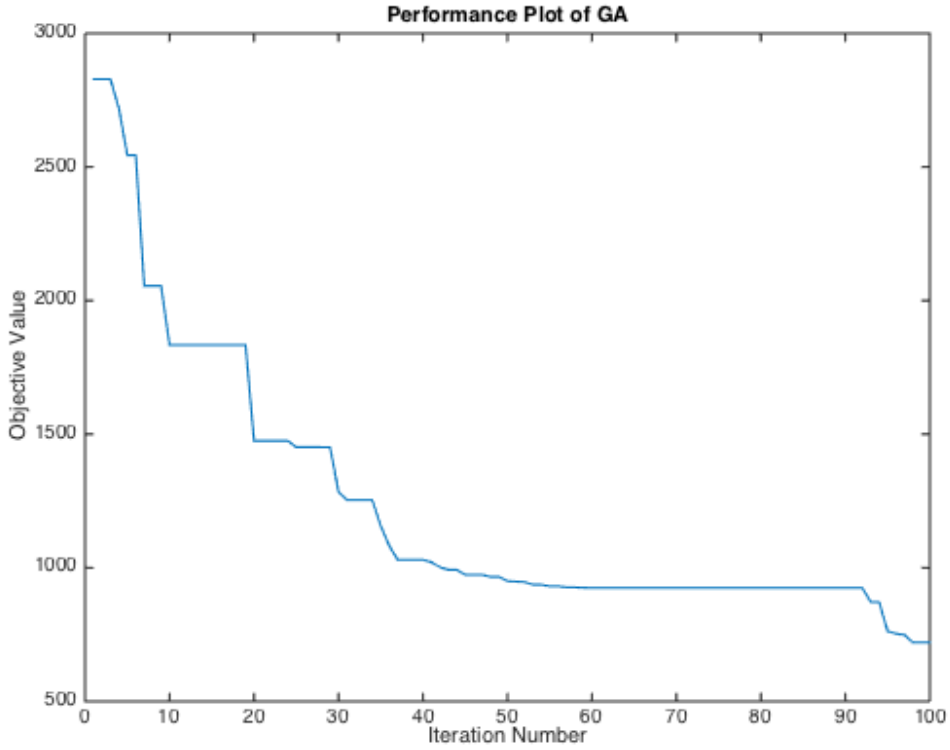


Figure 4.2: A plot showing the variation of objective function against iteration number.

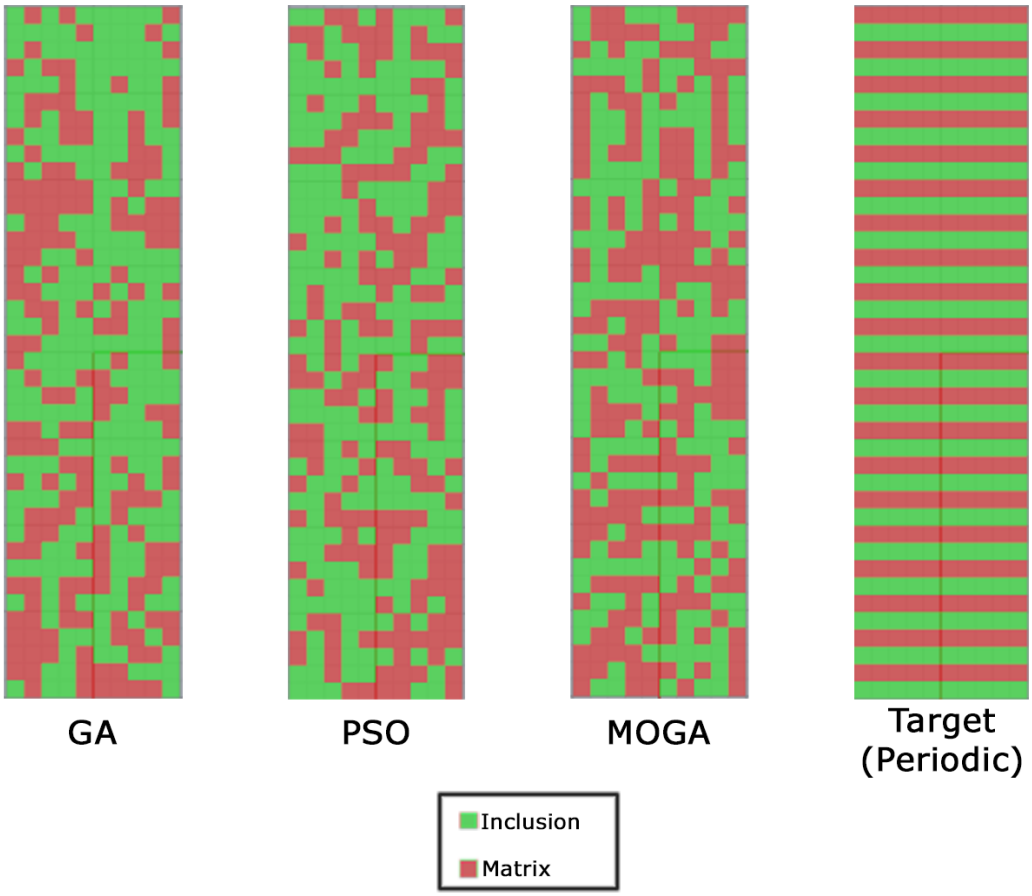


Figure 4.3: Model of best solutions using different algorithms.

Figure 4.3 above shows the solutions that each algorithm has converged to using the specified starting population. Comparing each solution beam to the target beam, it is hard to find any resemblance. This suggests there might exist multiple solutions for the same set of frequencies. As the test only consists of matching the first two natural frequencies, this can be highly possible.

Chapter 5

Phase 2 Results

5.1 Natural Frequencies of Composite Materials

Using the setup described for phase 2 in Chapter 3, for each target frequency set, a model with optimised material assignment is generated.

The algorithm is allowed to restart 3 times or maximum of 400 iterations and the best solution of these are taken.

Table 5.1: Natural Frequencies of Solution Sets

n	Set 1	Set 2	Set 3	Set 4	Set 5	Periodic
1	36.91	59.32	97.89	137.29	159.91	12.78
2	218.26	336.99	564.07	824.39	1001.7	73.92
3	219.18	371.21	564.50	839.61	1013.3	89.27
4	264.09	432.97	679.45	991.10	1205.0	138.59
5	618.80	964.54	1625.4	2338.2	2830.0	198.25
6	691.05	1102.6	1842.9	2600.2	3144.9	322.28

Table 5.1 shows the natural frequencies obtained using the calculated model. By comparing this to the target frequencies listed in Table 3.2, the error at every node can be evaluated and studied. The absolute error is shown in Table 5.2, where the relative error is shown in brackets.

Table 5.2: Error of Solution Set

n	Set 1 Err	Set 2 Err	Set 3 Err	Set 4 Err	Set 5 Err	Periodic Err
1	1.06(2.96%)	2.45(1.30%)	3.87(4.11%)	3.06(2.28%)	2.21(1.36%)	0.93(7.85%)
2	1.14(0.52%)	10.6(3.05%)	14.2(2.45%)	3.50(0.42%)	0.70(0.07%)	0.88(1.22%)
3	4.20(1.88%)	16.8(4.75%)	21.5(3.67%)	2.91(0.35%)	2.80(0.28%)	1.61(1.77%)
4	1.68(0.64%)	13.6(3.24%)	17.5(2.50%)	6.17(0.62%)	0.50(0.04%)	0.25(0.18%)
5	1.24(0.20%)	22.9(2.32%)	11.6(0.71%)	1.80(0.08%)	2.60(0.09%)	1.01(0.51%)
6	3.87(0.56%)	7.30(0.67%)	26.0(1.43%)	2.30(0.09%)	5.50(0.18%)	0.75(0.23%)

We can see that the algorithm did not do so well for the target frequencies of the periodic beam , which has the highest error of 7.848% at the first node. However, other than frequency set of periodic beam, others has remained an error percent below 5%. We can also observe a pattern from the table where the first frequency will always have the highest error despite having the highest penalty. This suggests that the first natural frequency domain is the hardest to match. Also as the frequencies get higher the relative error becomes less justifying, as higher values

naturally have lower relative error than lower values having the same absolute error. Hence it makes more sense to look at the absolute error as we increase in order of natural frequencies.

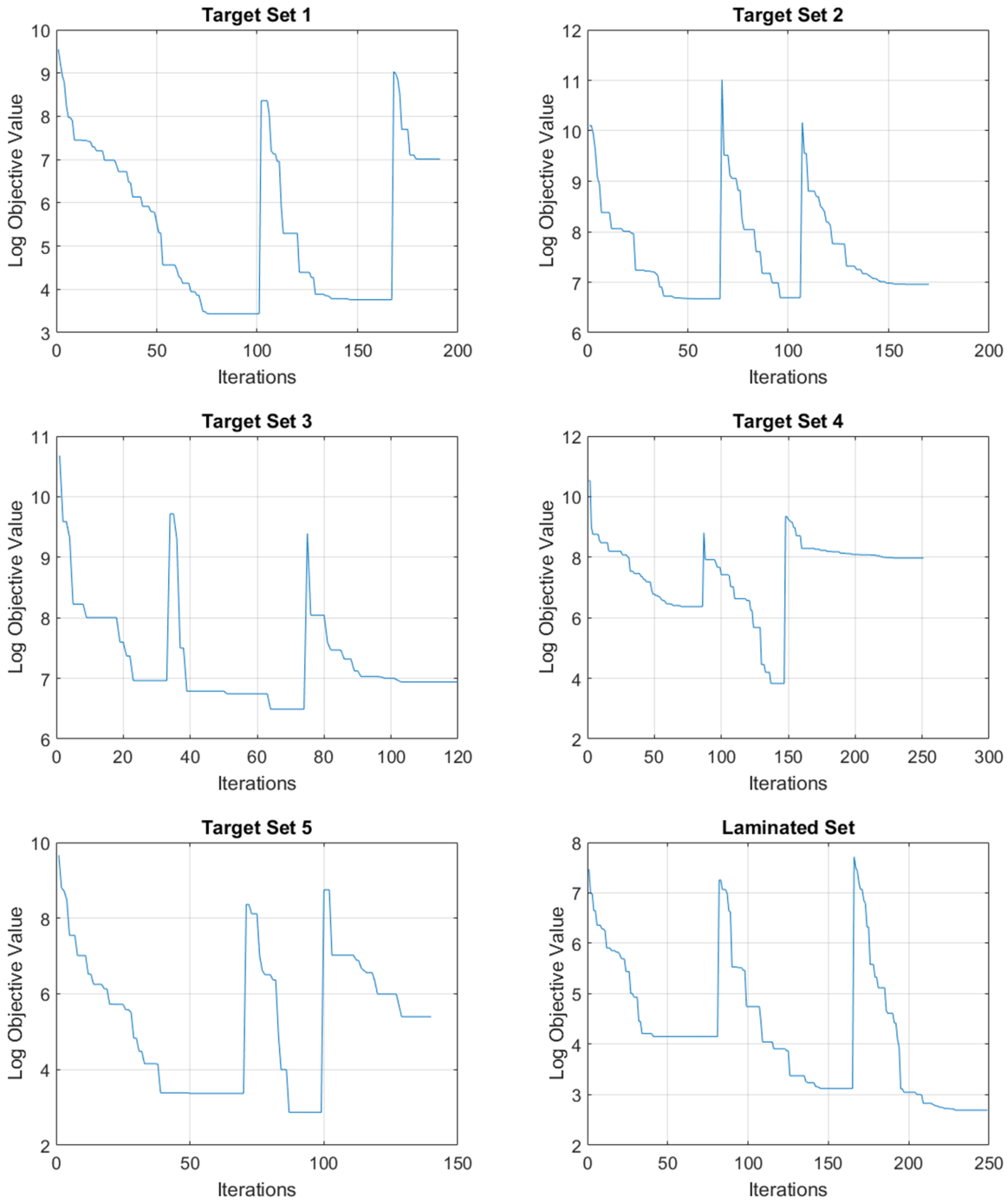


Figure 5.1: Performance plot for each target frequencies.

Figure 5.1 above shows the natural logged objective value variation as iteration number increases. There is an increase in objective function due to the algorithm resetting the population after the objective function stalling for 10 iterations. Additionally, by having the restarts, it has further improved the convergence rate of the algorithm.

5.2 Optimised Beam Examples

The figure below shows the optimised models that was generated using our algorithm. The top has a fixed boundary condition and the bottom is left free.

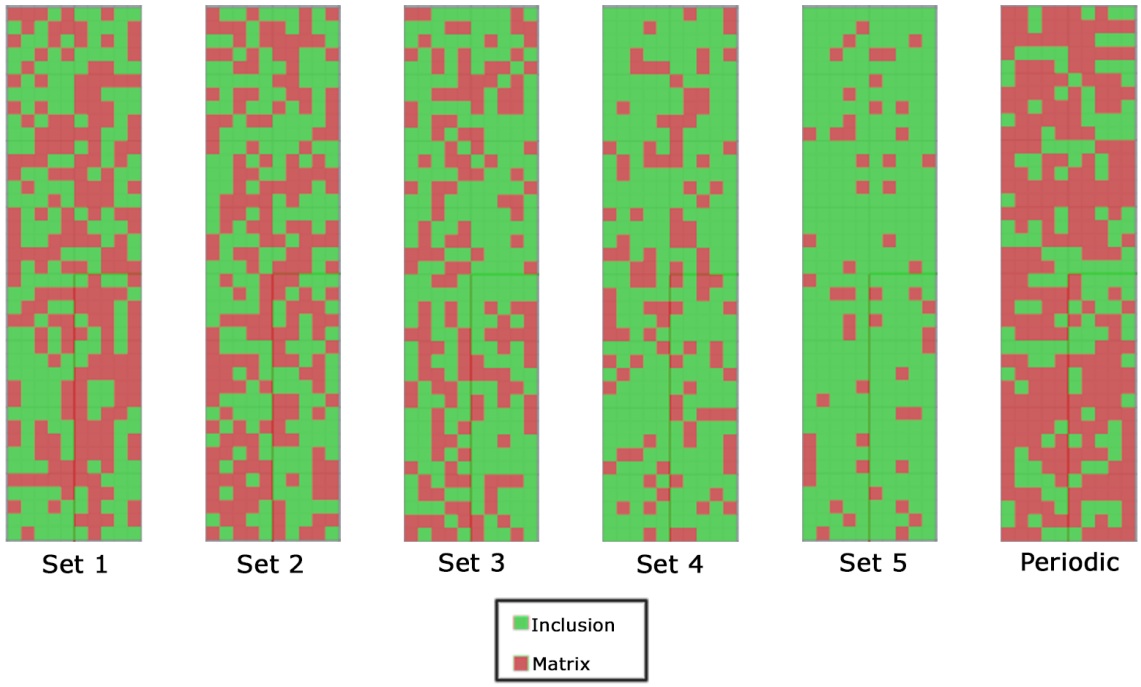


Figure 5.2: Optimised Beam Models.

As these beams are generated with sets of target frequencies that are randomly generated, there is no exact model to compare it to. However, due to the way that the target frequencies are generated, we know that the sets of target frequencies should be spread over the feasible objective space. This means the structures of these target frequencies should experience a decrease in inclusions as we move from set 1 to 5. This can be seen from the solutions models in Figure 5.2.

5.3 Starting Population Examples

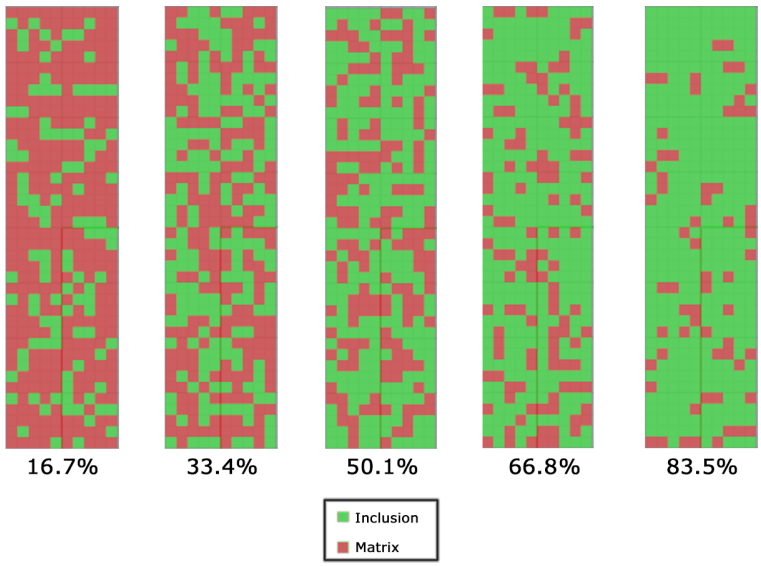


Figure 5.3: Five Candidates of the starting population with increasing number of inclusions.

Five candidates from the starting population that was generated using the updated population

generation function is selected and visualised to give a indication of the structural appearance. Comparing to Figure 5.2, we realise that each of the final models have a candidate that is to some degree similar. This validate the point of having a spread of candidates in the population, as we can see these directing the algorithm to the final solution.

Some of the tested samples are then 3D printed to see the physical difference between the digital model beams and the printed beams. Pictures of the printed beam can be found in Appendix C.

Chapter 6

Discussions

The chapters above describes a toolchain that can be used to generate a desired material property of an object without changing the existing topological design by optimising the assignment of different materials to voxels using Genetic Algorithm. The solutions for natural frequencies generated by the toolchain shown previously indicates that the system is fully capable of producing structures that satisfy the requirement of with a decent accuracy.

This toolchain can provide us opportunities to generate more advanced material property distributions hence enhancing the performance of the object. Using the conventional DTM and traditional CAD tools, this would not be achievable. By voxelating the object, the material can be assigned based on a voxel level using an optimisation algorithm. This is essential to allow the design of advanced material properties. The overview of our whole methodology is shown in Figure 2.1.

6.1 Voxelisation

Due to the lack of implementation of voxelistion in traditional CAD packages and the time consumption of constructing each voxel manually. We invented a method that can be fully automated, by creating points in Matlab and generating voxels at each point. Using these methods, simples structures can be voxelised quickly with a click of a button.

However the limitations of this approach are rather obvious, to successfully generate a voxelised object, a mathematical description of the model in 3D space is required. Hence, when voxelising complex structures it would be difficult to find the related function. Also by using a third party mathematical programming software, it defeats the purpose of being able to construct objects using CAD packages freely.

6.2 Accuracy and Speed

When it comes to the performance of a toolchain, speed and accuracy should be the most important factor. For our methodology, this is controlled by the optimisation algorithm. We have conducted 2 phases to tune and examine the accuracy of the algorithm.

Phase 1

In phase 1, we have discovered the that the performance of Genetic Algorithm is better than the other algorithms by 50%. This suggests that the ability of GA to take advantages of the

successful candidates by crossovering them to generate next generation, while eliminating the weaker candidates by restricting it reproduction is a better approach to the problem.

Studying the sensitivity analysis of the genetic algorithm, we notice that the algorithm is more sensitive to the crossover rate than the mutation rate. The average objective function values of genetic algorithm sensitivity analysis in Table 4.2 is plotted against the crossover rate.

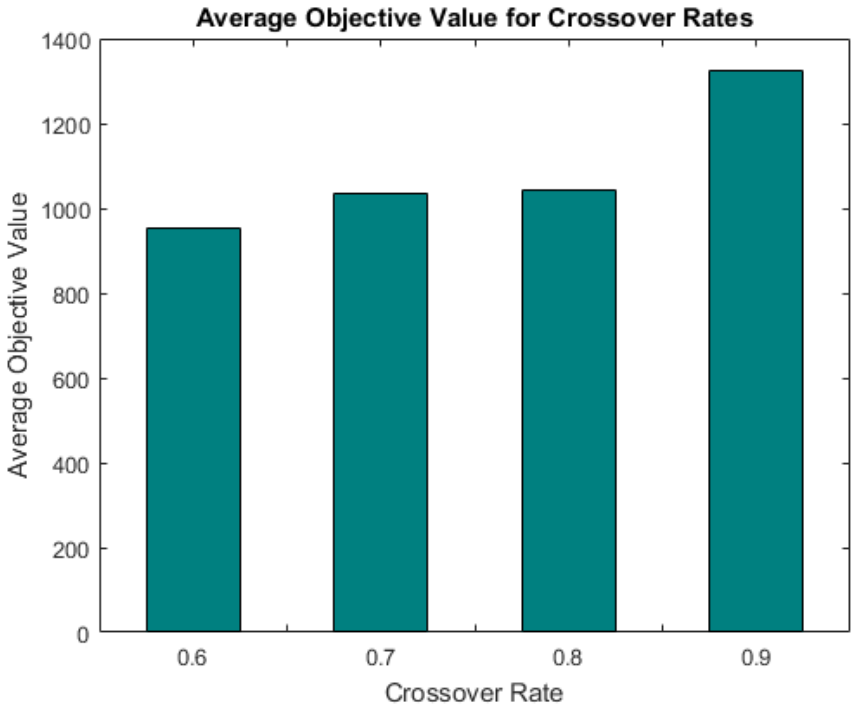


Figure 6.1: A plot showing the effect of Crossover rate on the objective function.

As we can see in figure 5.1 that as we decrease the crossover rate, the objective function values falls from 1324 at $p_c = 0.9$ to 954 at $p_c = 0.6$.

Phase 2

After tuning the algorithm, it is then tested using 5 sets randomly generated target frequencies and a periodic frequency set. The result shows that the solution is able to converge to solutions that are within 5% of the given frequencies. Existing structural optimisation strategies such as homogenisation only allow one or a few frequencies and depends heavily on the local gradients. Using our methodology, we are able to converge to many frequencies at once.

We have also noticed that the final objective solutions of restarts are not dependent on the starting objective values of the solutions but more to how the voxels are arranged. This can be visualised using the plot for target set 4 in Figure 5.1 which is enlarged in Figure 5.2.

We can see from the first restart of the algorithm with an initial log objective value of 8.4 find the overall best solution which had a logged objective value around 4. After 10 stalling iterations, the algorithm restarts again with an initial log objective of 8.6 which is similar to the first restart. However, this time it has only managed to converge to a log objective value of 8. This suggests that the initial objective of the second restart despite having an initial objective value similar to the previous restart, the material arrangements of the candidates have led the algorithm to fall in a local minimum.

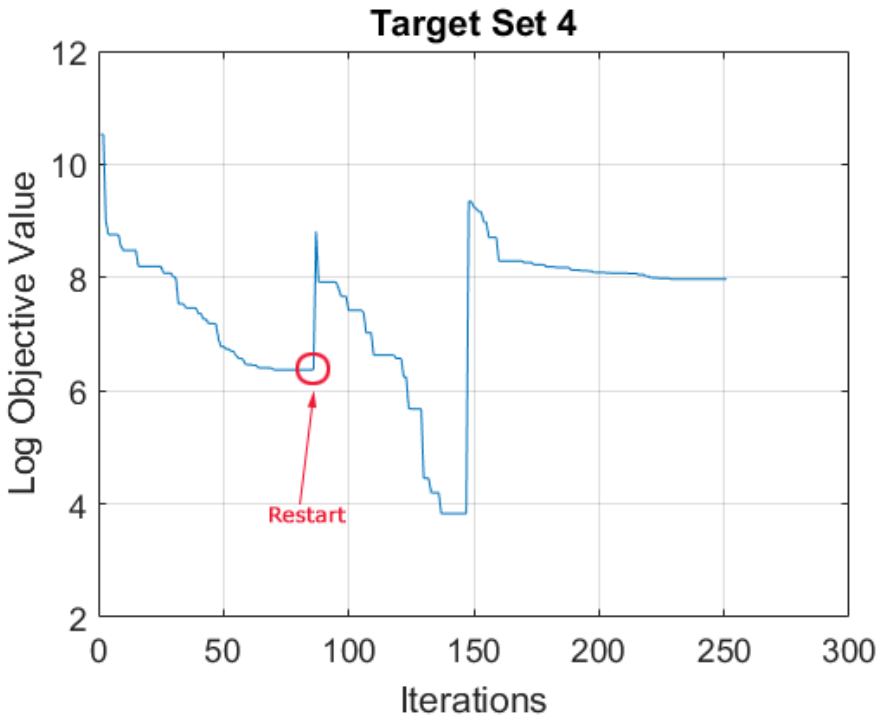


Figure 6.2: Performance plot of target frequency set 4.

Limitations

The limitation of the optimisation method is the time consumption of the algorithm. Due to the fact that the solution is not guaranteed to be optimum several runs needs to be conducted before a good solution can be found. Referring back to the random target frequency test that was described in Chapter 5, each restart took around 180 minutes to complete. The time consumption shows a linear relationship with the number of calculation performed. Hence We have tried to decrease the time consumption by decreasing the mesh size and changing the FEM discretisation. However, this can only decrease the time consumption by a limited amount, which means it can still take a long time to solve.

6.3 Visualisation

Visualisation is potentially the most interesting part of the results and could potentially be useful in terms of research. For example, by observing the model of the solution we have obtained for the periodic beam, we notice that the structure is not the same as the original periodic model. This shows that there can be multiple solutions to the same natural frequency problem, which enables more design space in the future engineering field.

Chapter 7

Conclusions

In this report, we presented a new approach to material property distribution controlling by using the combination of voxelisation and optimisation and the product can be fabricated using a multi material 3D printing method. This method completely opens up a new design space of 3D printing and even manufacturing, without having to worry about the having to change the topology of a object to obtain a certain property.

To illustrate and validate our method, we applied to natural frequency control to a cantilever beam, for which the beam is broken down into voxels and materials are assigned to each voxels. Our aim is to find the best optimisation algorithm that can be used for material distribution properties, and employ that algorithm for working our different problems. The frequencies of the beams are calculated using finite difference method. This demonstrated the ability to optimise a property of need, as well as the differences of material assignment as the target changes. More importantly, by validating our method, it also shows promises in the field of 3D printing is no longer only used for rapid prototyping but soon be a full new design space where models can be designed digitally rapidly generated to a product of use.

7.1 Future Work

Given the current status of multi-material printing, the global scope for future work is enormous. Even just considering our project scope, there are many points to focus for future work.

In the report, we have only described optimising for a set of natural frequencies. Other properties are should be tested and compared to see the robustness of our method for different properties. In reality most applications would require more than two types of materials, hence it is essential to make sure that the method works more multiple materials. We would also like to test the limits of this optimisation method, trying to optimise more natural frequencies until the system fails.

New methods of voxelisation would also be examined to find a better approach to the concept. Most of the existing voxelisers are only voxelising the surface of a model, however to change the property distribution this is clearly not enough. Hence, a method is required that voxelises the whole model which then can be read using a FEM calculator.

Appendix A

Process Diagrams

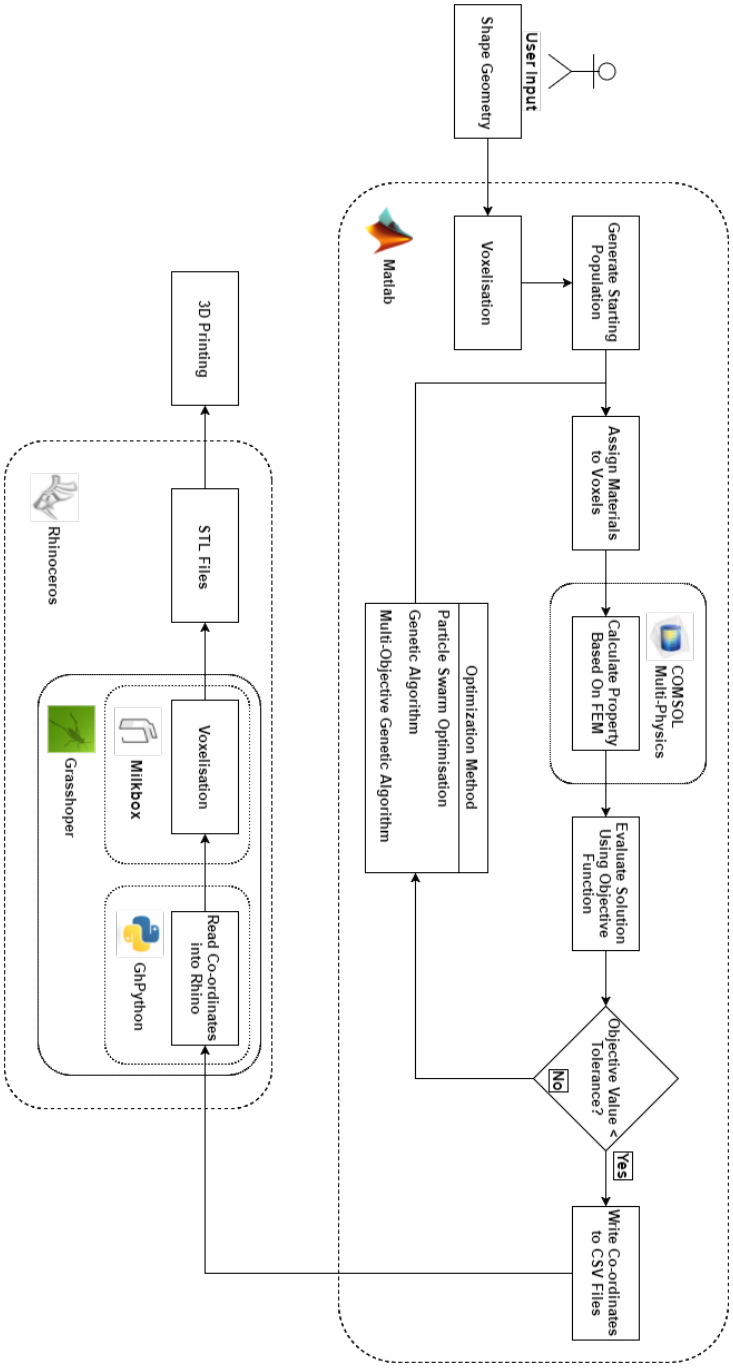


Figure A.1: Flowchart of the overall procedure

Appendix B

Matlab Codes

B.1 General Code

Objective Function

```
function [ fx ] = CalcNatFreq( pos )
global model target
import com.comsol.model.*
import com.comsel.model.util.*
pos1 = [];
pos2 = [];
n = length(target);
fx = zeros(1,n);

for i = 1:length(pos)
    if pos(i) == 1
        pos1= [pos1, i];
    else
        pos2 = [pos2, i];
    end
end

model.material('mat1').selection.set(pos1);
model.material('mat2').selection.set(pos2);

model.sol('sol1').runAll;
nat_freq = mphglobal(model,'solid.freq');

% Phase 1 Objective Function
fx = (nat_freq(1:2) - [128; 210]).^2;

% Phase 2 Objective Function
% for i = 1:6
%     w = (n - i + 1)/n;
%     fx(i) = + w*(target(i) - nat_freq(i))^2;
% end

fx = sum(fx);

t= toc;

s = sprintf('The objective function is: %f\n',fx,...
    'The natural frequencies are:[%d, %d]\n', nat_freq(1), nat_freq(2), ...
    'Time elapsed: %.2fmins\n',  t/60);
```

```
disp (s)
end
```

Comsol Model Creation

```
function out = create_model
%
% block.m
%
% Model exported on Jun 7 2016, 11:55 by COMSOL 4.3.0.151.

% clear all; clc;

import com.comsol.model.*
import com.comsol.model.util.*
model = ModelUtil.create('Model');
model.modelPath('H:\Documents\Project Code\Binary Genetic Algorithm');
model.modelNode.create('mod1');
model.geom.create('geom1', 3);
model.mesh.create('mesh1', 'geom1');
model.physics.create('solid', 'SolidMechanics', 'geom1');
model.study.create('std1');
model.study('std1').feature.create('eig', 'Eigenfrequency');
model.study('std1').feature('eig').activate('solid', true);
model.geom('geom1').lengthUnit('mm');

material1 = [];
material2 = [];
counter = 1;
for j = 0:39
    for i = 0:9
        for z = 0:0
            string = sprintf('Block');
            blockName = sprintf('blk%d',counter);
            if rem(j,2) == 0
                material1 = [material1 counter];
            else
                material2 = [material2 counter];
            end
            L = sprintf('%d',j);
            W = sprintf('%d',i);
            H = sprintf('%d',z);
            model.geom('geom1').feature.create(blockName,string);
            model.geom('geom1').feature(blockName).setIndex('pos',L,0);
            model.geom('geom1').feature(blockName).setIndex('pos',W,1);
            model.geom('geom1').feature(blockName).setIndex('pos',H,2);
            counter = counter + 1;
        end
    end
end

model.name('block.mph');

model.geom('geom1').run;


model.material.create('mat1');
model.material('mat1').name('Tango+');
model.material('mat1').set('family', 'custom');
```

```
model.material('mat1').propertyGroup('def').set('density', '1120[kg/m^3]');
model.material('mat1').propertyGroup.create('Enu', ...
    'Young''s modulus and Poisson''s ratio');
model.material('mat1').propertyGroup('Enu').set('poissonsratio', '0.48');
model.material('mat1').propertyGroup('Enu').set('youngsmodulus', '9.2e5[Pa]');

model.material.create('mat2');
model.material('mat2').name('Vero White');
model.material('mat2').set('family', 'custom');
model.material('mat2').propertyGroup('def').set('density', '1170[kg/m^3]');
model.material('mat2').propertyGroup.create('Enu', ...
    'Young''s modulus and Poisson''s ratio');
model.material('mat2').propertyGroup('Enu').set('poissonsratio', '0.35');
model.material('mat2').propertyGroup('Enu').set('youngsmodulus', '2.3e9[Pa]');

model.material('mat1').selection.set(material1);
model.material('mat2').selection.set(material2);

model.physics('solid').feature.create('fix1', 'Fixed', 2);
model.physics('solid').feature('fix1').selection.set([1 5 9 13 17 21 25 ...
    29 33 37]);

model.mesh('mesh1').feature.create('swe1', 'Sweep');
model.mesh('mesh1').feature('swe1').selection('sourceface').set([1 5 9 ...
    13 17 21 25 29 33 37]);
model.mesh('mesh1').feature('swe1').selection('targetface').set([1641 ...
    1642 1643 1644 1645 1646 1647 1648 1649 1650]);
model.mesh('mesh1').run;


model.physics('solid').prop('ShapeProperty').set('order_displacement', 1, '1');

model.sol.create('sol1');
model.sol('sol1').study('std1');
model.sol('sol1').feature.create('st1', 'StudyStep');
model.sol('sol1').feature('st1').set('study', 'std1');
model.sol('sol1').feature('st1').set('studystep', 'eig');
model.sol('sol1').feature.create('v1', 'Variables');
model.sol('sol1').feature('v1').set('control', 'eig');
model.sol('sol1').feature.create('e1', 'Eigenvalue');
model.sol('sol1').feature('e1').set('shift', '0');
model.sol('sol1').feature('e1').set('neigs', '6');
model.sol('sol1').feature('e1').set('transform', 'eigenfrequency');
model.sol('sol1').feature('e1').set('control', 'eig');
model.sol('sol1').attach('std1');

model.result.create('pg1', 3);
model.result('pg1').set('data', 'dset1');
model.result('pg1').feature.create('surf1', 'Surface');
model.result('pg1').feature('surf1').set('expr', {'solid.disp'});
model.result('pg1').name('Mode Shape (solid)');
model.result('pg1').feature('surf1').feature.create('def', 'Deform');
model.result('pg1').feature('surf1').feature('def').set('expr', {'u' 'v' 'w'});
model.result('pg1').feature('surf1').feature('def').set('descr', ...
    'Displacement field (Material)');


model.save('H:\Documents\Project Code\Binary Genetic Algorithm\test_block');

out = model;
```

Voxelisation for Rhinoceros

```
% for 10 times
M2GHfileName = 'H:\Matrix.csv';
inclusionFile = 'H:\Inclusions.csv';

x = 40;
y = 10;
z = 1;
s = 1;

% For periodic beam
% pos=[];
% counter = 1;
% for j = 0:39
%     for i = 0:9
%         for k = 0:0
%             if rem(j,2) == 0
%                 pos = [pos 1];
%             else
%                 pos = [pos 0];
%             end
%             counter = counter + 1;
%         end
%     end
% end

% posbinary = BestSol.Position;
% pos = [];
% for i = 1:length(posbinary)
%     if posbinary(i) == 1
%         pos = [pos i];
%     end
% end

[inclusions, matrix] = CalcInclusions(x,y,z,s,pos);


% csvwrite(M2GHfileName, parametres);
csvwrite(inclusionFile, inclusions);
csvwrite(M2GHfileName, matrix);


function [ result_inc, result_mat ] = CalcInclusions( x, y, z, s , pos)
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here

l = linspace(-x/2,x/2,x/s+1);
w = linspace(-y/2,y/2,y/s+1);
h = linspace(-z/2,z/2,z/s+1);

% handlex = linspace(-x/2-9,-x/2,11);
% handley = linspace(-y/2,y/2,y/s+1);
% handlez = linspace(-z/2,z/2,z/s+1);

result_mat = [];
result_inc = [];

% Calculate gradient distribution along beam
% grad = linspace(0,1,length(l)-1);
```

```
% grad = repmat(1,1,length(l)-1);
% ele = (length(w)-1)*(length(h)-1);

% for i = 1: length(l)-1
%     counter = 1;
%     matrix = [];
%     for j = 1:length(w)-1
%         for k = 1:length(h)-1
%             inclusions(counter,:) = [l(i),w(j),h(k)];
%             counter = counter+1;
%         end
%     end
%     for n = 1:round(grad(i)*ele)
%         [o,~] = size(inclusions);
%         ran = randi([1 o],1,1);
%         matrix(n,:) = inclusions(ran,:);
%         inclusions(ran,:) = [];
%     end
%     result_mat = vertcat(result_mat,matrix);
%     result_inc = vertcat(result_inc,inclusions);
% end

% IntPos assignment
% counter = 1;
% count = 1;
% for i = 1:length(l)-1
%     for j = 1:length(w)-1
%         for k = 1:length(h)-1
%             if count > length(pos)
%                 return
%             elseif pos(count) == counter
%                 result(count, :) = [l(i),w(j),h(k)];
%                 count = count + 1;
%             end
%             counter = counter + 1;
%         end
%     end
% end

% BinaryPos assignment

counter = 1;
for i = 1:length(l)-1
    for j = 1:length(w)-1
        for k = 1:length(h)-1
            if pos(counter) == 1
                result_inc = [result_inc; l(i),w(j),h(k)];
            else
                result_mat = [result_mat; l(i),w(j),h(k)];
            end
            counter = counter + 1;
        end
    end
end

% for i = 1:length(handlex)-1
%     for j = 1:length(handley)-1
%         for k = 1:length(handlez)-1
%             result_inc = [result_inc; handlex(i), handley(j),handlez(z)];
%
```

```
%         end
%     end
% end


end
```

B.2 Phase 1 Code

Genetic Algorithm

```
clc;
clear all;
close all;
global model
import com.comsol.model.*
import com.comsel.model.util.*
model = mphload('test_block.mph');
model.hist.disable

%% Problem Definition

CostFunction=@(x) CalcNatFreq(x);     % Cost Function

nVar=400;            % Number of Decision Variables

VarSize=[1 nVar];   % Decision Variables Matrix Size

%% GA Parameters
tic;

MaxIt= 50;  % Maximum Number of Iterations

Tol = 440*0.005;

nPop=21;    % Population Size

ANSWER=questdlg('Choose selection method:','Genetic Algorith',...
    'Roulette Wheel','Tournament','Random','Roulette Wheel');

UseRouletteWheelSelection=strcmp(ANSWER,'Roulette Wheel');
UseTournamentSelection=strcmp(ANSWER,'Tournament');
UseRandomSelection=strcmp(ANSWER,'Random');

if UseRouletteWheelSelection
    beta=8;         % Selection Pressure
end

if UseTournamentSelection
    TournamentSize=3;    % Tournamnet Size
end
%% Initiate population

empty_individual.Position=[];
empty_individual.Cost=[];

ini_pop=repmat(empty_individual,nPop,1);

for i=1:nPop
```

```
    % Initialize Position
    ini_pop(i).Position=randi([0 1],VarSize);

    % Evaluation
    ini_pop(i).Cost=CostFunction(ini_pop(i).Position);
end
% Array to Hold Best Cost Values
BestCost=zeros(MaxIt,50);

% pause(0.01); % Needed due to a bug in older versions of MATLAB
count = 1;
for x = 1:4
    for y = 1:4
        crossover = [0.9, 0.8, 0.7, 0.6];
        mute = [0.4,0.3,0.2,0.1];
        pc=crossover(x);                    % Crossover Percentage
        nc=2*round(pc*nPop/2);  % Number of Offsprings (also Parnets)

        pm= mute(y);                    % Mutation Percentage
        nm=round(pm*nPop);      % Number of Mutants
        mu=0.1;                 % Mutation Rate

        prevsol = 0;
        rep = 1;
        %% Initialization
        % Sort Population
        pop = ini_pop;

        Costs=[pop.Cost];
        [Costs, SortOrder]=sort(Costs);
        pop=pop(SortOrder);

        % Store Best Solution
        BestSol(count)=pop(1);



        % Store Cost
        WorstCost=pop(end).Cost;

        %% Main Loop
        % it = 1;
        %plot
        hold on
        h = animatedline('Marker','o');
        axis([0 50 -inf inf])
        xlabel('Iterations')
        ylabel('Solution Value')
        for it=1:MaxIt
            % Calculate Selection Probabilities
            if UseRouletteWheelSelection
                P=exp(-beta*Costs/WorstCost);
                P=P/sum(P);
            end

            % Crossover
            popc=repmat(empty_individual,nc/2,2);
            for k=1:nc/2
```

```
        % Select Parents Indices
        if UseRouletteWheelSelection
            i1=RouletteWheelSelection(P);
            i2=RouletteWheelSelection(P);
        end
        if UseTournamentSelection
            i1=TournamentSelection(pop,TournamentSize);
            i2=TournamentSelection(pop,TournamentSize);
        end
        if UseRandomSelection
            i1=randi([1 nPop]);
            i2=randi([1 nPop]);
        end

        % Select Parents
        p1=pop(i1);
        p2=pop(i2);

        % Perform Crossover
        [popc(k,1).Position, popc(k,2).Position]= ...
            Crossover(p1.Position,p2.Position);

        % Evaluate Offsprings
        popc(k,1).Cost=CostFunction(popc(k,1).Position);
        popc(k,2).Cost=CostFunction(popc(k,2).Position);

    end
    popc=popc(:);


    % Mutation
    popm=repmat(empty_individual,nm,1);
    for k=1:nm

        % Select Parent
        i=randi([1 nPop]);
        p=pop(i);

        % Perform Mutation
        popm(k).Position=Mutate(p.Position,mu);

        % Evaluate Mutant
        popm(k).Cost=CostFunction(popm(k).Position);

    end

    % Create Merged Population
    pop=[pop
        popc
        popm]; %#ok

    % Sort Population
    Costs=[pop.Cost];
    [Costs, SortOrder]=sort(Costs);
    pop=pop(SortOrder);

    % Update Worst Cost
    WorstCost=max(WorstCost,pop(end).Cost);

    % Truncation
```

```
            pop=pop(1:nPop);
            Costs=Costs(1:nPop);

            % Store Best Solution Ever Found
            BestSol(count)=pop(1);

            % Store Best Cost Ever Found
            BestCost(count, it)=BestSol(count).Cost;

            % Show Iteration Information
            disp(['Iteration ' num2str(it) ': Best Cost = ' ...
                num2str(BestCost(count, it))]);

            %     it= it+1;

            if prevsol == BestSol(count).Cost
                rep = rep + 1;
            else
                rep = 1;
            end

            prevsol = BestSol(count).Cost;

            addpoints(h,it,BestCost(count,it))
            s = sprintf('Best Solution Value %f', BestCost(count,it));
            title(s)
            drawnow;

            %             if rep > 5
            %                 break
            %             end
        end
        hold off
        count = count + 1;
    end
end


%% Results
figure;
plot(BestCost,'LineWidth',2);
xlabel('Iteration');
ylabel('Cost');
grid on;
```

Particle Swarm Optimisation

```
function [xOpt,fval,exitflag,output,population,scores] = ...
    psobinary(fitnessfcn,nvars,options)
% Particle swarm optimization for binary genomes.
%
% Syntax:
% psobinary(fitnessfcn,nvars)
% psobinary(fitnessfcn,nvars,options)
%
% This function will optimize fitness functions where the variables are
% row vectors of size 1xnvars consisting of only 0s and 1s.
%
% PSOBINARY is provided as a wrapper for PSO, to avoid any confusion. This
% is because the binary optimization scheme is not designed to take any
```

```
% constraints. PSOBINARY does not allow the passing of constraints. It
% takes a given optimization problem with binary variables, and
% automatically sets the options structure so that 'PopulationType'
% is 'bitstring'.
%
% This has exactly the same effect as setting the appropriate options
% manually, except that it is not possible to unintentionally define
% constraints, which would be ignored by the binary variable optimizer
% anyway.
%
% Problems with hybrid variables (double-precision and bit-string
% combined) cannot be solved yet.
%
% The output variables for PSOBINARY is the same as for PSO.
%
% See also:
% PSO, PSOOPTIMSET, PSODEMO

global model
import com.comsol.model.*
import com.comsel.model.util.*
model = mphload('test_block.mph');
model.hist.disable
nvars = 400;
fitnessfcn=@CalcNatFreq;

if ~exist('options','var') % Set default options
    options = struct ;
end % if ~exist

c1 = [0.5,1,2];
c2 = c1;

for i = 1:3
    for j = 1:3
        options = psooptimset(options,'PopulationType',...
            'bitstring','SocialAttraction',c2(i),...
            'CognitiveAttraction',c1(j)) ;
        [xOpt,fval(i,j),exitflag,output,population,scores] = ...
            pso(fitnessfcn,nvars,[],[],[],[],[],[],[],options) ;
    end
end
```

Multi-objective Genetic Algorithm

```
%% GAMULTIOBJ with integer constraints
%
global model
import com.comsol.model.*
import com.comsel.model.util.*
model = mphload('test_block.mph');
model.hist.disable
tic

%% Problem setup
fitnessFunction = @CalcNatFreq;  % Function handle to the fitness function
numberOfVariables = 400;    % Number of decision variables
populationSize = 21;
stallGenLimit = 10;
generations = 50;
```

```
% Bound Constraints
ub = ones(1,400);
lb = zeros(1,400);
Bound = [lb; ub];

%% Solve the problem with integer constraints
%Add one of the following lines to each of the custom functions:
%IntCon = [];       - no integer constraints
%IntCon = 1;        - x(1) of the solution will be integer
%IntCon = 2;        - x(2) of the solution will be integer
%IntCon = [1, 2];   %- both cordinates of the solution will be integer
f1 = zeros(4,8,2);
c = [0.6, 0.7, 0.8, 0.9];
for i = 1:4
        figure;
        options = gaoptimset('PopulationSize',populationSize,...
            'CreationFcn', @int_pop,...
            'MutationFcn', @int_mutation,...
            'CrossoverFcn',@int_crossoverarithmetic,...
            'StallGenLimit', stallGenLimit,...
            'Generations', generations,...
            'PopulationSize',populationSize,...
            'PlotFcn',{@gaplotpareto,@gaplotbestf},...
            'CrossoverFraction', c(i),...
            'Display','iter',...
            'PopInitRange', Bound);

        [x1, f1(i,:,:), exitflag1, output1, population1, score1] =...
            gamultiobj(fitnessFunction,...
            numberOfVariables, [], [], [], [], lb, ub, options);
end
```

B.3 Phase 2 Code

Main code

```
clc;
clear;
close all;
global model target
import com.comsol.model.*
import com.comsel.model.util.*
model = mphload('test_block.mph');
model.hist.disable

n = 5;

lb = [4.5794, 23.002, 28.443, 28.853, 73.382, 79.891];
ub = [183.64, 1134.6, 1144.7, 1366.3, 3203.7, 3557.4];

targets = Gen_target(n, ub, lb);



%% Problem Definition

CostFunction=@(x) CalcNatFreq(x);     % Cost Function
```

```
nVar=400;            % Number of Decision Variables

VarSize=[1 nVar];   % Decision Variables Matrix Size

%% GA Parameters

MaxIt= 400; % Maximum Number of Iterations

Tol = 10;

nPop=21;    % Population Size

pc=0.7;                  % Crossover Percentage
nc=2*round(pc*nPop/2);  % Number of Offsprings (also Parnets)

pm=0.4;                  % Mutation Percentage
nm=round(pm*nPop);       % Number of Mutants
mu=0.1;                 % Mutation Rate

ANSWER=questdlg('Choose selection method:','Genetic Algorith',...
    'Roulette Wheel','Tournament','Random','Roulette Wheel');

UseRouletteWheelSelection=strcmp(ANSWER,'Roulette Wheel');
UseTournamentSelection=strcmp(ANSWER,'Tournament');
UseRandomSelection=strcmp(ANSWER,'Random');

if UseRouletteWheelSelection
    beta=8;         % Selection Pressure
end

if UseTournamentSelection
    TournamentSize=3;    % Tournamnet Size
end

% Array to Hold Best Cost Values
BestCost=zeros(MaxIt,n);

% pause(0.01); % Needed due to a bug in older versions of MATLAB

for j = 1:n
    %% Initialization
    target = targets(j,:);
    tic;
    empty_individual.Position=[];
    empty_individual.Cost=[];
    pop = create_population(nPop, CostFunction);

    % Sort Population
    Costs=[pop.Cost];
    [Costs, SortOrder]=sort(Costs);
    pop=pop(SortOrder);

    % Store Best Solution
    BestSol(j)=pop(1);
    BestSolSoFar(j) = BestSol(j);


    % Store Cost
    WorstCost=pop(end).Cost;
```

```
%% Main Loop
it = 1;
prevsol = 0;
rep = 1;
restart = 1;
hold on
figure();
h = animatedline('Marker','o');
axis([0 50 -inf inf])
xlabel('Iterations')
ylabel('Solution Value')
while BestSol(j).Cost > Tol && it <= MaxIt && restart <= 2
    % Calculate Selection Probabilities
    if UseRouletteWheelSelection
        P=exp(-beta*Costs/WorstCost);
        P=P/sum(P);
    end

    % Crossover
    popc=repmat(empty_individual,nc/2,2);
    for k=1:nc/2

        % Select Parents Indices
        if UseRouletteWheelSelection
            i1=RouletteWheelSelection(P);
            i2=RouletteWheelSelection(P);
        end
        if UseTournamentSelection
            i1=TournamentSelection(pop,TournamentSize);
            i2=TournamentSelection(pop,TournamentSize);
        end
        if UseRandomSelection
            i1=randi([1 nPop]);
            i2=randi([1 nPop]);
        end

        % Select Parents
        p1=pop(i1);
        p2=pop(i2);

        % Perform Crossover
        [popc(k,1).Position, popc(k,2).Position]=...
            Crossover(p1.Position,p2.Position);

        % Evaluate Offsprings
        popc(k,1).Cost=CostFunction(popc(k,1).Position);
        popc(k,2).Cost=CostFunction(popc(k,2).Position);

    end
    popc=popc(:);


    % Mutation
    popm=repmat(empty_individual,nm,1);
    for k=1:nm

        % Select Parent
        i=randi([1 nPop]);
        p=pop(i);
```

```
        % Perform Mutation
        popm(k).Position=Mutate(p.Position,mu);

        % Evaluate Mutant
        popm(k).Cost=CostFunction(popm(k).Position);

    end

    % Create Merged Population
    pop=[pop
        popc
        popm]; %#ok

    % Sort Population
    Costs=[pop.Cost];
    [Costs, SortOrder]=sort(Costs);
    pop=pop(SortOrder);

    % Update Worst Cost
    WorstCost=max(WorstCost,pop(end).Cost);

    % Truncation
    pop=pop(1:nPop);
    Costs=Costs(1:nPop);

    % Store Best Solution Ever Found
    BestSol(j)=pop(1);

    % Store Best Cost Ever Found
    BestCost(it,j)=BestSol(j).Cost;

    % Show Iteration Information
    fprintf('Iteration %d : Best Cost = %.5f\n',it, BestCost(it,j));


    if prevsol == BestSol(j).Cost
        rep = rep + 1;
    else
        rep = 1;
    end

    if BestSol(j).Cost < BestSolSoFar(j).Cost
        BestSolSoFar(j) = BestSol(j);
    end

    if rep > 20
        pop = create_population(nPop, CostFunction);
        restart = restart + 1;
    end

    prevsol = BestSol(j).Cost;
    addpoints(h,it,BestCost(it,j))
    s = sprintf('Current Iteration: %d,', it,...
        'Current Solution Value:%.2f Best Solution Value: %.2f'...
        , BestCost(it,j), BestSolSoFar(j).Cost);
    title(s)
    if it > 50
        axis([0 inf, -inf, inf]);
    end
    drawnow;
```

```
        it= it+1;
    end
end
%% Results

figure;
plot(BestCost,'LineWidth',2);
xlabel('Iteration');
ylabel('Cost');
grid on;
```

Population Generation

```
function [ population ] = create_population( nPop , CostFunction)

empty_individual.Position=[];
empty_individual.Cost=[];

population=repmat(empty_individual,nPop,1);

for i=1:nPop

    counter = 1;
    for z = 0:0
        for k = 0:9
            for j = 0:39
                ran = rand(1,1);
                if ran <=(i-1)/nPop && (i-1)/nPop > 0
                    population(i).Position(counter) = 1;
                else
                    population(i).Position(counter) = 0;
                end
                counter = counter +1;
            end
        end
    end
    population(i).Cost = CostFunction(population(i).Position);

end

end
```

Target Frequency Generation

```
function [ population ] = create_population( nPop , CostFunction)

empty_individual.Position=[];
empty_individual.Cost=[];

population=repmat(empty_individual,nPop,1);

for i=1:nPop

    counter = 1;
    for z = 0:0
        for k = 0:9
            for j = 0:39
                ran = rand(1,1);
```

```
                if ran <=(i-1)/nPop && (i-1)/nPop > 0
                    population(i).Position(counter) = 1;
                else
                    population(i).Position(counter) = 0;
                end
                counter = counter +1;
            end
        end
    end
    population(i).Cost = CostFunction(population(i).Position);

end

end
```

Appendix C

Fabricated Models

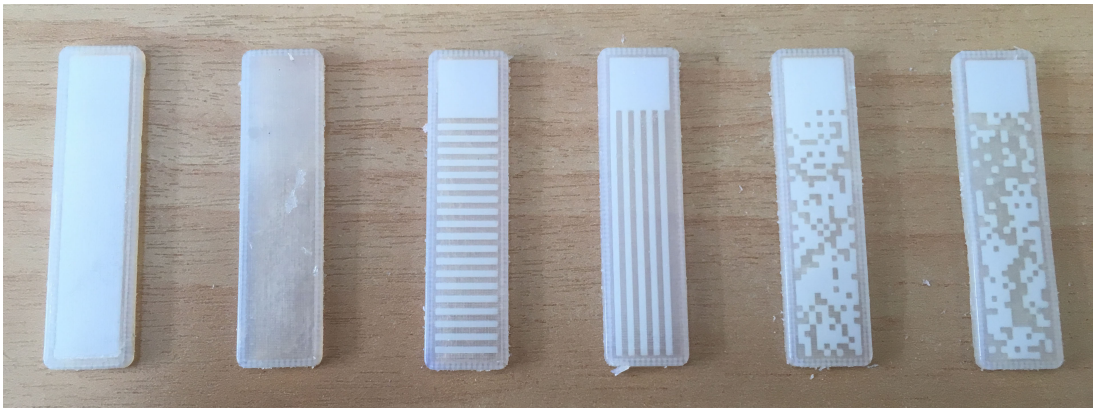


Figure C.1: Image of all fabricated beams.

Figure C.2: Printed models are compared with their corresponding digital models.

References

[1] S. Yang, Y. F. Zhao.
"Additive manufacturing-enabled design theory and methodology: a critical review".
Springer-Verlag London 2015

[2] T. Tomiyama, et al.
"Design methodologies: industrial and educational applications"
CIRP Anna Manuf Techno 58(2):543-565

[3] K. V. Wong and A. Hernandez.
"A Review of Additive Manufacturing".
International Scholarly Research Network, ISRN Mechanical Engineering, Volume 2012, Article ID 208760, 2012.

[4] B. Vayre, F. Vignat, F. Villeneuve.
"Metallic additive manufacturing: State-of-the-art review and prospects".
Mechanics and Industry 13(2):89-96 · January 2012

[5] D. S. Kim, S. W. Bae, K. H. Choi.
"Development of industrial SFF system using dual laser and optimal process".
Robotics and Computer-Integrated Manufacturing, 23(6):659–666 , December 2007

[6] Grimm, Todd.
"User's Guide to Rapid Prototyping".
Society of Manufacturing Engineers, p. 55

[7] LaurensvanLieshout.
"STL-file. Simple description of a STL-file from a donut. The drawing is used to explain the accuracy of a STL file."
7 March 2009. Availability: https://commons.wikimedia.org/wiki/File:STL-file.jpg.

[8] M. Pickton.
"A decent visual representation of the differences, albeit a bit unfair to voxels."
March 06, 2012. Availability: http://media.gamersnexus.net/images/media/2012/features/voxels-vs-vertexes.png.

[9] B. Elliott.
"The 1940 Tacoma Narrows Bridge collapsing in a 42 miles per hour (68 km/h) gust on November 7, 1940."
November 7, 1940. Availability: https://en.wikipedia.org/wiki/File:TacomaNarrowsBridgeCollapse_in_color.jpg.

[10] W Gao, et al.
"The status, challenges, and future of additive manufacturing in engineering".
Computer-Aided Design, 2015

[11] Bianchi, Leonora, Marco Dorigo, Luca Maria Gambardella, Walter J. Gutjahr.
"A survey on metaheuristics for stochastic combinatorial optimization".
Natural Computing: an international journal. 8.2: 239-287. 2009

[12] Whitley, Darrell.
"A genetic algorithm tutorial".
Statistics and computing 4.2: 65-85. 1994

[13] Kennedy, James.
"Particle swarm optimization".
Encyclopedia of machine learning. Springer US, 760-766. 2011

[14] Deb, Kalyanmoy, et al.
"A fast and elitist multiobjective genetic algorithm: NSGA-II".
IEEE transactions on evolutionary computation 6.2: 182-197. 2002

[15] R. M. Badalló.
"Frequencies extraction of a cantilever beam using Finite Element Modelling".
Eduacero. 2(2) 2013

[16] R., J. Narasimha.
"An introduction to the finite element method".
McGraw-Hill New York.3(2.2) 2006

[17] D. J. Inman.
"Engineering vibrations".
Prentice-Hall, 2010.

[18] J. Sun, M. R. Jolly, and M. Norris.
"Passive, adaptive and active tuned vibration absorbers - a survey".
Journal of mechanical design, :117-234, 1995.

[19] C. Q. Howard, C. H. Hansen and A. C. Zander.
"Noise reduction of a rocket payload fairing using tuned vibration absorbers with translational and rotational dofs".
In proc. Australian Acoustical Society, :165 - 171, 2015.

[20] A. de Avila Borges, et al.
"Size and shape optimization of structures by harmony search".
2013

[21] M. B. Du hhring, J. S. Jensen, and O. Sigmund.
"Acoustic design by topology optimization".
Journal of sound and vibration, 317(3):557–575, 2008.

[22] E.L. Doubrovski, E.Y. Tsai, D. Dikovsky, J.M.P. Geraedts, H. Herr, N. Oxman.
"Voxel-based fabrication through material property mapping: A design method for bitmap printing".
Computer-Aided Design 60, 2015.

[23] B. Marshall.
"Automated Fabrication".
Prentice Hall. 1993.

[24] Khan, S. A.
"Voxel-Man Junior Interactive 3D Anatomy and Radiology in Virtual Reality Scenes, Part I: Brain and Skull".
JAMA: The Journal of the American Medical Association. 280 (8): 754–754. 1998.

[25] T. Takahashi, A. Masumori, M. Fuji, H. Tanaka.
"FAV File Format Specification".
July 12, 2016.

[26] N. Cheney, E. Ritz, H. Lipson.
"Vibrational Design and Natural Frequency Tuning of Multi-Material Structures".
GECCO'14, July 12-16, 2014.

[27] Rhino 3D. *"http://www.rhino3d.com/download/rhino/5/latest"*.
2016. Availability: http://www.rhino3d.com/download/rhino/5/latest.

[28] Yarpiz.
"Binary and Real-Coded Genetic Algorithms".
2015. Availability: http://yarpiz.com/23/ypea101-genetic-algorithms.

[29] S. S. Chen.
"Constrained Particle Swarm Optimization".
2016. Availability:
https://au.mathworks.com/matlabcentral/fileexchange/25986-constrained-particle-swarm-optimization.

[30] MathWorks.
"Optimisation Toolbox"
2016. Availability: http://au.mathworks.com/products/optimization/index.html

[31] McNeelEurope.
"GhPython".
Jul 23, 2014. Availability: http://www.food4rhino.com/project/ghpython?ufh

[32] S. Davidson, et al.
"Grasshopper".
2016. Availability: http://www.grasshopper3d.com/

[33] M. Zwierzycki, et al.
"MilkBox".
2016. Availability: http://www.grasshopper3d.com/group/milkbox

[34] MathWorks.
"MATLAB"
2015. Availability: http://au.mathworks.com/products/matlab/

[35] ANSYS.
"ANSYS".
2016. Availability: http://www.ansys.com/

[36] Dassault Systemes.
"ABAQUS".
2016. Availability: http://www.3ds.com/products-services/simulia/portfolio/abaqus/

[37] COMSOL.
"COMSOL Multiphysics".
2016. Availability: https://www.comsol.com/comsol-multiphysics