Topology Optimization of Flat Structures with Adaptive Finite Elements

Diogo Miguel Fael Paraíso
diogo.paraiso@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Portugal

November 2021

Abstract

Topology optimization, as a field of structural optimization, has gained extensive use as a tool in design problems across all fields of engineering and industry alike, however, the growing complexity of the problems being solved is making it more computationally expensive to solve them. Adaptive finite elements or adaptive mesh refinement presents itself as a natural solution to this problem, by reducing the number of finite elements and thus, the computational cost. In this work, the topology optimization of flat structures is approached using an adaptive finite element algorithm. For this purpose, a MATLAB program using quadrilateral 4-node elements was developed to perform a minimum compliance optimization, using the Solid Isotropic Material with Penalization and solving it using the Optimality Criteria method. To better understand the algorithm, a parametric study was conducted where the impact of the adaptive mesh refinement criteria and the topology optimization parameters was tested. The optimum designs using the developed adaptive mesh refinement program were also analyzed and compared against the design obtained on a comparable uniform mesh. The results demonstrate (1) that the adaptive mesh refinement algorithm produced a good mesh behavior and refinement was limited to the necessary regions, (2) it produced adequate results with considerably fewer finite elements, and (3) further filtering techniques need to be studied in order to better utilize the algorithm.

Keywords: Topology Optimization, Adaptive Finite Elements, Adaptive Mesh Refinement, Solid Isotropic Material with Penalization, Non-uniform mesh

1. Introduction

Topology optimization at its core, as one of the basic categories of structural optimization, has always been present in the mind of engineers and thinkers alike. The need to find the optimum design of a structure in order to solve a problem is intemporal. Finding that optimal design, before the advancements in computational aided design (CAD), was always a major challenge and based on previous knowledge and experimentation.

The main goal of this work is developing an algorithm that produces optimum designs with less computational resources, while being easy to use and modify, and relying on one of the programming languages most used in education and research, MATLAB. This is to be done through the implementation of a technique known as adaptive finite elements or adaptive mesh refinement (AMR), which adaptively selects elements to be refined before a new optimization cycle and thus reducing the need of a large number of finite elements when computing the problem.

As the field of topology optimization grows and matures, its use is becoming more common, with numerous applications in both academia and industry alike. The advancements and availability of CAD software made it the natural choice for solving problems such as the one of minimizing weight while maintaining (or increasing) stiffness in structures like the body of a car.

In Aeronautics, for example, the use of topology optimization, often combined with manufacturing restrictions, is increasingly used as a mean to save material, weight and fuel. In recent years, it has been used to achieve an optimal design for an aircraft components, like the seat and seat legs [1]; in the design of morphing wing structures [2, 3] and for the design of the landing gear and engine mount [4].

As the complexity of the problems being solved increases, there is also a growing need of more computational capacity for solving them. An algorithm that produces results comparable to traditional methods, but requiring less computational resources is of particular interest.

Topology optimization as a subject has been researched for many years, dating back to 1988 when was first introduced by Bendsøe [5]. Numerous methods have been developed since then with the use of the homogenization method [5], Solid Isotropic Material with Penalty (SIMP) [6] and more recent ones, such as the moving morphable components method [7].

The use of the SIMP approach, used to solve the 0-1 material problem, was able to overcome the enormous computational requirements needed to solve the continuum problem and was quickly popularized. Not without its problems, such as the presence of intermediate densities and microstuctures, SIMP continues to be one of the most used and crucial methods of solving optimization problems. This is especially true with the use of additive manufacturing and composite materials [8].

To achieve the stated goal, two MATLAB codes for topology optimization [9, 10] were considered and used as a template to build the proposed algorithm. This resulted in an easy to use and modify MATLAB algorithm that uses square 4-node elements for the discretization of the design domain and solves the minimum compliance problem, using SIMP and the optimality criteria method with an AMR cycle after each optimization.

This work is organized as follows. In section 2 the topology optimization problem is formulated and an introduction to AMR is made. The detailed implementation of the MATLAB algorithm follows in section 3. In section 4, to better understand the behaviour of the proposed algorithm to user defined parameters a parametric study was conducted, followed by the validation of the results obtained against a comparable uniform mesh and a brief performance analysis. Lastly, a conclusion is presented in section 5.

The produced algorithm can be found in Appendix in the full dissertation.

2. Background

2.1. Topology Optimization

As previously stated, Topology Optimization is one of the basic categories of Structural Optimization (SO), as such, the general form of the problem is the same and can be simply described as finding the 'best' (optimized) solution to a design problem. This design is mathematically evaluated using an objective function, $f(x, y)$, which measures the performance of said design and represents the aim of the optimization (maximizing or minimizing f). The objective function is dependent on the design variable (x) - representing geometry - and the state variable (y) - representing displacement or force - and the general problem takes the form [11]:

$$
(\mathrm{SO}) \begin{cases} \text{minimize } f(x, y) \text{ with respect to } x \text{ and } y \\ \text{subject to } \begin{cases} \text{behavioral constraints on } y \\ \text{design constraints on } x \\ \text{equilibrium constraint} \end{cases} \end{cases}
$$

The TO problem can be described as a *material distribution problem*, whose solution defines the distribution of material within a defined design domain, and minimizes/maximizes the objective function. Here, this simple problem takes the form of minimizing the strain energy, minimum compliance design, which equates to maximizing global stiffness, i.e., the objective function is the compliance of the design.

Along a discretized domain, using finite elements, the problem takes the form [12]

$$
\begin{aligned} \min_{u,E} \quad & \boldsymbol{u}^T \boldsymbol{K} \boldsymbol{u} \\ \text{subject to :} \quad & \boldsymbol{K}(E_e)\boldsymbol{u} = \boldsymbol{f}, \\ & E_e \in E_{\text{ad}} \end{aligned} \tag{1}
$$

where $\boldsymbol{u}$ are the displacements, $\boldsymbol{f}$ the load vectors and $\boldsymbol{K}$ the stiffness matrix, which depends on E_e, the stiffness tensor for element e, and E_{ad} denotes the admissible stiffness tensors for the design.

For our *material distribution problem* we are interested in determining the optimal placement of isotropic material (or remain void) in our discretized design domain, represented by a finite element mesh. This corresponds to viewing the geometry as a collection of black and white 'pixels' that represent a rough description of the optimal continuum structure. This leads to an integer problem whose solution is computationally heavy, since it results in many design variables/functions, limiting its application on large scale problems [12, 13].

A popular way to address this is to make use of interpolation models for the material properties, such as the SIMP model, which can be written, in his modified state, as [14]:

$$
\begin{aligned} E_e(x_E) = E_{min} + x^p(E_0 - E_{min}), \\ x \in [0, 1],\ p > 1 \end{aligned} \tag{2}
$$

where the density x is the design function, E_0 the material properties of an isotropic material and p the penalization power. The use of a minimum Young's modulus, E_{min}, is essential to avoid possible singularities when solving the equilibrium problem, and in this modified state allows for easier implementation of different filters, such as a density filter.

We are now able to write the entire discretized problem, as is solved by the algorithm, based on the SIMP interpolation where the objective is to minimize compliance[10]

$$\begin{aligned}
\min_{x}: \quad & c(\boldsymbol{x}) = \boldsymbol{U}^T\boldsymbol{K}\boldsymbol{U} = \sum_{e=1}^{N}(x_e)^p \boldsymbol{u}_e^T \boldsymbol{k}_0 \boldsymbol{u}_e \\
\text{subject to}: \quad & \frac{V(\boldsymbol{x})}{V_0} = f \\
& \boldsymbol{K}\boldsymbol{U} = \boldsymbol{F} \\
& \boldsymbol{0} < \boldsymbol{x}_{min} \leq \boldsymbol{x} \leq \boldsymbol{1}
\end{aligned} \tag{3}$$

where $\boldsymbol{U}$ and $\boldsymbol{F}$ are the global displacement and load vectors, respectively, $\boldsymbol{K}$ is the global stiffness matrix, $\boldsymbol{u}_e$ and $\boldsymbol{k}_e$ are the element displacement vectors and stiffness matrix, respectively, $\boldsymbol{x}$ is the design variables vector (to be interpreted as relative density), $\boldsymbol{x}_{min}$ the minimum relative density, N is the number of elements used in the FE mesh, p is the penalization power, $V(\boldsymbol{x})$ and V_0 are the material volume and design domain volume, respectively, and f is the volume fraction (user defined).

2.2. Solution methods

To solve the discretized problem detailed above one must deploy the use of efficient computational methods, since the problem implies a design variable for each discretized element.

An extremely efficient way to solve some optimization problems is the Optimality Criteria (OC) method [12]. This heuristic method relies on finding and selecting designs that fulfill the necessary conditions of optimality, and iteratively, select better and better designs in an attempt to find a global minima (or maxima). This iterative update scheme can be formulated as [9]:

$$x_e^{new} = \\
= \begin{cases} max(0, x_e - m) & \text{if } x_e B_e^{\eta} \leq max(0, x_e - m) \\ min(1, x_e + m) & \text{if } x_e B_e^{\eta} \geq min(1, x_e - m) \\ x_e B_e^{\eta} & otherwise \end{cases} \tag{4}$$

where m is a positive move limit, η $(= 1/2)$ is a numerical damping coefficient and B_e is obtained from the optimality condition as:

$$B_e = \frac{-\frac{\partial c}{\partial x_e}}{\Lambda \frac{\partial V}{\partial x_e}} \tag{5}$$

where Λ is the Lagrange multiplier that can be determined using a bisection algorithm.

To complete the update scheme we must first obtain the derivative of the objective function in respect to the design variable x_e, also called sensitivity of the objective function, which can be written as

$$\frac{\partial c}{\partial x_e} = -p x_e^{p-1}(E_0 - E_{min})\boldsymbol{u}_e^T \boldsymbol{k}_0 \boldsymbol{u}_e \tag{6}$$

The derivative $\frac{\partial V}{\partial x_e}$ is dependent on the size of the element, for unitary elements this assumes the value one, for non-uniform elements, as is the case here, $\frac{\partial V}{\partial x_e} = a_e$, which is the area of the element.

2.3. Difficulties

The computational calculation of the optimization problem requires the implementation of a filtering technique. This is used to ensure the existence of solutions and prevent the appearance of checkerboards patterns and mesh-dependency [12]. Two types of filtering techniques are explored, sensitivity filtering and density filtering.

The use of a sensitivity filter has proven to ensure mesh independence in a highly efficient way, with the added benefit of easy implementation. This is a purely heuristic filter that produces similar results to local gradient constraints based filters. The filter works by modifying the sensitivity as follows [12]:

$$\widehat{\frac{\partial c}{\partial x_e}} = \frac{1}{x_e \sum_{i \in N_e} H_{ei}} \sum_{i \in N_e} H_{ei} x_i \frac{\partial c}{\partial x_i} \tag{7}$$

where N_e is the set of elements i for which the center-to-center distance $\Delta(e, i)$ to element e is smaller than the filter radius r_{min} and H_{ei} is a weight factor defined by $H_{ei} = max(0, r_{min} - \Delta(e, i))$. The modified sensitivities are then used in the optimization scheme (OC method).

The use of a density filter can also solve the problems mentioned above by directly limiting the variation of the densities that appear in the set of admissible stiffness tensor E_{ad} by only admitting filtered densities. Essentially, the filter transforms the original densities x_e into:

$$\tilde{x}_e = \frac{1}{\sum_{i \in N_e} H_{ei}} \sum_{i \in N_e} H_{ei} x_i \tag{8}$$

This new way to refer to the densities, $\tilde{x}_e$, emphasizes the difference between the original densities x_e, which should now be referred to as design variables, and filtered densities $\tilde{x}_e$, which refer to physical densities. Equation 6 remains valid but is now in respect to the physical densities $\tilde{x}_e$. The sensitivity in respect to the design variables x_j is given by

$$\frac{\partial \psi}{\partial x_j} = \sum_{e \in N_j} \frac{\partial \psi}{\partial \tilde{x}_e} \frac{\partial \tilde{x}_e}{\partial x_j} = \sum_{e \in N_j} \frac{1}{\sum_{i \in N_e} H_{ei}} \frac{\partial \psi}{\partial \tilde{x}_e} \tag{9}$$

where ψ is the objective function c or the volume V.

2.4. Adaptive Mesh Refinement

Topology Optimization, as previously mentioned, can be very computationally heavy, since problems are commonly solved using an uniform mesh with a large number of finite elements so as to achieve high accuracy designs. As the TO problem is being solved regions of solid or void material form that don't require such a fine mesh, however, these regions are unknown *a priori*. So it is only natural that adaptive mesh refinement (AMR) be researched as a solution to making solving the TO problem more economical. Thus, the purpose of AMR when applied to TO can be described as a way to obtain accurate results, comparable to those obtained when using a uniform mesh, but through the use of considerably less elements, by refining the mesh when and where necessary. This is not without risk, as the problem of mesh-dependency can be exacerbated by the refinement (and subsequent refinement) of elements.

A number of different approaches have been published on how to integrate AMR in TO to improve computational cost and accuracy. The earliest of these approaches follows a strategy of using the previous optimization to compute the elements to refine followed by optimization of the new mesh [15]. Another approach is to use AMR after each design change in the optimization cycle as proposed by [16].

The integration of an optimize → refine AMR technique, just like the one implemented, on a typical optimization algorithm can be seen in figure 1.

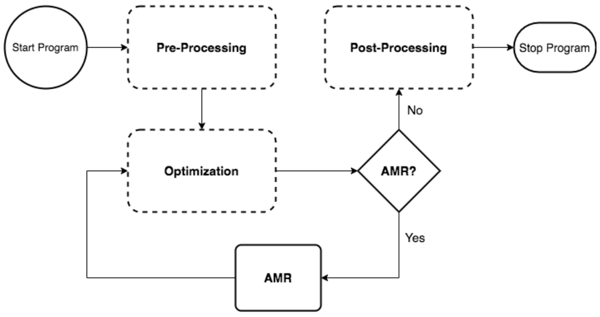

Figure 1: Flowchart of a topology optimization program with AMR, following the optimize → refine technique.

2.5. Theory, difficulties and refinement criterion

In order to implement a basic adaptive mesh refinement some modifications need to be made to the topology optimization program. Without going into computational implementation details, a brief summary of the theory involved is necessary to better understand the algorithm.

The algorithm uses 4-node quadrilateral elements (QUAD4). Refinement is then made by dividing the existing element into four equal elements of the same type, see figure 2 a). By refining an existing element we are creating new nodes and some of those will be classified as 'hanging nodes', this designation is used to identify nodes that have a single-level mesh-incompatibility (again, see figure 2(b)).

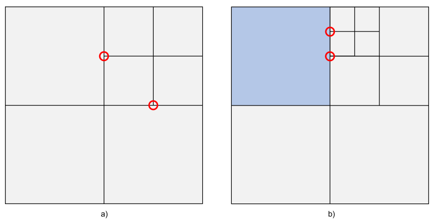

Figure 2: Illustration of the mesh refinement and mesh incompatibilities that arise from it. The refinement represented at a) satisfies single-level mesh-incompatibility, while refinement b) does not. Circles in red indicate 'hanging nodes'. The blue tint means the element is marked for refinement.

In order to fully compute the mesh the following constraint needs to be imposed to these nodes

$$u_i = (\frac{u_j + u_k}{2}) \tag{10}$$

where the subscript i denotes the 'hanging node' and j and k the nodes at the vertices of the same edge. This constrains the displacement of the 'hanging node' to the displacement of its adjacent nodes. In order to auto impose this for element i, the element stiffness matrix needs to be modified with the following:

$$\begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} \\ k_{12} & k_{22} & k_{23} & k_{24} \\ k_{13} & k_{23} & k_{33} & k_{34} \\ k_{14} & k_{24} & k_{34} & k_{44} \end{bmatrix} = \\ = \begin{bmatrix} k_{11} & k_{12} & \frac{k_{13}}{2} & (k_{14} + \frac{k_{13}}{2}) \\ k_{12} & k_{22} & \frac{k_{23}}{2} & (k_{24} + \frac{k_{23}}{2}) \\ \frac{k_{13}}{2} & \frac{k_{23}}{2} & \frac{k_{33}}{4} & (\frac{k_{34}}{2} + \frac{k_{33}}{4}) \\ (k_{14} + \frac{k_{13}}{2}) & (k_{24} + \frac{k_{23}}{2}) & (\frac{k_{34}}{2} + \frac{k_{33}}{4}) & (\frac{k_{33}}{4} + k_{44} + k_{34}) \end{bmatrix} \tag{11}$$

This method for solving the mesh-incompatibility only allows for single-level incompatibilities, for the level-two incompatibilities shown at figure 2 b) (the upper red circle shows a node that would be constrained by an already 'hanging node') the explained method does not work. Since an implementation for a level-two incompatibility is non-practical, a simple way to solve this is to force the refinement of the element adjacent to it (again, see figure 2 b), element in blue) so as to eliminate the level-two incompatibility.

The next step in an AMR is to define the refinement criterion. Several methods have been used, namely, selecting the elements that define the design boundary [17] or selecting the elements that define the design [18]. In a material problem this can be approximated by selecting the intermediate densities, i.e., $\rho_{t^-} \leq \rho_e \leq \rho_{t^+}$ where ρ_e is the density of element e and ρ_{t^-}/ρ_{t^+} represent the lower and upper threshold; or by selecting the solid densities, i.e., $\rho_{t^-} \leq \rho_e \leq 1$. This is the way the developed algorithm selects the elements to be refined.

3. Implementation

The MATLAB code uses the SIMP interpolation method along with the OC method for optimization with a sensitivity or density filter, as previously described.

The program can be called using the prompt:

fem_ite(nelx,nely,rmin,ite_max,filter_type)

where *nelx* and *nely* are the number of elements in the horizontal and vertical directions, respectively, *rmin* is the filter initial size, *ite_max* is the number of optimization cyles that the program will compute and output, and *filter_type* selects the type of filter (0 for sensitivity filter, 1 for density filter) . Other variables, such as volume fraction, which is also the initial (guess) density of each element, and penalization power are defined in the pre-processing section along with other important variables, like the name of the directory where the outputs will be saved, material properties, the refinement criteria bounds (*crt_low* and *crt_high* are ρ_{t^-} and ρ_{t^+}, respectively), the frequency of filter radius change and the boundary conditions (defined at the end of the Refinement/FE stage), all of these can be edited by the user.

The code can be divided into four parts, the initial starting stage (pre-processing), the finite element analysis/refinement, the optimization loop, and the final stage (post-processing). A simplified pseudocode of the developed algorithm, with the stages highlighted in blue can be found in figure 3.

3.1. Pre-processing – Variables and Material Properties (lines 2-64)

In simplified terms, the pre-processing stage houses the initialization of important variables and loop variables used throughout. Some of these include the penalization power (*penal*), volume fraction (*volfrac*), frequency of filter radius change (*div_r*), the lower and upper bounds of the refinement criteria (*crt_low* and *crt_high*, respectively). Other variables, such as an element area matrix and design boundary matrix, central to the functioning of the code, are also created here.

3.2. Finite Element Analysis / Refinement Section (lines 66-478)

After the starting stage the program is completely written within an infinite *while* loop that stops with a *break* command when the number of iterations reaches the number inputted by the user. This is done so the refinement process works in a loop with the optimization cycle, so as to implement the desired 'optimize $\rightarrow$ refine' strategy. A summary of the steps required at this stage can be found at figure 3.

The primary function of this stage is to handle the refinement of the mesh. This includes all the necessary steps to ensure mesh-compatibility (lines 272-323) and impose the constraints for 'hanging nodes' (lines 359-389). The way the algorithm solves the problem of level-two mesh-incompatibility is by refinement the elements around the problematic element, an illustration of this process can be found in figure 4.

3.3. Optimization Section (lines 479-584)

The optimization stage is a standard optimization algorithm modified to ensure compatibility with the AMR algorithm, namely, a non-uniform mesh. These modifications are present throughout and can be seen on lines 482-484, where the filter radius is decreased with every iteration, on line 503, where the program computes the compliance of each element taking into account the different element stiffness matrices, and on lines 528-532 where the filtering function is called. The filtering function had to be completely modified and a way to compute the distance between element centroids had to be implemented (lines 667-723). The filtering in itself is a sensitivity filter or a density filter, depending on user input, and the modification of the sensitivities is present in lines 712-717 and the filter is applied in lines 538-542. A number of different output variables used in the post-processing stage are stored and some verifications, to limit the number of optimization iterations (for example a user defined maximum amount of iterations, *max_opt_ite*), are also implemented.

The optimization stage ends when convergence is achieved, described as when between two consecutive designs the change in design variable is less than 1%, or when the maximum number of optimization iterations is reached.

3.4. Final Stage - Plots and Outputs (lines 589-618)

The final stage of the program is used as a post-processing stage, where a number of plots

Algorithm 1 Simplified AMR algorithm

```
procedure $fem_{ite}(nelx, nely, rmin, ite\_max, filter\_type)$    ▷ program prompt
    Pre-processing stage    ▷ Starting variables and material properties
    while Infinite loop do
        Refinement/FE Stage
        if refine == 'yes' then
            procedure REFINEMENT ALGORITHM
                Numbering of new nodes
                Update node connectivity matrix
                Registry of refinement in 'ledger'
                Update coordinate system
                Update possible node constraints (for single-level mesh incompatibility)
                Update notes at design boundary (if applicable)
                Update DoF connectivity matrix
                Update area matrix
                Detecting and solving two-level mesh incompatibility
                Assign old element density to new elements
                Constraint 'hanging nodes'    ▷ use of node constraints matrix
                Compute $[KE]$ for constrained elements and then $[K]$
            end procedure
        elseInitial Iteration
            Uniform mesh solution
        end if
        procedure OPTIMIZATION ALGORITHM
            TO algorithm with small modifications to for non-uniform mesh
        end procedure
        Post-processing stage    ▷ Plots and Outputs
        if $ite_{max}$ == 'Inputted maximum number of iterations' then
            break
        end if
        AMR Criterion
    end while
end procedure
```

Figure 3: Simplified pseudocode of the developed AMR algorithm.

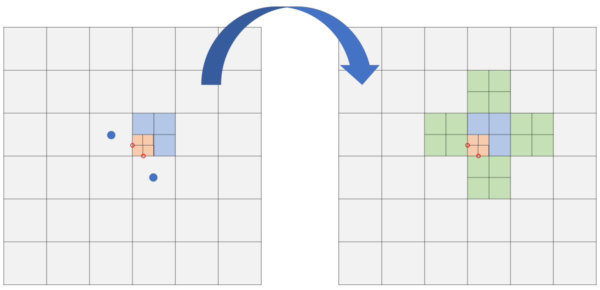

Figure 4: Illustration of how the program solves mesh-incompatibilities. On the left is the problematic mesh, on the right is the solved mesh.

and outputs are stored and displayed, and the place where the AMR criterion is defined and the elements to be refined in the next cycle are selected.

To plot the densities a new function had to be made, it works by extracting the coordinates of each node and filling the corresponding element with its density in grayscale (0 - white and 1 - black).

Finally the algorithm ends with two possible outcomes, if the user specified number of optimization iterations has been reached (line 608) the code *breaks*, leaving the infinite loop and ending the program, if that's not the case the code continues, the elements to be refined are defined using the AMR criterion and the code starts again from the Refinement Section. This last possibility is coded on line 617, which as previously stated selects the new elements to be refined. This is done by selecting the elements that are within specified density bounds. The user can modify the refinement criterion by changing the way the program selects the elements and storing them in the vector, $refinar$. This contributes to the requirement of a straightforward and easy to modify code.

4. Results

To better understand the behavior of the developed algorithm a parametric study was conducted, followed by the validation of the results against those obtained from a comparable uniform mesh. Finally, a brief performance analysis is presented.

4.1. Parametric Study

In order to try to select the best parameters in which to run the algorithm, a parametric study was conducted. For this, the developed algorithm was used to optimize the MBB-beam. The starting mesh consisted of 32×24 elements, and unless that was the parameter being tested, the volume fraction (equal to 50%), penalization power (the usual value of $p = 3$ [10]), filter radius ($r_{min} = 1.5$) and the filter radius change frequency ($div\_r = 2$) will remain constant. The convergence criterion remains the same at less than 1% for consecutive change in design variable. The stiffness of the material ($E_0 = 1$), the stiffness of void regions ($E_{min} = 1e^{-9}$) and the Poisson's ratio ($\nu = 0.3$), remain constant throughout. The number of iterations was kept at 5 to ensure a maximum refinement of four times per element. The design domain for the described problem can be seen in figure 5.

This parametric study consisted in varying the following parameters, refinement criteria (intermediate or solid densities), penalization

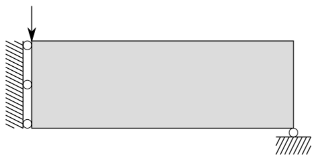

Figure 5: Illustration of the symmetric MBB-beam design domain, boundary conditions, and external load for the topology optimization problem ($C = 35.4125$). Reprinted from [9, p.2].

power, volume fraction, and filtering technique and radius of the filter. The filtering technique used was sensitivity filtering, and the full results can be seen in the full dissertation.

For the refinement criteria study several intervals were tested, ranging from a lower bound of 0.2 to 0.5 with 0.1 increments, to an upper bound of 0.7 to 0.9 with 0.1 increments, and the refinement of 'solid' densities. After analyzing the final design and corresponding final mesh, it is possible to exclude the narrowest intervals and those that exclude the more external densities, since it produced subpar designs with poor detail around the boundary of the material. The corresponding final meshes also presented a poor refinement around the border of the material with areas left unrefined due to the limitation of the interval. The results for 'solid' densities presented a good and detailed final design with the expected final mesh of only material refined, ultimately, this proved to be computationally heavier and the compliance of the design increased with each iteration, which did not occur for the refinement of intermediate densities. It was found that the algorithm worked best for intermediate densities when most densities were selected and the interval of 0.2 to 0.8 was selected, as presented in figure 6 and 7. The analysis of compliance over time shows a decrease in compliance with each iteration (see figure 8, which is the preferred behaviour.

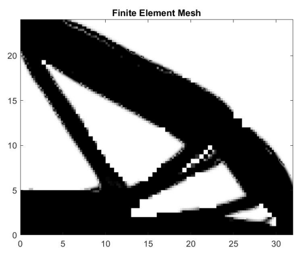

Figure 6: Topology optimized design with AMR for a refinement criterion of $0.2 - 0.8$.

The study also shows a better behaviour when using the typical penalization power of $p = 3$ and

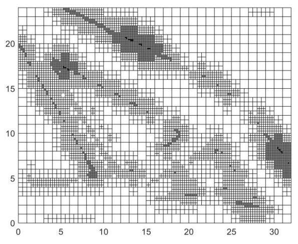

Figure 7: Resulting mesh of the AMR algorithm for a refinement criterion of $0.2 - 0.8$.

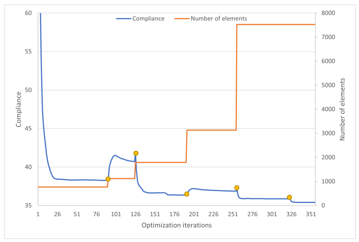

Figure 8: Compliance and number of elements over time (final number of elements, $N_{ele} = 7521$), the yellow dots represents the start of a new cycle. AMR criterion of $0.2 - 0.8$.

using smaller volume fractions, this is because increasing the penalization power reduces the amount of intermediate densities, and for high volume fractions the starting mesh is prone to regions with large amounts of intermediate densities, which causes the algorithm to produce mesh-dependent designs. This is evidence of the need to identify a better filtering technique.

4.2. Filtering technique and radius

With the previous results in mind, several variations to the filtering technique were researched to analyse and improve the behaviour of the sensitivity filter, an analysis of the behaviour of the density filter was also performed. These variations involved changing the radius and implementing a solution where the filtering radius decreases with the iterations.

The results are evident as to the need to use a radius that is better adapted to the adaptive nature of the mesh. When using a constant radius both filtering techniques produced inferior results with holes and a poor definition of the design boundary. This is because the radius is unable to efficiently filter several element sizes with such a high radius (when compared to the final element size).

A natural evolution was to implement a decreasing filter radius, so as to better adjust the filter to the element sizes. To analyse this three filter radius change frequencies were tested, $div\_r = 1, 2$ and 3, where the filter radius was divided by a factor of two every div_r iterations.

The results show that there is a need for fine control over the filtering technique and radius. For $div\_r = 1$ the final design appeared to be adequate and was obtained with a quick convergence, since the use of a small filter radius helped the design to remain more 'black', with less intermediate densities, this produced a worse mesh when compared to a $div\_r = 2$. For $div\_r = 3$ the final design was poor, with microstructures and holes in the design, with a poor boundary detail. Ultimately, a filter radius change frequency of $div\_r = 2$ proved to be more successful at preventing the appearance of holes and microstructures, while also ensuring a good mesh.

The results of the density filter show a similar, if not marginally better, behaviour to the sensitivity filter, but with added computation time. The refinement of intermediate densities produced a good design with good boundary definition, and with a good final mesh. The design obtained for the refinement of solid densities is once again similar to that previously obtained when using sensitivity filtering, with the limitation of being much more computationally heavy.

Ultimately, and since the use of a density filter produced similar results with added computation time, the use of the sensitivity filter is preferred.

There was also an attempt at implementing a better adaptive filter radius, by making it dependent on the size of the element being filtered. This was unsuccessful and the designs presented a border of unconnected gray elements around the design, for both filtering techniques. As a final attempt at producing a better design, the algorithm was modified to perform more filtering cycles, with decreasing filter radius, after the final optimization. The results showed that after optimization, lowering the filtering radius produced some effects, smoothing out the border, but having little impact in the design, especially as the radius continues to decrease, where it has no effect (see figure 9). This produced somewhat of a good result, with compliance decreasing from $C = 35.8774$ at the 5th iteration to $C = 35.4125$ at the 6th, but nothing too significant, nonetheless, one might use it to achieve a better description of the border after the AMR algorithm has finished.

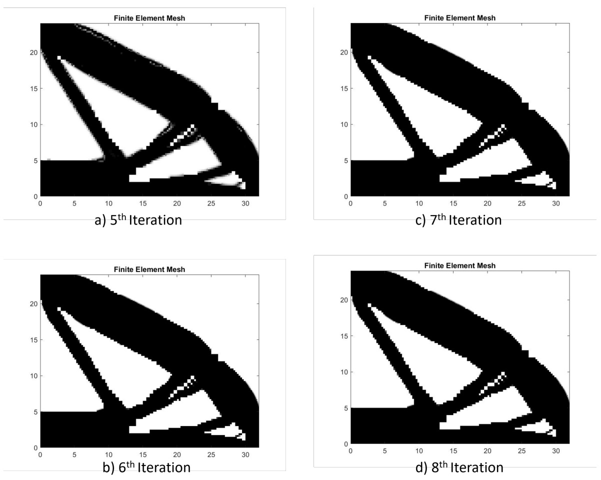

Figure 9: Evolution of the topology optimized design for a decreasing filter radius without refinement. The last iterations are showed with filter radius of a) $r = 0.375$, b) $r = 0.1875$, c) $r = 0.09375$ and d) $r = 0.046875$.

4.3. Validation: MBB-beam

Now that the parametric study identified the better parameters to use in the algorithm, a comparative study was performed to compare the solution obtained using the developed AMR algorithm, against a standard optimization algorithm using a comparable uniform mesh.

The validation of the algorithm consisted of analysing the solutions for four well known problems, the MBB-beam, the cantilever beam, the stocky cantilever beam and the 'wheel'. Only the MBB-beam solution is presented here and one should refer to the full dissertation for further details.

For the MBB-beam problem, knowing that the smallest element using the AMR algorithm after five iterations is ES_f we are able to extrapolate the size of the uniform mesh using the following equation

$$N = \sqrt{\frac{1}{ES_f}} \tag{12}$$

where N represents the factor by which we need to multiply the amount of original elements along x ($nelx$) and y ($nely$).

We are now able to compute N and obtain the size of the comparable uniform mesh, so for the original mesh of 32×24 and $ES_f = \frac{1}{256}$ we get $N = 16$ and thus the uniform mesh is 512×384.

To achieve a fair comparison between algorithms the filter radius also needs to be taken into account, the solution is to use a radius that encompasses an equivalent number of elements on both algorithms. For the MBB-beam problem the final filter radius is $r = 0.1875$, which equates to a filter radius in the uniform mesh of $r = 3$.

Now that all parameters have been established the design on a comparable uniform mesh can be computed, see figure 10.

Firstly, the results demonstrate a certain level of mesh-dependency, since the design is not just a better description of the design found at coarser meshes, but a different, more detailed design. When compared against the design from the AMR algorithm (remember figure 5), and since this

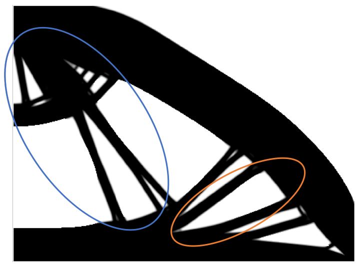

Figure 10: Topology optimized design of a MBB-beam using uniform mesh ($C = 44.9256$), with details

mesh is comparable to that from the AMR algorithm, this can be interpreted as further evidence that the filtering technique was insufficient for some parts of the design.

Nonetheless, the results show similarities, on the lower right corner of the designs there is the formation of a 'beam' that is present in both, the middle 'beam' also splits in two in both designs and the bottom decreases its height in both. The main difference is on the top left corner and along the small 'beam' that originates in it (see detail in blue in figure 10), on the AMR algorithm the 'beam' remains single and the corner doesn't bulge out. This might be because of the phenomena previously described that an AMR algorithm that follows the optimize $\rightarrow$ refine strategy can 'lock' in local minima from the first iterations. The space in the detail in orange, on the same figure, is also present in the design from the AMR algorithm.

Analyzing now the compliance over time, see that it decreased (see figure 8) with the iterations, sometimes increasing as the cycles progress, this is because of refinement creating elements that are initially 'populated' with intermediate densities, being especially present from the 1^{st} to the 2^{nd} iteration. The number of elements over time, as it should, increased over time, except for the last cycle, as this was exclusive for decreasing the filter radius. Over time the number of elements increased from the initial 768 to the final 7521, but significantly less when compared to the uniform mesh's 196608.

Overall, the results from the AMR algorithm were good with compliance decreasing from $C = 44.9256$ to $C = 35.4125$, for the uniform mesh and AMR algorithm, respectively, with the AMR algorithm producing a comparable and adequate design while using far less elements.

4.4. Performance

To analyze the performance of the AMR algorithm the time spent computing the problems used for validation was used. This was done using the MATLAB function, cputime, which retrieves the CPU time used by MATLAB. The code used for comparison is the '88 lines code'[9], and both algorithms ran on MATLAB R2020b on an AMD 3900X CPU with 128GB of RAM. The results are shown in table 1, which presents the average CPU time spent by MATLAB over the optimization iterations.

The results show that the optimization cycles are similar or shorter in the uniform mesh program, even though not the desired outcome, this is because of the highly optimized code used for comparison, since the less optimized '99 lines code' [10] would take a prohibitively amount of time to compute (at a cputime = 25 min per iteration), this reveals the potential of an AMR algorithm.

More interesting, is the fact that the whole program, except for the 'wheel' problem, is always faster than that of the uniform mesh. This could be because of two things, first, the AMR algorithm required less iterations overall to solve the same problem, leading to less computing time, and second, the AMR algorithm, with all its inefficiencies, presents itself as a way to achieve comparable results using less computation time.

Table 1: Table of performance comparison between the AMR algorithm and the '88 lines code' [9]. Note: the two times in the AMR algorithm is for with and without extra filtering.

	AMR Algorithm		'88 lines code'	
	Optimization iteration (s)	Whole program (min)	Optimization iteration (s)	Whole program (min)
MBB-beam	7.6 - 9.48	22.45	9.78	71.59
Cantilever beam	8.86 - 9.64	31.35	8.03	65.37
Stocky cantilever beam	11.26 - 15.60	42.99	7.8	53.34
'Wheel'	16.44 - 17.13	95.61	7.98	50.07

5. Conclusion

The main purpose of this work was to build an easy to use and modify MATLAB code for topology optimization using adaptive finite elements, which has been successful.

Nevertheless, a number of shortcomings have been identified that need to be addressed in order to use the algorithm at its full potential.

Based on the results previously presented the main limitation is related to the filtering technique. The non-uniform nature of the mesh makes the classic sensitivity filter inadequate, even with the filter radius decreasing with every iteration. The density filter behaved much better, but came with heavier computational costs.

The refinement criteria proved to be effective at selecting the intended elements and the algorithm produced a good final mesh.

Despite the general use of for loops, the developed algorithm showed that for most problems, it requires less computational resources than when compared with a traditional program on a uniform mesh.

It is also important to acknowledge that the designs obtained, even though crude in some ways, always produced a good description of an optimum design.

For future work, a better and more adequate filter needs to be researched and implemented. Furthermore, improving the computational efficiency of the algorithm is necessary, since the general use of for loops comes at the expense of compute time. The implementation of derefinement would allow further savings in compute time by reducing the number of elements in void regions. Lastly, the use of a MATLAB function to calculate distance between all points results in high memory usage, a different method, to allow for larger problems, should be investigated.

References

[1] Stephen W. Roper, Haksung Lee, Mongyoung Huh, and Il Yong Kim. Simultaneous isotropic and anisotropic multi-material topology optimization for conceptual-level design of aerospace components. *Structural and Multidisciplinary Optimization*, 64(1):441–456, 2021.

[2] F Sousa, F Lau, and A Suleman. Topology optimization of a wing structure. *Engineering Optimization 2014*, page 507–512, 2014.

[3] Peter Dørffler Jensen, Fengwen Wang, Ignazio Dimino, and Ole Sigmund. Topology optimization of large-scale 3d morphing wing structures. *Actuators*, 10(9):217, 2021.

[4] David J. Munk, Douglass J. Auld, Grant P. Steven, and Gareth A. Vio. On the benefits of applying topology optimization to structural design of aircraft components. *Structural and Multidisciplinary Optimization*, 60(3):1245–1266, 2019.

[5] Martin Philip Bendsøe and Noboru Kikuchi. Generating optimal topologies in structural design using a homogenization method. *Computer Methods in Applied Mechanics and Engineering*, 71(2):197–224, 1988.

[6] M. P. Bendsøe and O. Sigmund. Material interpolation schemes in topology optimization. *Archive of Applied Mechanics (Ingenieur Archiv)*, 69(9-10):635–654, 1999.

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

[8] Jikai Liu and Yongsheng Ma. A survey of manufacturing oriented topology optimization methods. *Advances in Engineering Software*, 100:161–175, 2016.

[9] Erik Andreassen, Anders Clausen, Mattias Schevenels, Boyan S. Lazarov, and Ole Sigmund. Efficient topology optimization in matlab using 88 lines of code. *Structural and Multidisciplinary Optimization*, 43(1):1–16, 2011.

[10] Ole Sigmund. A 99 line topology optimization code written in matlab. *Structural and Multidisciplinary Optimization*, 21(2):120–127, 2001.

[11] Peter W. Christensen and Anders Klarbring. *An introduction to structural optimization.* Springer, 2009.

[12] Martin P. Bendsøe and O. Sigmund. *Topology Optimization: Theory, methods and applications.* Springer, 2003.

[13] Yuan Liang and Gengdong Cheng. Topology optimization via sequential integer programming and canonical relaxation algorithm. *Computer Methods in Applied Mechanics and Engineering*, 348:64–96, 2019.

[14] Ole Sigmund. Morphology-based black and white filters for topology optimization. *Structural and Multidisciplinary Optimization*, 33(4-5):401–424, 2007.

[15] K. Maute and E. Ramm. Adaptive topology optimization. *Structural Optimization*, 10(2):100–112, 1995.

[16] Miguel A. Salazar de Troya and Daniel A. Tortorelli. Adaptive mesh refinement in stress-constrained topology optimization. *Structural and Multidisciplinary Optimization*, 58(6):2369–2386, 2018.

[17] Shanglong Zhang, Arun L. Gain, and Julián A. Norato. Adaptive mesh refinement for topology optimization with discrete geometric components. *Computer Methods in Applied Mechanics and Engineering*, 364:112930, 2020.

[18] Shun Wang, Eric de Sturler, and Glaucio H. Paulino. Dynamic adaptive mesh refinement for topology optimization. *ArXiv*, abs/1009.4975, 2010.