Accelerating topology optimization codes using mesh refinement continuation

by

Austin Chen

B.S. in Civil Engineering
University of California, Berkeley, 2022

Submitted to the Department of Civil and Environmental Engineering in partial fulfillment of the requirements for the degree of

Master of Engineering in Civil and Environmental Engineering

at the

Massachusetts Institute of Technology

June 2023

Authored by:	Austin Chen
	Civil and Environmental Engineering
	May 12, 2023
Certified by:	Josephine V. Carstensen
	Assistant Professor of Civil and Environmental Engineering
	Thesis Supervisor
Accepted by:	Colette L. Heald
	Professor of Civil and Environmental Engineering
	Chair, Graduate Program Committee

Accelerating topology optimization codes using mesh refinement continuation

by

Austin Chen

Submitted to the Department of Civil and Environmental Engineering on May 12, 2023
in partial fulfillment of the requirements for the degree of Master of Engineering in Civil and Environmental Engineering

Abstract

A new concept for an algorithm accelerating topology optimization programs is introduced and explained in detail, which involves a continuation of increasing mesh resolutions to achieve low-compliance solutions with largely reduced computation times. Comparisons with examples from relevant literature show speedups of up to approximately 60% on discretizations up to the order of 10^6 elements for common benchmark problems. Improvements in speed can be attributed to taking advantage of running code on coarse meshes as a faster way to generate smart initial guesses to be reused as inputs for subsequent runs on finer meshes. A MATLAB script for the new algorithm and associated modifications to existing topology optimization code is included.

Keywords: Topology Optimization, Mesh Refinement, Computational Efficiency, MATLAB

Thesis Supervisor: Josephine V. Carstensen
Title: Assistant Professor of Civil and Environmental Engineering

Acknowledgments

First and foremost, I would like to express huge gratitude to my teacher, supervisor, and advisor, Professor Josephine V. Carstensen, for bringing me into the program at MIT, for recognizing and allowing me to grow academically as a member of her research team (the Carstensen Group), and for patiently guiding me throughout this rigorous year. She has reignited my passion for structural engineering and programming, and I owe a large part of my academic success to her education and constant support.

I would also like to recognize the bright minds of the Carstensen Group for opening my eyes to the fascinating applications and potential of topology optimization. Thank you to Dat and Jackson for providing me with the resources I needed to perform meaningful research, Zane for his pro bono tutoring in my early stages of learning topology optimization, and Hajin, Gillian, and all of the team for their valuable feedback and inspiring work.

Next, I am grateful to Dr. Vittoria Laghi, Dr. Caitlin Mueller, Dr. Tal Cohen, and the rest of the faculty and staff within the Department of Civil and Environmental Engineering at MIT for the highest quality of education, guidance, connections, and snacks.

Thank you to my fellow M.Eng. cohort for all the fun times and for endlessly supporting me, not only along my academic journey as amazing classmates but also in my other endeavors as amazing friends.

And finally, a thank you to my mom and dad, whom I hope I have made proud.

Table of Contents

1 Introduction

1.1 Background and Motivation

Topology optimization is a complex process often used in structural design to determine where it is most efficient within a design space to place material for the best performance. This method of design allows designers to create strong, and often beautiful, structures while maintaining a low volume of material, reducing environmental impacts and costs. Generally, a design domain is processed through an optimization algorithm that continuously moves material around until its stiffness is maximized. Though topology optimization may seem like the perfect solution for structural design, its advantages are offset by many challenges. Each added detail to ensure the physical feasibility of optimized results abrades the ideal simplicity of the topology optimization problem. Considerations such as adjusting allowable angles or member sizes for manufacturability or accounting for second-order effects as they would affect real structures all astronomically complicate the process, oftentimes pushing existing technology to its very limit.

The classic problem formulation of structural topology optimization in its simplest form requires minimizing the compliance c (inverse of stiffness) of a design given a volume and equilibrium constraint. This formulation is mathematically understood as the following set of equations (1.1), reproduced from Andreassen et al. (2011):

$$\begin{aligned}
\min_{\mathbf{x}}: \quad & c(\mathbf{x}) = \mathbf{U}^{\mathrm{T}}\mathbf{K}\mathbf{U} = \sum_{e=1}^{N} E_e(x_e)\,\mathbf{u}_e^{\mathrm{T}}\mathbf{k}_0\mathbf{u}_e \\
\text{subject to}: \quad & V(\mathbf{x})/V_0 = f \\
& \mathbf{K}\mathbf{U} = \mathbf{F} \\
& \mathbf{0} \leq \mathbf{x} \leq \mathbf{1} \\
\text{where}: \quad & E_e(x_e) = E_{\min} + x_e^p (E_0 - E_{\min})
\end{aligned} \tag{1.1}$$

where **U** and **F** are the global displacement and force vectors; **K** is the global stiffness matrix; V measures the total volume of material, which is divided by a constant design domain volume V_0 to match a prescribed volume fraction f; and **x** is the set of N design variables, which are each constrained to a density between 0 and 1. To reduce intermediate densities and encourage values of **x** close to 0 and 1, the SIMP (Solid Isotropic Material with Penalization) approach is often incorporated into the problem formulation by means of an exponential factor p applied to the density of each element e, where E_0 is the stiffness of the material and $E_{\min}$ is a lower bound on stiffness to prevent singularity in the stiffness matrix. The resulting stiffness E_e of each element is multiplied by the element displacement vector $\mathbf{u}_e$ and element stiffness matrix $\mathbf{k}_0$.

This type of finite element analysis is usually used iteratively within computer programs to achieve an optimal design. Unfortunately, finite element analysis is a highly computationally expensive procedure with non-linear complexity. As the number of elements increases, the required number of calculations increases at an even faster rate. For simple problems, existing topology optimization programs perform adequately; however, research today progresses at a rate where adequacy no longer suffices. With each added parameter over time, applied to increasingly finer discretizations, the required computational time and power often exceed the available resources of the average researcher. To increase the accessibility of topology optimization research, the tools used must constantly be improved.

At the core of all optimization problems is the search for the highest-performing solution at the lowest expense. When comparing the performance of different optimization algorithms, a common consideration is the degree of efficiency (Sigmund and Maute 2013). As research continues to increase in complexity, efforts have also been dedicated to maximizing the speed of topology optimization scripts.

One of the most commonly used programs today is `top88`, an 88-line MATLAB script published by Andreassen et al. in 2011. `top88` follows the original 99-line `top` code published by Sigmund in 2001. As stated in the 2011 paper, one of the main motivations for the publication of the 88-line code was the recognition of several flaws that limit the original 99-line code from reaching its full potential in speed. This was one of the earliest steps towards improving the efficiency of topology optimization programs, with which Andreassen et al. reported a total computation time reduction by a striking factor of 100 for a given example problem involving 7,500 elements. Nine years later, a new publication by Ferrari and Sigmund (2020), along with a successor program `top99neo`, was released which again focused on improving the efficiency of its predecessor `top88`. `top99neo` was reported to further improve speeds up to 5.5 times for discretizations up to 4.8×10^5 elements. It is clear that the push toward creating a perfect computer program to enable analysis for increasingly complex problems is ongoing and highly valuable.

1.2 Concept

Both `top88`'s improvements upon `top` and `top99neo`'s improvement upon `top88` involve a reworking of the internal algorithms. While there is an evidently drastic improvement from each script to the next, the relative speedups seem to be reaching a limit. From a positive point of view, this implies that current codes are close to maximizing computation speeds. On the other hand, further improvements will become increasingly difficult, which may require more innovative ideas for improving efficiency. In this study, a new concept for reducing computation times is explored, involving an algorithm that performs as a wrapper function, as opposed to a reworking of the internal algorithms of any existing codes. This feature separates this concept from previous scripts by essentially functioning as an extension to any given base code to improve speeds extrinsically.

The inspiration for the new algorithm is a combination of two ideas. The first assumes that most methods of optimization, especially for nonlinear problems, are sensitive to an initial guess (Wicklin 2014). Essentially, a topology optimization process can converge or terminate within fewer iterations if the initial guess resembles the output solution. This assumption effectively ensures that close proximity to the solution would provide a head-start to the progression of the optimizer. To illustrate this claim, the path of an optimizer for a hypothetical single-variable problem can be considered with different initial guesses (Fig. 1.2-1). Especially with a fixed step size, the optimizer evidently must take many more steps to reach the same solution if the initial guess is far away from the global minimum. Additionally, for problems such as in the illustration where multiple local minima exist, a distant initial guess may even run the risk of leading the optimizer to an undesired minimum.

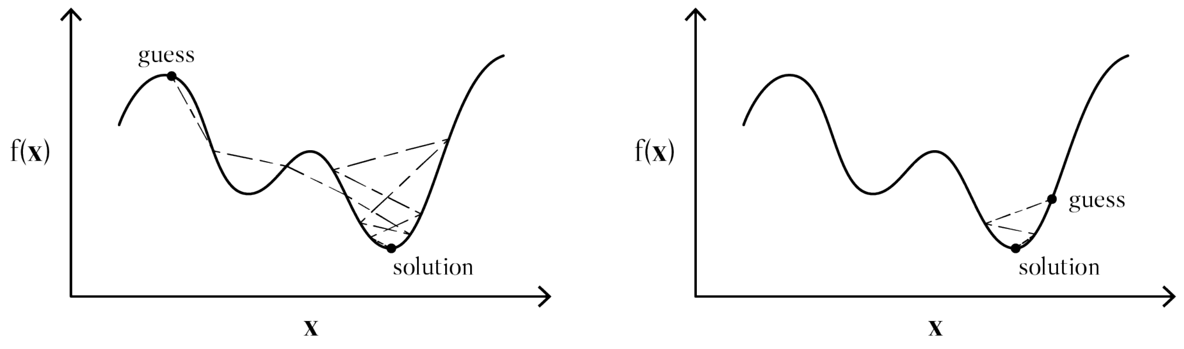

Fig. 1.2-1 Illustrating the effect of the proximity of the initial guess

Classically with existing algorithms, a matrix consisting of a uniform distribution of the prescribed volume fraction is chosen as the arbitrary initial guess. This practice may be adequate for most cases, as it is a convenient way to initiate the loop independent of any other variables. However, this "blind" guess may quickly bring consequences with increasing element matrix sizes. Assuming a prescribed volume fraction f and a desired output of only 0s and 1s, each of n elements must on average move a distance of:

$$\frac{fn(1-f)+(1-f)nf}{n} = 2f(1-f) \tag{1.2}$$

If each element is initiated at a point much closer to its endpoint, the number of steps to reach a final value should decrease, shortening the optimization loop. There are two approaches to solving the problem of initial guess dependency: either the algorithm can be altered to become independent of the initial guess (Wang et al. 2007), or the process for finding the initial guess itself can be refined; the new algorithm discussed in this study focuses on the latter.

The second assumption for the new algorithm references the observation that computation speeds are much higher at lower resolutions. The resolution of a mesh, for the purposes of this paper, refers to a multiplier to an arbitrary set of dimensions (number of elements in each direction). The examples used in this study are extracted from existing literature, in which certain dimensions are used for demonstration; the mesh discretizations of these past examples will be defined to have a resolution of 1 to normalize them as a point of reference. For example, an original problem from Andreassen's study was run with a mesh of 60×20 elements; these dimensions are thus considered to have a resolution of 1. A subsequent run on a 120×40 mesh would thus be defined as a resolution of 2 (Fig. 1.2-2).

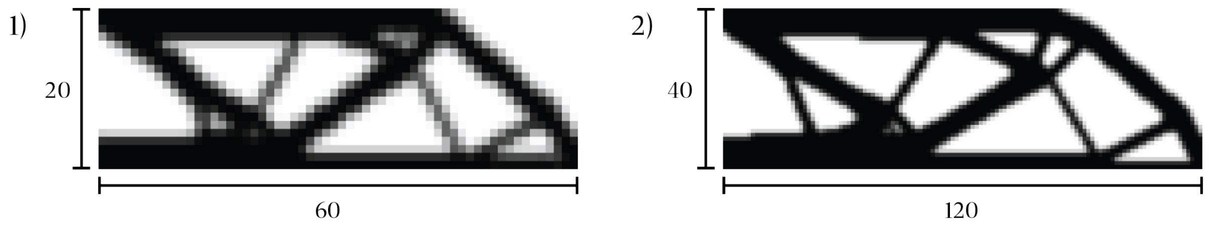

Fig. 1.2-2 (1) A solution using the original 60×20 mesh (resolution = 1), and (2) the same solution using a refined 120×40 mesh (resolution = 2)

While it is obvious that increasing the total number of elements should increase computation time, especially for such a complex calculation like a finite element analysis, a simple comparison of computation times across different mesh resolutions for a commonly tested problem can easily illustrate the validity of the claim.

The relationship demonstrated in this plot demonstrates that solutions are obtained in much less time on coarser meshes and computation times seem to increase exponentially with increasing resolution (Fig. 1.2-3, top). Additionally, normalizing each runtime to the number of required iterations shows that each iteration also grows exponentially in computation time with increasing mesh resolution (Fig. 1.2-3, bottom). From these two observations, it is clear that higher speeds at lower resolutions can be taken advantage of.

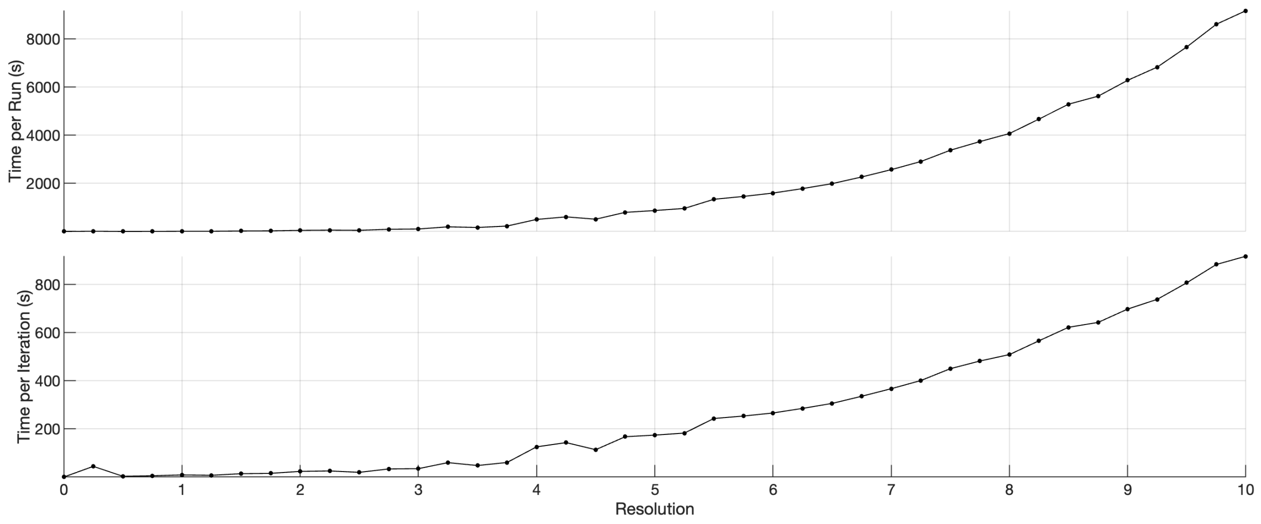

Fig. 1.2-3 Total computation time vs. resolution plot (top) and computation time per iteration vs. resolution plot (bottom) for a classic topology optimization problem

2 Method

2.1 Algorithm Overview

A simple algorithm that takes advantage of the two previous ideas can thus be developed as follows (illustrated in Fig. 2.1-1):

1. Begin with a coarse mesh
2. Obtain the solution on the coarse mesh
3. Refine the mesh
4. Project the previous solution onto the finer mesh
5. Reevaluate the solution using the projected solution as the initial guess
6. Repeat steps 3-5 until a target mesh resolution is achieved

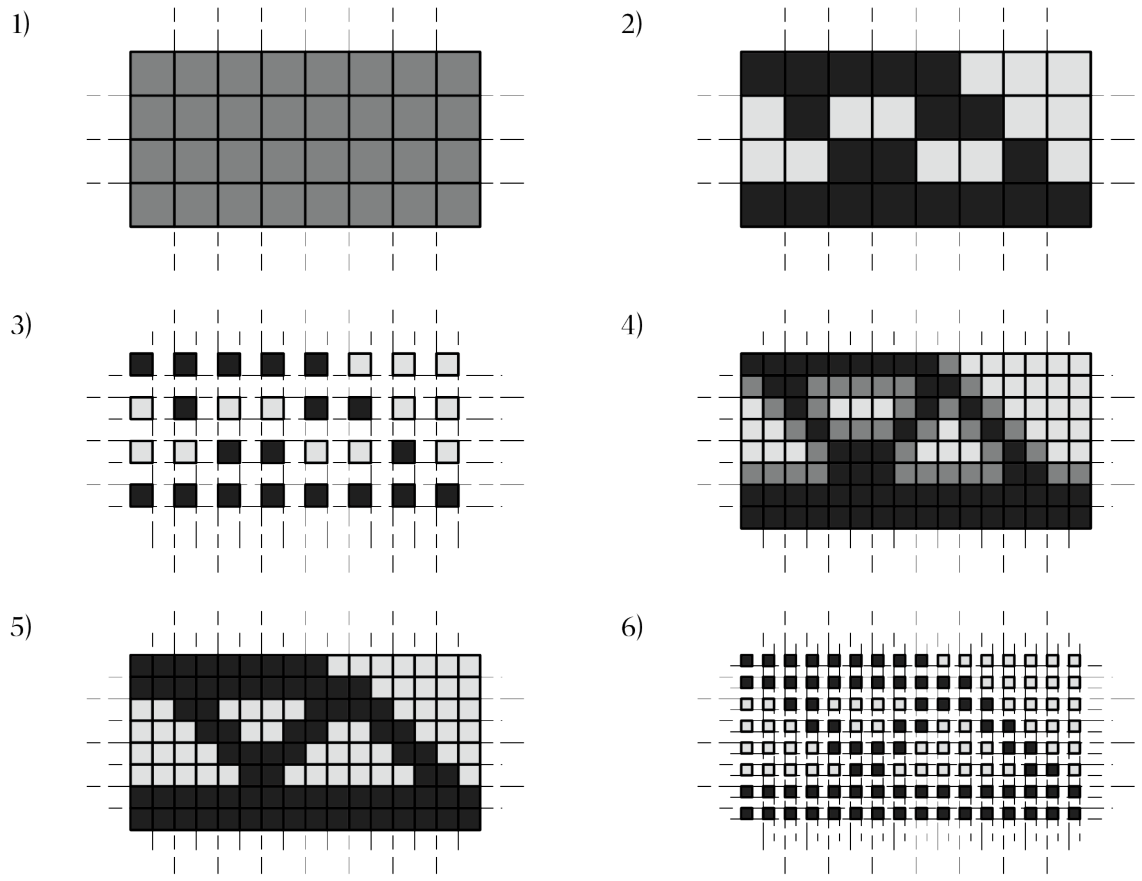

Fig. 2.1-1 Illustrated example of the proposed new algorithm

The above illustration is a simplified and idealized version of how the algorithm performs on a classic topology optimization problem. In step 1, the mesh is initialized with a matrix per usual with a specified uniform volume fraction across all elements. The matrix undergoes one run of a topology optimization algorithm, such as `top88`, to produce the result depicted in step 2, where each element roughly diverges into either a 0 or 1 density. In step 3, the mesh is refined, splitting each original element into smaller elements in each direction; the illustration depicts each element splitting into two in each direction, creating roughly 4 times the number of elements as the previous mesh. The densities from step 2 are preserved and stored in the upper-left corner of the refined mesh, leaving the other new elements empty to be filled in. Step 4 shows a linear interpolation scheme for populating the empty elements, creating a "blurred" image of the solution from step 2, and reintroducing intermediate densities. The matrix created from interpolation is used as the input for the next optimization run, producing the new solution shown in step 5 that once again filters intermediate values towards 0 and 1 densities. Step 6 shows the next mesh refinement, repeating steps 3-5 until a target mesh resolution is reached.

The significance of this algorithm mainly lies in step 5 as mentioned above. In theory, the fine-mesh projection of the solution from a coarse-mesh run should be a highly improved initial guess compared to a matrix of uniform values. Combined with the assumption that solutions can be obtained much faster on coarse meshes, the system of reusing outputs as inputs in an iterative manner over increasing resolutions can significantly reduce the total number of calculations required to achieve a solution at a target resolution.

2.2 Definitions and Details

For consistent terminology, this paper will distinguish between an *iteration* and a *run*. An *iteration* will refer to a single loop within a topology optimization algorithm like `top88`; with each *iteration*, the density matrix is updated once. A *run* will then refer to a sequence of multiple *iterations*; a complete *run* allows the optimizer to progress through multiple *iterations* until a target change or the maximum *iteration* limit is reached.

As the new algorithm centers around a mesh refinement continuation scheme, it will be labeled as `MRC` to reduce verbosity. `MRC` is divided into three stages: the *initial*, *intermediate*, and *final* stages. The *initial* stage serves to obtain an initial solution at the coarsest resolution defined; there is one *initial run*. The *final* stage serves to find the final solution at the target resolution defined; the *final run* occurs once at the end of `MRC`. The *intermediate* stage consists of *runs* where solutions are not significant but rather only serve as checkpoints on the way to the *final* stage. The number of *intermediate runs* can vary, including 0, depending on the initial and final (target) resolution.

While the overall algorithm is simple in nature, there are a few complexities that must be addressed to remove the arbitrariness of some aspects of the algorithm. These are studied using placeholder parameters P_i that are assigned numerical values following a parameter sweep. The first complexity is determining where to start; it is important to define a robust method of determining the appropriate initial resolution. To account for all possible sets of input parameters, the initial resolution should in some way inversely relate to the dimensions of the target mesh (X and Y) such that if a high target resolution is desired, more intermediate runs will happen to take advantage of computation speeds on coarser meshes. There must also be a limit imposed on the initial resolution so as to avoid unreasonably coarse resolutions during the progression of the algorithm. A proposed calculation of the initial resolution res_0 is as follows in Equation (2.2-1). Variables are labeled both originally and in accordance with conventions within `top88`; for more details on the specific variables and their definitions, see Andreassen et al. (2011).

$$\begin{gathered} res_0 = \max\left(\frac{P_1}{\min(X,Y)}, P_2\right) \\ \text{where } X = \frac{nelx}{res_k} \text{ and } Y = \frac{nely}{res_k} \end{gathered} \tag{2.2-1}$$

In this calculation, P_2 is the lower limit on available initial resolutions, while P_1 represents a function for determining the initial resolution based on a limiting dimension.

The next complexities must be addressed to avoid early convergence to suboptimal solutions during initial and intermediate runs. If initial and intermediate runs are allowed to run to completion, the optimizer may risk converging down a local minimum on a coarse mesh that is far from a local minimum on the desired mesh. To prevent this potential risk, a continuation scheme must be adopted in regard to the number of iterations per run. This experiment uses the following continuation scheme in Equation (2.2-2), allowing P_3 iterations in the initial run, followed by increasing the maximum number of iterations by P_4 from the previous run, then allowing the final run to continue to completion to the prescribed maximum number of total iterations.

$$\begin{array}{ll} \text{Initial run:} & maxit_0 = P_3 \\ \text{Intermediate runs:} & maxit_{k+1} = maxit_k + P_4 \\ \text{Final run:} & maxit_n = maxit \end{array} \tag{2.2-2}$$

The final complexity accounts for potential errors that may arise during the projection step of each run. A proposed method of reducing the impact of such errors while remaining independent of the projection scheme is by multiplying the physical filter radius r_{min} (scaled by the resolution) by a factor P_5 during the initial and intermediate runs to effectively "relax" the mesh. During the final run, the filter radius should return to the original r_{min} to produce similar results as the original algorithm. The blurriness during the initial and intermediate runs, in conjunction with the iteration limits, should allow for some semblance of a solution to be obtained at the end of each run; however, the solution should remain unclear enough to provide the optimizer more freedom on subsequent runs.

$$\begin{array}{ll} \text{Initial/intermediate runs:} & r_{\text{min}} = P_5 \cdot res_k \cdot r_{\text{min, phys}} \\ \text{Final run:} & r_{\text{min}} = res_k \cdot r_{\text{min, phys}} \end{array} \tag{2.2-3}$$

While many other complexities also exist, the five aforementioned parameters are key values in creating an optimized version of `MRC`. The remaining major complexity is the choice of a projection scheme; during initial testing of `MRC`, it was discovered that the only scheme that runs without failure is splitting the elements of a coarse mesh into two in both directions and linearly interpolating between existing values to fill in the missing elements. Other schemes, including tripling the number of elements in each direction or using cubic interpolation, almost always resulted in failure to converge. For the five key parameters, a crude parameter sweep was performed to determine a suitable combination of values. The final set of values is set as P_1=4, P_2=1/8, P_3=10, P_4=5, and P_5=2.5. More details on how these values are integrated into the `MRC` code, as well as a step-by-step documentation of the full algorithm, are included in the full script attached in the Appendix.

2.3 Base Code Modifications

To enable the iterative nature of `MRC`, a few minor modifications must be made to any existing topology optimization codes. First, the solution matrix (typically assigned as the variable *xPhys*) must be added as an output to the function. Second, the initial design variable matrix (typically assigned as the variable *x*) must be added as an input to the function. Finally, for codes like `top88` that do not have an existing iteration limit, there must be another added input for breaking the main loop at a desired iteration count. The resulting edits to `top88` as an example can also be found in the Appendix. As other relevant codes like `top99neo` and `top3D125` were developed from `top88`, the modifications made to those are very similar and are omitted for redundancy.

2.4 Testing Procedure

For all experiments conducted in this study, the original and new algorithms are tested in parallel on MATLAB instances on remote computers via Open OnDemand for maximal consistency in processing speed. Each problem is run over a range of resolutions in series, usually beginning at 0.25 and incrementing by 0.25, as well as in batches to minimize fluctuations across different sessions. The time limit for any single set of runs is capped at 12 hours when using this tool, thus data was not attainable for certain ranges of resolutions.

The internal timer in MATLAB tracks the total amount of time required for each run, and the final compliance, iteration count, and solution are obtained directly from the base topology optimization code being tested. To compare the performance of the original and new algorithms, the two sets are plotted against mesh resolution on the same graph for each metric.

3 Results

Problems in this study are evaluated based on three metrics: final compliance, total number of iterations, and total computation time. The main focus is on the computation time, though the iteration count gives additional insight into the progression of the optimization. Compliance, while it is the objective function of all problems, is not the focus of this study; however, it is still considered to ensure that `MRC` produces reasonable results compared to results using the original codes.

The main codes referenced in detail in this study are `top88` by Andreassen et al. (2011) and `top-99neo` by Ferrari and Sigmund (2020). A brief additional exploration of `top3D125` (Ferrari and Sigmund 2020) is also included to demonstrate the applicability of the algorithm to a more recent and specialized code. As `MRC` is a wrapper function, it is essentially applied to a base code; in terms of nomenclature, the name of the base code will be appended. For example, `MRC` applied to `top88` will be referred to as `MRC(top88)`.

3.1 Using `top88`

Initial tests were performed using the 88-line MATLAB code by Andreassen et al. due to its popularity and relevance in the community. As it is often used or extended upon, the 88-line script is an appropriate starting point for proving the concept of the new algorithm. `MRC` was tested on the two types of problems explored in the 2011 study, the MBB beam and the short cantilever.

3.1.1 MBB Beam

The classic MBB beam problem is defined with simply-supported connections and a single point load at mid-span, often simplified to half the geometry due to symmetry. Frequently used as a benchmark problem in topology optimization codes, the MBB beam is an appropriate initial test case for `MRC`. The dimensions, as shown in Fig. 3.1.1-1, are taken from literature as the reference data point (when the resolution is 1). After testing this problem up to a resolution of 10, the following data were collected and plotted.

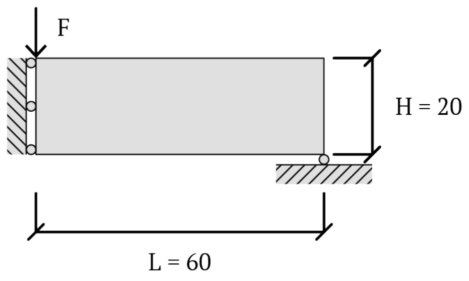

Fig. 3.1.1-1 Illustration of the MBB beam problem

It is clear from Fig. 3.1.1-2a that past low resolutions, `MRC` begins to show a significant improvement in the amount and time of computation as seen with the divergence of the plots of `top88` and `MRC(top88)`, reaching a striking 57.4% reduction in time at a resolution of 10. Note that for the purpose of this study, the maximum iteration count was capped at 1,000 for the MBB beam problem; if the original algorithm had been permitted to run indefinitely, the difference in computation time would be even more drastic, as `MRC(top88)` significantly reduces the required number of iterations.

To confirm the validity of `MRC`, it is important to compare the final compliances and geometries of the respective solutions. As seen in Fig. 3.1.1-2a for the MBB beam, there is surprisingly a consistent reduction in compliance when using `MRC`. At a resolution of 10, the compliance improves by a reduction of 3.2%. Comparing the image outputs at resolutions 1 and 10 (Fig. 3.1.1-2b), however, there is a visible difference in topology between the original and new solutions. This result may be attributed to the optimizer finding a different path to a solution due to a dilated filter radius during intermediate runs. Although there is this disparity in topology, the reduction in compliance shows that the solution using `MRC` is equally valid and even performs better in this case.

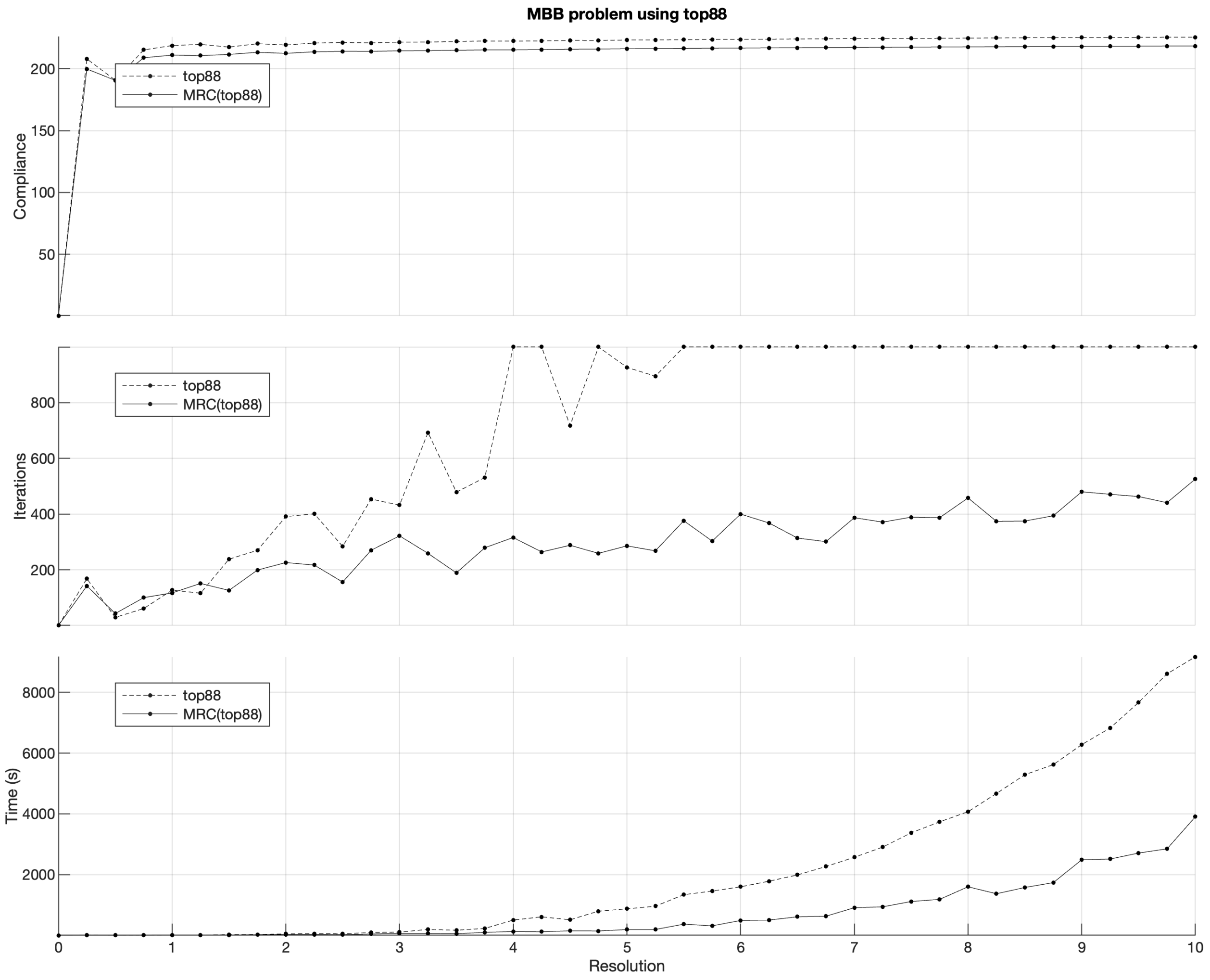

Fig. 3.1.2-2a Comparison of the compliance, iteration counts, and computation times vs. resolution for the MBB beam problem using `top88` and `MRC(top88)`

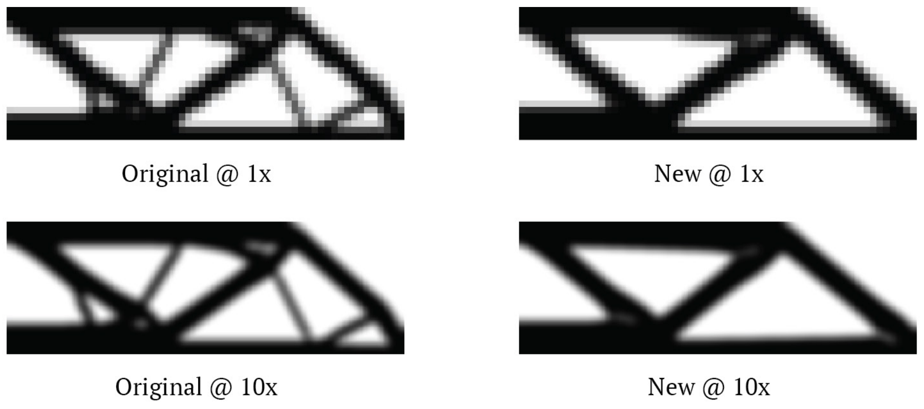

Fig. 3.1.1-2b Comparison of the geometry for the MBB beam at low and high resolutions using `top88` and `MRC(top88)`

3.1.2 Short Cantilever

The short cantilever problem features a fixed edge cantilevering with a point load at the bottom of the free end with the reference dimensions as shown in Fig. 3.1.2-1. This problem was also run up to a resolution of 10.

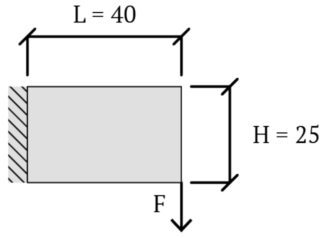

Fig. 3.1.2-1 Illustration of the short cantilever problem

A very similar trend shows for this problem, with the computation times diverging increasingly towards higher resolutions, though not as drastically as the trend from the MBB beam (Fig. 3.1.2-2a). There is only a 15.6% decrease in computation time at a resolution of 10 for the short cantilever problem because `MRC` tended to reach the iteration count limit more frequently this time. A comparison of the compliances and geometries also shows high similarity and validity of the output of `MRC` (Fig. 3.1.2-2b). The solutions generated by `top88` and `MRC(top88)` are nearly identical, which explains the nearly-equal compliances (0.7% reduction). Thus, with regard to the problems set forth in the original `top88` and its respective paper, `MRC` is consistently effective in reducing computation time while maintaining roughly equal compliance.

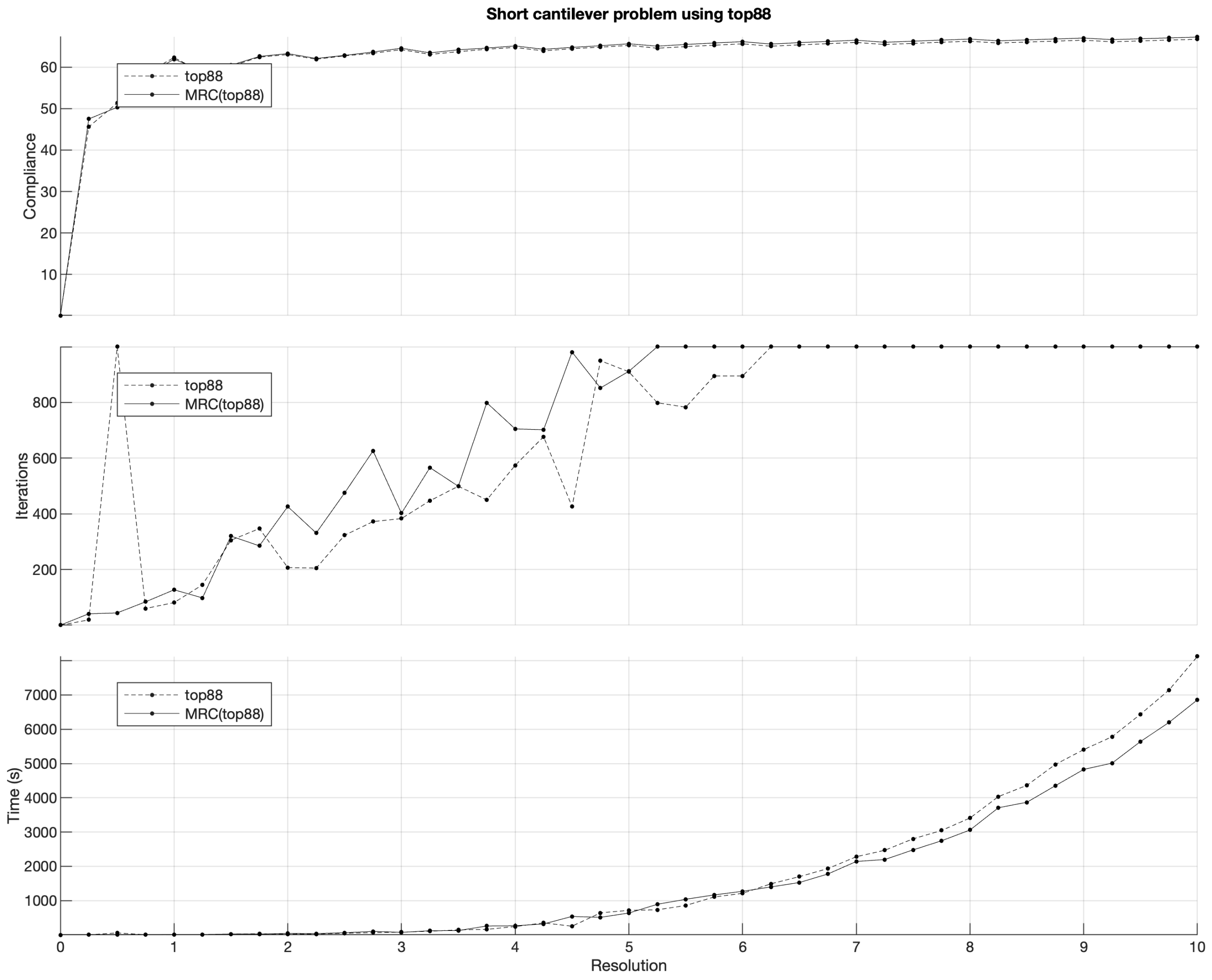

Fig. 3.1.2-2a Comparison of the compliance, iteration counts, and computation times vs. resolution for the short cantilever beam using `top88` and `MRC(top88)`

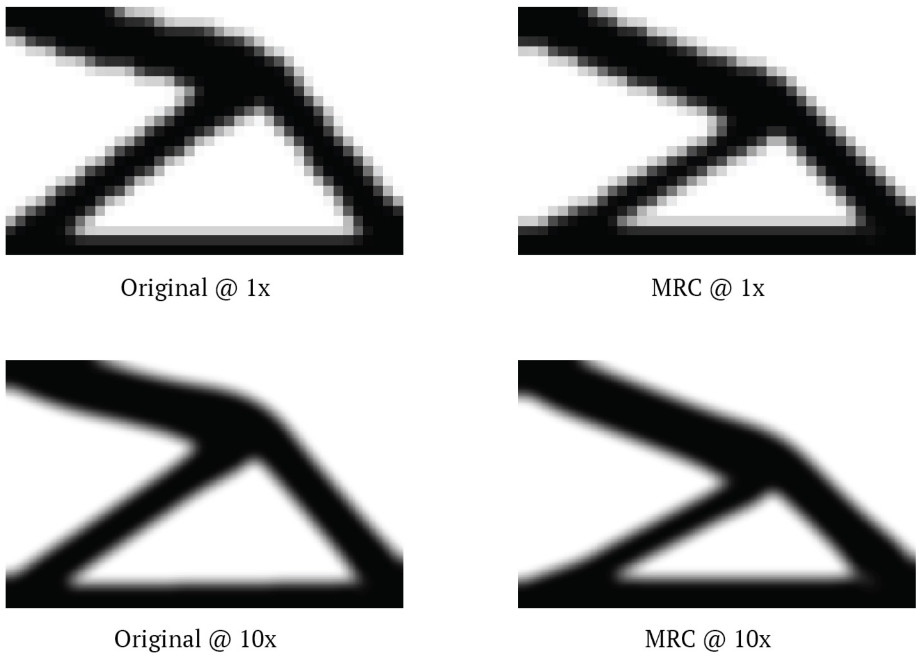

Fig. 3.1.2-2b Comparison of the geometry for the short cantilever beam at low and high resolutions using `top88` and `MRC(top88)`

3.2 Using `top99neo`

With promising results using `top88` as the base code, the next step was to continue experimenting with the successor algorithm, `top99neo` by Ferrari and Sigmund (2020). As the study claims a 2.55 to 5.5 times improvement in speed from `top88`, there was a possibility that `MRC` may have a less pronounced effect using such an intrinsically faster code. `top99neo` was similarly tested on the problems included in this study, including the same MBB beam problem as before, followed by a new and more complex frame reinforcement problem.

3.2.1 MBB Beam

The MBB beam is re-evaluated using `top99neo`, providing a useful comparison between the performances of the original algorithms published nine years apart, along with the performances of `MRC` using these as the base algorithms. Two major changes in `top99neo` from `top88` are the calculation method for the change threshold (at which point the change in densities between iterations is small enough to terminate the optimization loop) and the addition of a continuation feature for penalization and projection parameters. Due to these new features, each MBB beam test was also run with and without continuation at change cutoffs of 10^{-6}, 10^{-5}, 10^{-4}, and 10^{-3} to explore the effects of these parameters on the efficacy of `MRC`. `top99neo` by default runs without continuation and with a change cutoff of 10^{-6}; the same compliance, iteration count, and time versus resolution plots are shown in Fig. 3.2.1-1a for this default set of parameters.

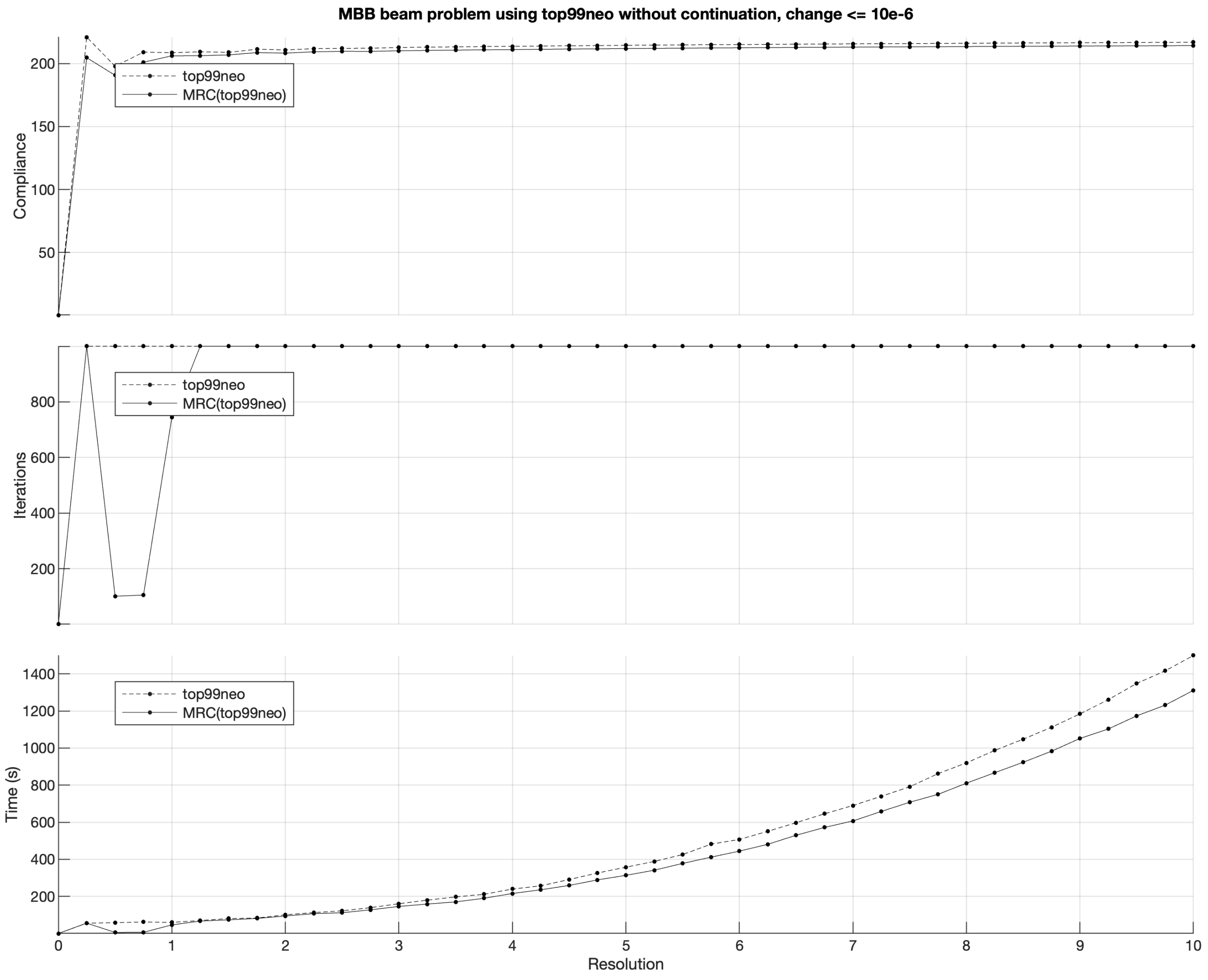

Fig. 3.2.1-1a Comparison of the compliance, iteration counts, and computation times vs. resolution for the MBB beam using `top99neo` and `MRC(top99neo)` without continuation and a change cutoff of 10^{-6}

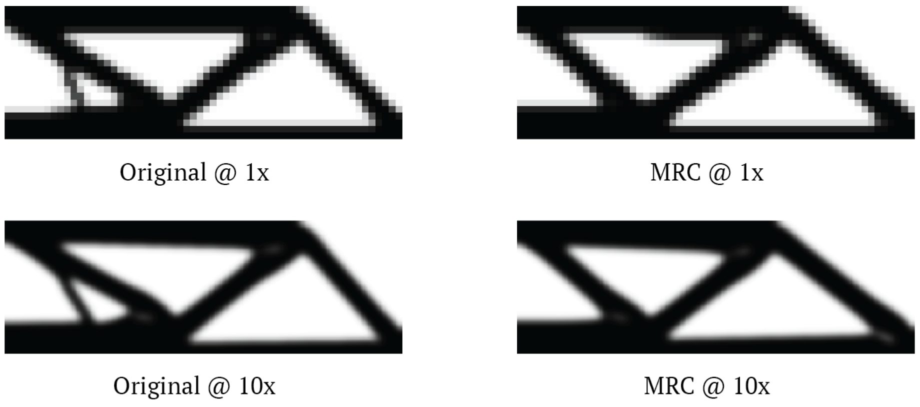

Fig. 3.2.1-1b Comparison of the geometry for the MBB beam at low and high resolutions using `top99neo` and `MRC(top-99neo)` without continuation and a change cutoff of 10^{-6}

Immediately there is a noticeable difference with this plot (Fig. 3.2.1-1a) compared to the previous analysis using `top88`. The divergence in computation time is far less drastic, with only a 12.5% reduction in time. The solution of `top99neo`, as seen in Fig. 3.2.1-1b, slightly differs from that of `top88` in topology, along with a 3.9% reduction in compliance, implying that `top99neo` by itself improved in efficiency and effectiveness from its predecessor code. On the other hand, the solution of `MRC(top99neo)` is almost identical to that of `MRC(top88)`; this confirms the theory that `MRC` is less advantageous when its base algorithm is intrinsically more efficient.

Fig. 3.2.1-2a and Fig. 3.2.1-2b show the same trend when continuation is applied using the same change cutoff of 10^{-6}. Note that discontinuities from invalid data in the plots occasionally arise at low resolutions due to breakdowns when the number of elements in each dimension is insufficient for analysis; however, this issue is negligible as resolutions below 1 are coarser than the intended use for most topology optimization codes.

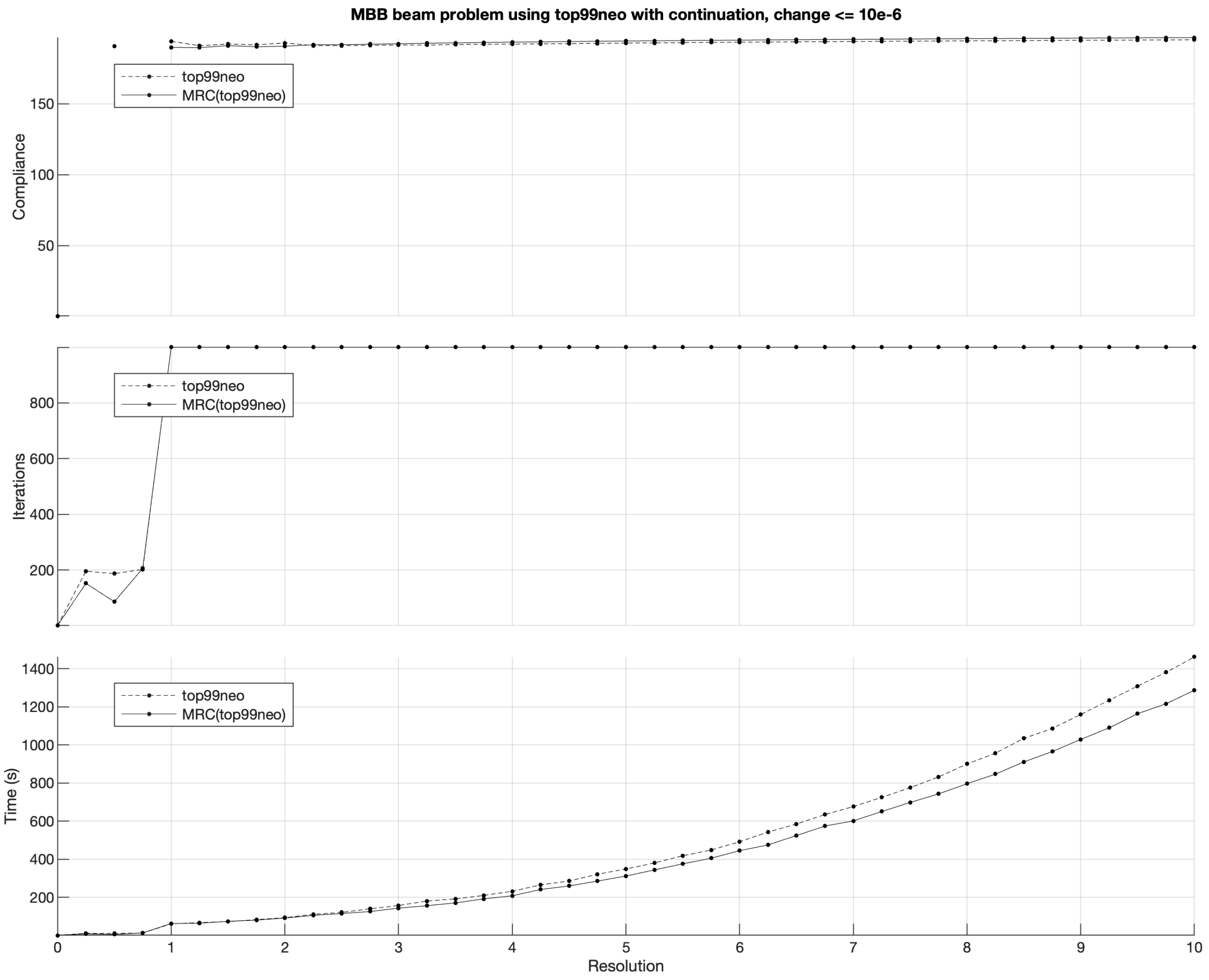

Fig. 3.2.1-2a Comparison of the compliance, iteration counts, and computation times vs. resolution for the MBB beam using `top99neo` and `MRC(top99neo)` with continuation and a change cutoff of 10^{-6}

The solutions generated with continuation, aside from the increased image clarity, closely match those generated without. With continuation, `top99neo` decreased even further in computation time by 2.5% and in compliance by 9.9% from running without continuation. `MRC(top99neo)` with

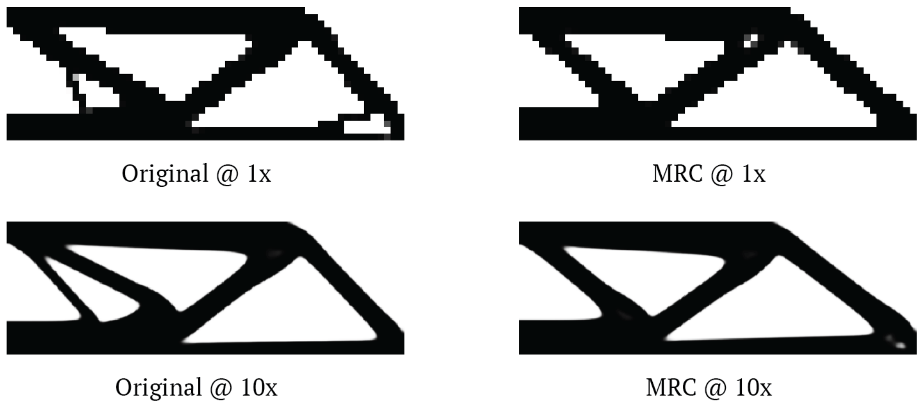

Fig. 3.2.1-2b Comparison of the geometry for the MBB beam at low and high resolutions using `top99neo` and `MRC(top-99neo)` with continuation and a change cutoff of 10^{-6}

continuation, while remaining better in performance, does not have the same level of improvement. It can be concluded that both `top99neo` and `MRC` improve upon `top88` to a large degree, but the difference is less significant relative to each other. Although this result may seem to undermine the effectiveness of `MRC`, it is important to note that the change cutoff of 10^{-6} is very strict. While both cases show a reduced benefit from using `MRC`, this can be explained by observing that the iteration counts are reaching the predefined limit for all resolutions past 1. Since the original and new algorithms are running the same amount of iterations, it follows that the computation times are much more similar, whereas before with `top88` there was a clear difference in the number of required iterations. Furthermore, the iteration counts are reaching a maximum due to the small change cutoff in the default set of parameters. If the change cutoff is relaxed to higher values, the required iterations should decrease. Figure 3.2.1-3 presents the time reductions between `top99neo` and `MRC(top99neo)` at the highest resolution of 10 for various change cutoffs.

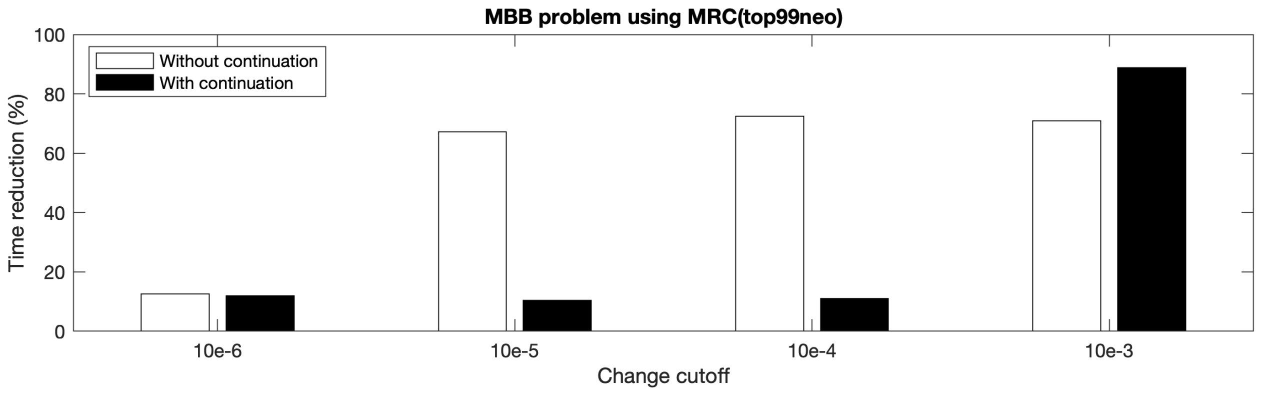

Figure 3.2.1-3 Comparison of the time reduction for the MBB beam using `MRC(top99neo)` across various change cutoffs, with and without continuation

It is evident from the large disparity in time reductions between a change cutoff of 10^{-6} and 10^{-3} that the change cutoff is an important factor in determining the speed of convergence. If the cutoff is relaxed, the optimization terminates earlier in less iterations, with which `MRC` clearly performs much better than the base code (up to 88.8% with continuation). This highlights an interesting trade-off between the change cutoff and the efficacy of `MRC`. Without continuation, `MRC` performs much faster with just an increase in the cutoff of a factor of 10.

3.2.2 Frame Reinforcement

The frame reinforcement problem from Ferrari and Sigmund (2020) is a more complex example that involves defining active/passive and solid/void regions, as well as a distributed load, which renders it an important benchmark problem for both the original `top99neo` script and `MRC(top99neo)`. Featuring a 900×900 discretization in its default case, this super-fine mesh is also useful in testing the limits of the algorithms and their performances with large datasets. Tests with this problem are capped at a resolution of 3 due to system memory shortages that arose during runtime and insufficient MATLAB session durations for any higher resolutions, which amount to over 7 million elements and required upwards of 12 hours for each run. Continuation is not applied per the original example, the change cutoff is fixed at the original 10^{-6} due to the low likeliness of it controlling the loop, and the dimensions are reproduced and illustrated as in Fig. 3.2.2-1.

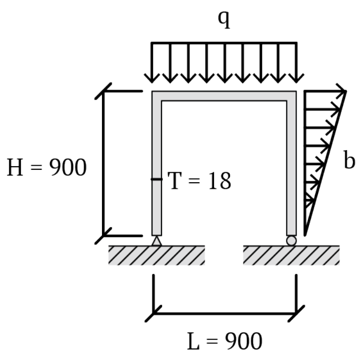

Fig. 3.2.2-1 Illustration of the frame reinforcement problem

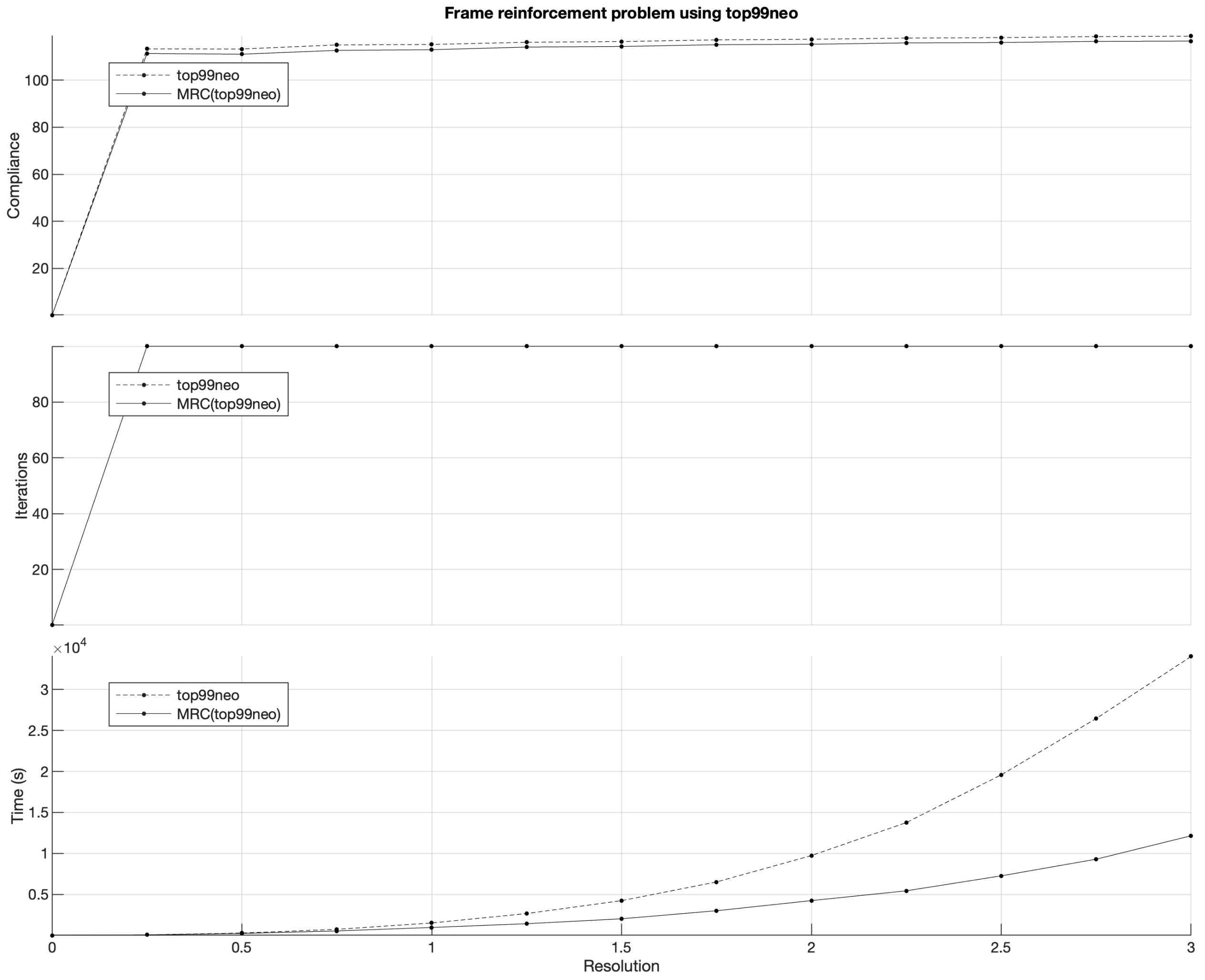

Fig. 3.2.2-2a Comparison of the compliance, iteration counts, and computation times vs. resolution for the frame reinforcement problem using `top99neo` and `MRC(top99neo)`

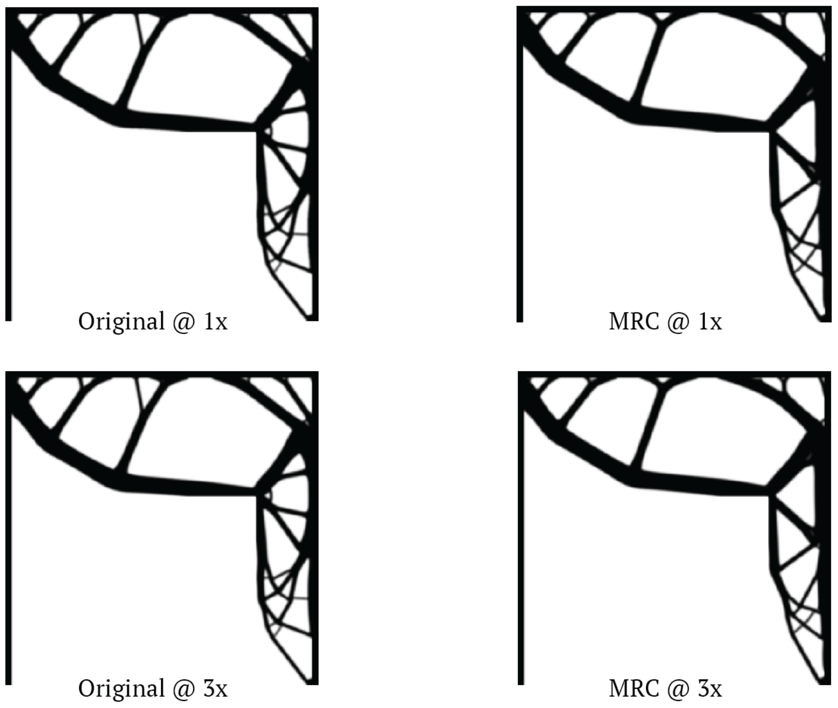

Fig. 3.2.2-2b Comparison of the geometry for the frame reinforcement problem at low and high resolutions using `top99neo` and `MRC(top99neo)`

3.3 Using `top3D125`

A final exploration of the potential of `MRC` focuses on the 3D cantilever problem solved with `top3D125`, the 3D extension of `top99neo` by Ferrari and Sigmund (2020). The dimensions as listed in the paper are reproduced as shown in Figure 3.3-1. Tests for this problem were run only up to a resolution of 1.5, past which MATLAB consistently crashed on both `top3D125` and `MRC(top3D125)`.

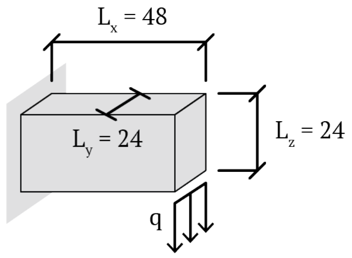

Fig. 3.3-1 Illustration of the frame reinforcement problem

With just six data points for each algorithm (Figure 3.3-2a), there is already a clear divergence in computation time, reaching a 59.5% reduction by a resolution of just 1.5. Although its performance past this point is not confirmed in this study, it can be assumed that `MRC` would continue to produce significant time savings at higher resolutions.

In Figure 3.3-2b, `MRC(top3D125)` can be seen to generate quite a different solution from the original `top3D125`. The solution of `MRC` seems to have fewer and larger members and fewer nodes; this may be attributed to the relatively low resolution compared to previous tests in this study, such that the continuation of the mesh refinement does not develop enough to escape the effects of a large filter radius. Nevertheless, the compliance is still improved with `MRC`, reducing from `top3D125` by 1.8%. This result is promising as it shows the potential of `MRC` for complex problems such as this 3D test case.

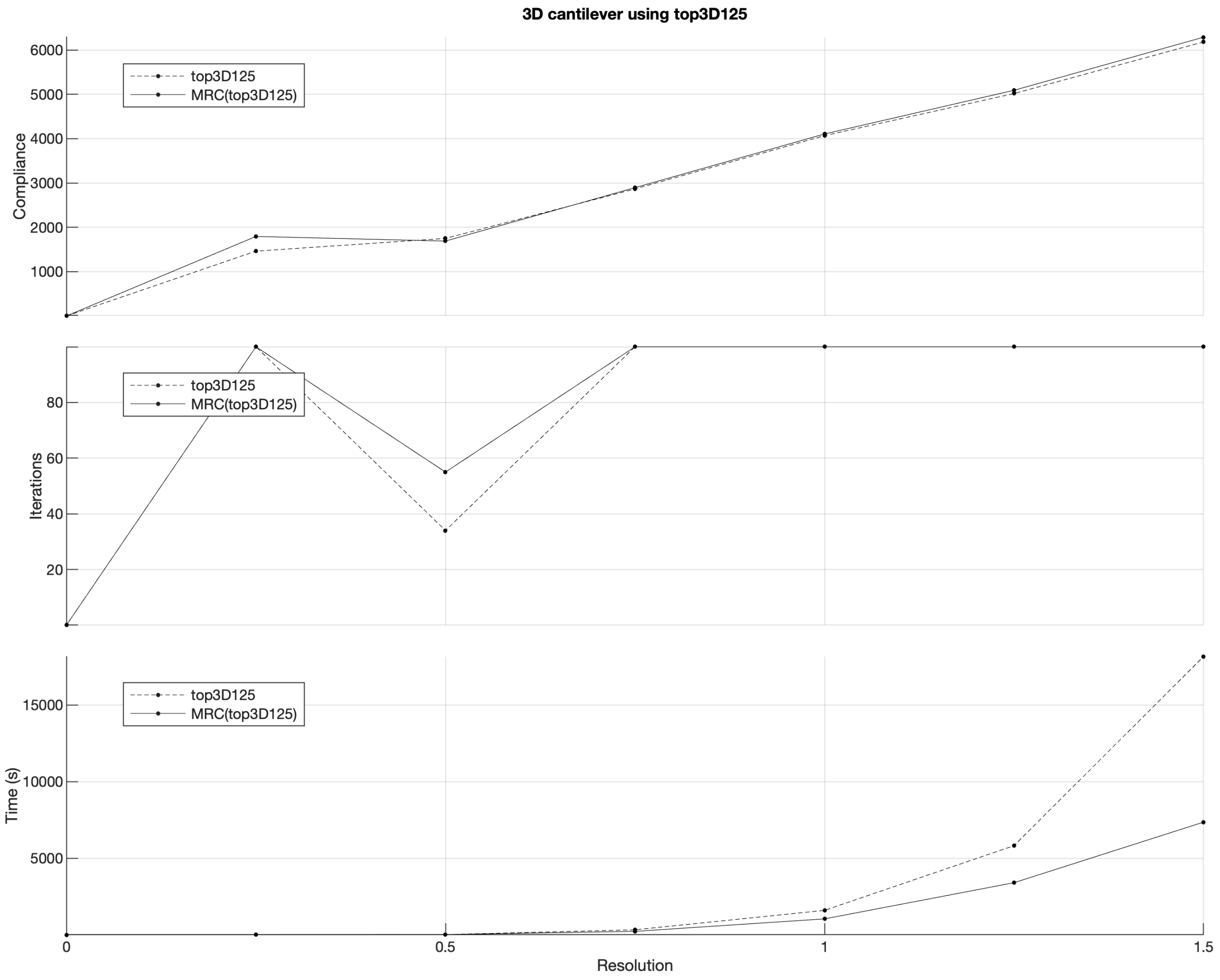

Fig. 3.3-2a Comparison of the compliance, iteration counts, and computation times vs. resolution for the 3D cantilever problem using `top3D125` and `MRC(top3D125)`

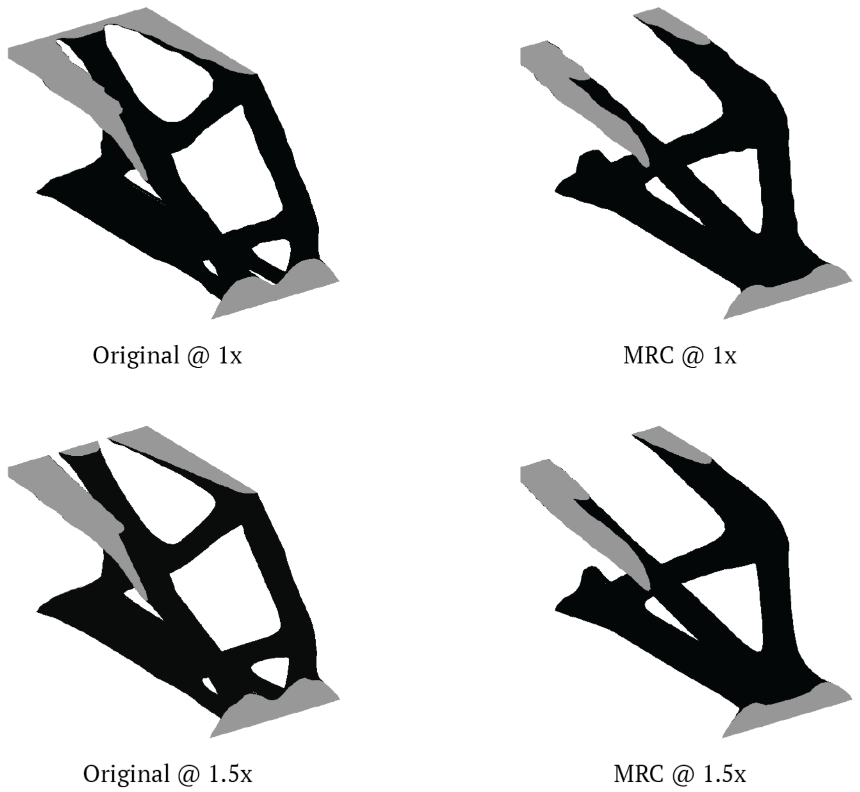

Fig. 3.3-2b Comparison of the geometry for the 3D cantilever problem at low and high resolutions using `top3D125` and `MRC(top3D125)`

4 Discussion

After exploring the applicability of `MRC` to the common topology optimization codes, it is evident that the benefits vary from case to case. While `MRC` generally shows a significant improvement in computation speed, depending on the level of complexity desired, the original code may still perform better or more stably. Table 4.1 summarizes some results of the previously mentioned tests, demonstrating the large reductions compared to the respective base codes at the highest resolutions tested ranked from lowest reduction to highest.

Problem	**Top resolution**	**Equivalent mesh**	**Total number of elements**	**Time reduction using `MRC`**
MBB beam (`top99neo`)	10	600×200	1.2×10^6	12.5%
Short cantilever (`top88`)	10	400×250	1.0×10^6	15.6%
MBB beam (`top88`)	10	600×200	1.2×10^6	57.4%
3D cantilever (`top3D125`)	1.5	72×36×36	9.3×10^4	59.5%
Frame reinforcement (`top99neo`)	3	2700×2700	7.3×10^6	64.3%

Table 4.1 Summary of time reductions across different benchmark problems

By far, `MRC` saw the most improvement when applied to `top99neo` on a high-complexity, fine-mesh test case like the frame reinforcement problem. On the other hand, `MRC` saw the least improvement for the same base code, but on the simple MBB beam problem. One possible explanation is that topology optimization codes by now are perfected for a classic benchmark problem like the MBB beam, such that any further improvement in computation speeds is very difficult to achieve. Whereas for less-tested cases like the frame reinforcement or 3D cantilever problems, there is still wide room for improvement, as these are more specialized cases and originate in relatively more recent publications.

Besides the complexity or discretization of problems, certain parameters also affect the performance of `MRC`. By far the most influential parameter is the change cutoff by which the optimization loop references as a breakpoint. For the MBB beam problem run with `top99neo` and `MRC(top99neo)` without continuation, relaxing the change cutoff by a factor of 10 instantly increased time reductions from 12.5% to 67.3%, as this effectively terminates the loop earlier. As the average time per iteration using `MRC` is shorter due to a large portion of its iterations being run at low resolutions, `MRC` also benefits in cases where no iteration limit is set. Relaxing the change cutoff again by another factor of 100 produces a similar spike in improvement when continuation is applied, from 12.0% to 88.8% in its time reduction. From these results, there are evidently several main situations in which `MRC` provides the most benefit: when problems are complex, when super-fine discretizations are desired, when no continuation is applied, or when there is no maximum iteration count.

5 Conclusion

Presented in this paper is the concept for a new mesh refinement continuation (`MRC`) algorithm that enhances the performance of common topology optimization codes, such as the 88-line MATLAB code by Andreassen et al. (2011) and the 99-line and 125-line 3D code by Ferrari and Sigmund (2020). By iteratively reusing outputs from coarser meshes as inputs for finer meshes, `MRC` is able to take advantage of higher computational speeds at low resolutions to more efficiently guide the optimization down a path to a minimum. For benchmark problems as outlined in their respective original papers, `MRC` produces reductions in computation times of up to approximately 60% on discretizations on the order of 10^6 elements.

The tests conducted in this study are far from comprehensive, and the full range of potential (or drawbacks) of this approach is yet to be explored. Presented here is only a concept and not a definitive algorithm by any means; perhaps with further refinement of parameters, `MRC` can become an even more effective and robust method. The initial success of `MRC` is a promising step towards fulfilling the goal of increasing the accessibility of topology optimization by improving the efficiency of existing tools in more innovative ways.

6 References

Andreassen, E., Clausen, A., Schevenels, M., Lazarov, B. S., & Sigmund, O. (2010). Efficient topology optimization in MATLAB using 88 lines of code. In Structural and Multidisciplinary Optimization (Vol. 43, Issue 1, pp. 1–16). Springer Science and Business Media LLC. https://doi.org/10.1007/s00158-010-0594-7

Ferrari, F., & Sigmund, O. (2020). A new generation 99 line Matlab code for compliance topology optimization and its extension to 3D. In Structural and Multidisciplinary Optimization (Vol. 62, Issue 4, pp. 2211–2228). Springer Science and Business Media LLC. https://doi.org/10.1007/s00158-020-02629-w

Sigmund, O. (2001). A 99 line topology optimization code written in Matlab. In Structural and Multidisciplinary Optimization (Vol. 21, Issue 2, pp. 120–127). Springer Science and Business Media LLC. https://doi.org/10.1007/s001580050176

Sigmund, O., & Maute, K. (2013). Topology optimization approaches. In Structural and Multidisciplinary Optimization (Vol. 48, Issue 6, pp. 1031–1055). Springer Science and Business Media LLC. https://doi.org/10.1007/s00158-013-0978-6

Wicklin, R. (2014, June 11). *How to find an initial guess for an optimization*. SAS Blogs. https://blogs.sas.com/content/iml/2014/06/11/initial-guess-for-optimization.html

Appendix

Modification to the Base Code

The following edits were made to `top88` to be compatible with the `MRC` code. Similar edits following the same concepts can be made to `top99neo` and `top3D125`.

Line 1

```
function [out,loop] = top88(nelx,nely,volfrac,penal,rmin,ft,maxit,in)
```

`out` (the solution matrix) and `loop` (iteration count) must be added to an output array, and `maxit` (iteration limit) and `in` (input matrix) must be added as an argument.

Line 46

Line 46 is replaced with the following block:

```
if isempty(in)
    x = repmat(volfrac,nely,nelx);
else
    x = in;
end
```

In the `MRC` algorithm, the initial run initializes the design variable matrix as prescribed by the input parameters and the setup code unique to each problem. For the initial run, the input is passed as an empty array to signify that the default operation for matrix assembly should be carried out. For subsequent runs, the design variable matrix is no longer initialized by the default operations but is rather replaced with an input matrix.

Line 51

```
while change > 0.01 && loop < maxit
```

A second condition must be added to break the loop once the count reaches a limit.

Line 88

An optional line of code that renames `xPhys` as `out` is added between lines 87 and 88 for clarity.

```
out = xPhys;
```

MRC Code with Documentation

```
%% Define inputs
X = 60;                                            % equivalent of nelx from top88
Y = 20;                                            % equivalent of nely from top88
volfrac = 0.5;
penal = 3;
rminphys = 1.5;                                    % equivalent of rmin from top88
ft = 2;
res = 10;                                          % target resolution
maxit = 1000;                                      % iteration limit
%% Initialize parameters and looping variables
r = max(4/min(X,Y),1/8);                           % resolution iterator
nelx = ceil(r*X);                                  % scale mesh size to the current resolution
nely = ceil(r*Y);
relax = 2.5;
rmin = relax*r*rminphys;                           % scale filter size to the current resolution
split = 2;                                         % split each element in half both ways
runs = ceil(log(res/r)/log(split));                % count the total number of runs
maxit_interm = 10;                                 % maxit iterator for initial/intermediate runs
%% Main loop
in = [];                                           % initial input defaults to top88 internal setup
[meshx,meshy] = meshgrid(0:Y/(nelx-1):Y,...        % defines a map for the initial mesh
    0:X/(nely-1):X);
[out,iters_total] = top88(nelx,nely,volfrac,...    % initial run
    penal,rmin,ft,maxit_interm,in);
for i=1:runs
    r = min(r*split,res);                          % ensure resolution is capped at the target
    nelx = ceil(r*X);
    nely = ceil(r*Y);
    [meshxq,meshyq] = meshgrid(0:Y/(nelx-1):Y,...  % defines a new map with the refined mesh
        0:X/(nely-1):X);
    in = interp2(meshx,meshy,out,meshxq,meshyq,... % projects values using the maps
        'linear');
    meshx = meshxq;                                % update map for next run
    meshy = meshyq;
    maxit_interm = maxit_interm + 5;
    if (i<runs) && (iters_total+maxit_interm>=...  % skips a run if maxit will be exceeded early
            maxit)
        out = in;
        continue
    elseif i==runs                                 % final run
        rmin = r*rminphys;                         % rmin returns to original
        [out,iters] = top88(nelx,nely,volfrac,...  % runs with the remaining iterations
            penal,rmin,ft,maxit-iters_total,in);
    else                                           % intermediate runs
        rmin = relax*r*rminphys;
        [out,iters] = top88(nelx,nely,volfrac,...
            penal,rmin,ft,maxit_interm,in);
    end
    iters_total = iters_total + iters;             % update iteration count
end
```