Clemson University

TigerPrints

All Theses Theses

May 2020

Scanning Pattern Optimization to Reduce Temperature Gradients in DMLS

Caroline Victoria Buck
Clemson University, buckc13@gmail.com

Follow this and additional works at: https://tigerprints.clemson.edu/all_theses

Recommended Citation

Buck, Caroline Victoria, "Scanning Pattern Optimization to Reduce Temperature Gradients in DMLS" (2020). *All Theses*. 3332.
https://tigerprints.clemson.edu/all_theses/3332

This Thesis is brought to you for free and open access by the Theses at TigerPrints. It has been accepted for inclusion in All Theses by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.

SCANNING PATTERN OPTIMIZATION TO REDUCE TEMPERATURE GRADIENTS IN DMLS

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Mechanical Engineering

---

by
Caroline Victoria Buck
May 2020

Accepted by:
Dr. Georges Fadel, Committee
Chair Dr. Gang Li
Dr. Xin Zhao

ABSTRACT

Direct Metal Laser Sintering (DMLS) is a specific type of additive manufacturing that is used to create metal parts. In this process, an optical laser is used to melt a metal powder, fusing the powder into a solidified form. The laser follows instructions from a CAD model, which illustrates the design and from which the layer pattern the laser should follow is extracted. A scan path is generated for the laser to trace on the powder, that initially follows the perimeter, and then draws parallel neighboring contour lines in the interior of the scanned perimeter to cover the surface and form a consolidated layer. Currently, there is an issue with DMLS builds; due to the high heat generated by the laser and the uneven cooling patterns of the metal after fusion. Deformations are forming in the design builds after cooling. Today, additive manufacturing technicians and or designers are having to adjust the original CAD files through the use of predictive software to account for the deformations caused by the laser heat. After an extensive literature review, it was confirmed that the scanning pattern the laser takes in the DMLS to sinter the powder metal affects the magnitude of the temperature gradient and thus affects subsequent build part deformations.

A simulation that computationally mimics the moving laser heat source in DMLS on a build-part allows for control of the scanning pattern and the number of layers to be scanned and returns the temperature profile of the part as it is built and subsequently cooled. A genetic algorithm is tied to the simulation in order to optimize the scanning pattern of the laser with the ultimate goal to reduce the overall temperature gradients induced on the build part. The effects of layer build-up were also investigated with the optimization of the scanning pattern of the laser. The combination of the DMLS simulation and Genetic Algorithm application show qualitatively that optimal scanning patterns can reduce the temperature gradients from the DMLS process. This research work is meant to be a basis for the optimization of scan pattern on specific build part designs and be able to be applied to a wider array of designs as well as in-situ DMLS builds in the future.

DEDICATION

This thesis is dedicated to my all my friends—both at Clemson and beyond—and family, whose endless support made the completion of this research possible.

ACKNOWLEDGMENTS

This research project would not have been possible without the continued help and support from many people. First, an enormous thanks to my parents, Margaret and Bill, for their unconditional support and love the past two years. Also, a huge thanks to my roommates over the past two years who were always there to help provide life perspective and keep me motivated.

I would like to thank Dr. Fadel and my committee members for their help throughout my project. Thanks to Dr. Fadel for giving me the opportunity to work on this project and working through the various roadblocks with me that appeared throughout this research project. I would also like to thank Dr. Zhao for his continued insight and help into the specifics of the laser modelling and Dr. Li for his continued insight and help into the specifics of FEA and simulating the DMLS process.

I would like to thank Jennifer Snipes and S. Ramaswami for allowing me to use and build upon the research work they had done during their time at Clemson in post-doctoral positions. I would like to thank Rama for his continued support and help in understanding all they had done; I greatly appreciated him taking the time to answer my questions to help understand and finish the simulation.

I would like to thank the people who support and run Clemson's Palmetto Cluster, as this project would not have been possible without the Cluster's high computing resources.

I would like to extend thanks to Grigori Yourganov for the immense help with implementing my codes onto the Palmetto Cluster. The optimization would not have run nearly as efficient or smoothly without his persistence and help in reworking and debugging the genetic algorithm codes on the Cluster. I appreciate his willingness to answer all questions I had and work with me for what this project needed specifically.

Lastly, I would like to thank the CEDAR group for their constructive criticisms and insightful comments for ways to improve my project. I would like to thank John Kremar for helping me work through and debug the genetic algorithm code, and I would like to thank Sindora Baddam and Nicole Zero for their support in various research challenges.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

1.1 Direct Metal Laser Sintering

1.1.1 Process description

Additive manufacturing (AM) is a burgeoning process which can create solid 3-dimensional parts directly from computer-aided design (CAD) file drawings. A vast array of materials are being made increasingly available in additive manufacturing processes, including metals and even multi-materials. One such process that has gained popularity is Direct Metal Laser Sintering (DMLS). Developed by EOS GmbH in Munich, Germany in the 1990s, the DMLS process is able to build metal parts through additive manufacturing [1], [2]. DMLS builds such parts by using a laser to sinter layers of metal powder into a liquid phase—more analogous to a viscous fluid— onto a substrate, as shown in Figure 1.1, and then allowing the layers to cool and solidify into a solid, fully dense metal part [1], [3]. Sintering in this case is using the laser to heat the metal powder just below the liquid phase melting temperature, thereby allowing for the heated metal to bond with the surrounding metal [1], [4]. Variations of the process melt the metal and part of the substrate and adjoining built structure to form a fully dense part.

To begin, "a powder layer (about 50 μm in thickness) is spread on the base plate using a moving wiper (mechanical re-coater) [5]". From the U.S. Patent for the Direct Metal Laser Sintering Machine, the process is described as follows: "After the powdered material is deposited, [the] laser head emits a laser beam at the powdered material to melt or sinter the powdered material. After the laser beam has melted or sintered the powered material, a solid layer of material is formed. This process is continued along the design of the part until the part is completely formed of solid material [6]."

A representation of the DMLS process is shown in Figure 1.1, including both the full system set up as well as a closer look at the laser beam and powder metal interactions.

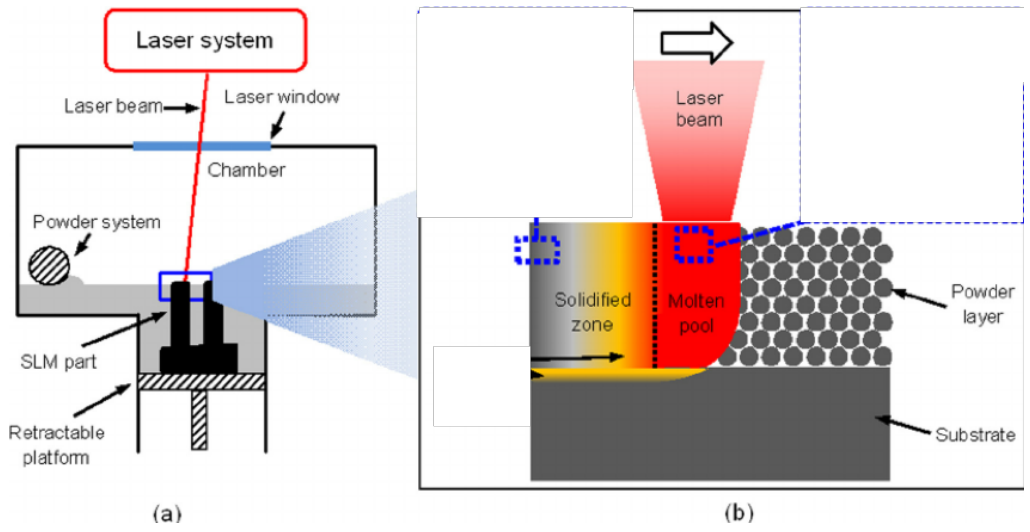


Figure 1. 1 Representation of the DMLS Process [3]

DMLS is claimed to have many benefits over other means of building three-dimensional metal parts. In particular, DMLS is not as costly as, and is industrially safer than, traditional machining methods [7]. DMLS offers a solution to building three-dimensional complex parts without having to use subtractive processes such as milling or lathing, which can sacrifice complexities in build parts [3].

1.1.2 Residual Stress Issues in DMLS

While a viable solution to many metal part builds, the DMLS as is currently implemented has drawbacks that make it more challenging in many cases to build complete parts. Perhaps the most significant drawback is inherent in the use of a laser to sinter the metal. Since the laser generates high heat at a very small spot, heating and cooling are highly uneven; the upper layers are rapidly heated while the bottom layers experience slower cooling through conduction [3]. The rapid heating and slow cooling induces steep temperature gradients in the part as it is built layer by layer. These temperature gradients ultimately cause deformations in the layers as they become restricted in their expansion and compression due to the heating and cooling cycles, respectively [3]. As the laser heats the top layer, the metal's expansion into the liquid phase is restricted by the layers around and beneath it and the previously scanned paths on the layer; a tensile stress is induced on the section of the material being directly heated and a compressive stress is induced on the surrounding material. As the laser then moves on, the metal recently heated begins to cool, but cannot contract as it is sintered to the adjacent layers; compressive stresses are induced on the element just heated and tensile residual stresses are induced on the surrounding material top surface [3], [7].

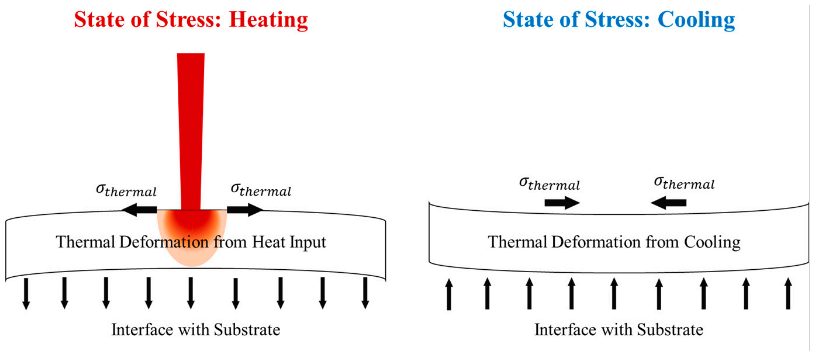


Figure 1. 2 Thermal Deformations And State of Stress in DMLS [7]

As the scans continue, these residual stresses accumulate and can eventually reach the yield stress of the material, causing distortion and cracking [8][9]. Currently, there is insufficient control of these residual stresses in DMLS, which leads to a "detrimental effect on the manufacturability and integrity of a component. [10]." Engineers use predictive software to adjust designs prior to build in order to compensate for the inevitable deformations in DMLS [11].

1.2 Preliminary Research Questions and Research Motivations

Given the current issues in Direct Metal Laser Sintering related to steep temperature gradients during the build and residual stress accumulation on the build part, it is important to research ways to control and overall reduce the residual stresses and subsequent warping in DMLS; doing so will help mitigate the need for designing to predict manufacturing deformations, and allow for a smoother build process. One hypothesis is that the scanning pattern affects this deformation. From this, the first research question is posed for this research.

1. How can one generate an optimal scanning pattern to minimize the thermal gradients induced on the part from the laser, thus minimizing the post-build deformations?

The thermal gradients are found from calculating the temperature differences across the build part surface before and or after cooling. From the first question, the following hypothesis is made: *The lower the temperature gradient over the build part, the lower the thermal stresses, and thus the lower the amount of total part warping post-build.*
The first research question begs the second.

2. Which metric should be used in optimization to address the thermal gradient problem in DMLS?

From the second question, the following hypothesis is made: *The variations of temperature on the surface and in the part are indications of a temperature gradient.*

This research aims to find a method to reduce the thermal gradients in a part during build. Finding a metric to define the temperature gradient helps to define improvements in the DMLS process from scanning pattern optimization; doing so helps answer the first research question, which can then help to develop a more controlled and reliable DMLS process.

In order to answer both questions, this research proposes a computational simulation of the Direct Metal Laser Sintering process on a simple part, and an optimization of the scanning pattern used by the laser in order to reduce the uneven heating and cooling cycles the part undergoes; this follows the hypothesis that by lowering these temperature gradients, the overall deformations will be reduced.

1.3 Thesis Organization

This thesis is organized in the following manner. In Chapter 1, the Direct Metal Laser Sintering process is discussed and the current residual stress problems inherent with the process are presented. The research questions this thesis covers and the motivations for these questions are also presented. Chapter 2 reviews the literature on the residual stress problem, the thermal gradient definition, and the effects of scanning patterns on the residual stresses in DMLS. Chapter 3 outlines the DMLS simulation used in this research, including parameters and heat transfer effects considered; this chapter also presents preliminary runs of scanning patterns used in DMLS and the generated benchmark used to assess and compare optimization results. The fourth chapter explains the optimization approach based on a genetic algorithm and created for this research project, the combining of the algorithm with the DMLS simulation, and the intermediate and final optimization results. A comparison between the scanning patterns pre- and post-optimization is also shown here, with the final optimal scanning pattern corresponding to the greatest thermal gradient improvement selected. Chapter 5 summarizes the main conclusions of this research and suggests future work to further expand on the idea of optimizing the scan pattern in DMLS to minimize thermal gradients and validating the numerical results.

CHAPTER 2

LITERATURE REVIEW

2.1 Residual Stresses and Distortion

Previous studies, both based on simulations and experiments, have analyzed the process and end results of residual stress build up over the course of a metal part being built via DMLS. They focus on the subsequent distortion, warping, cracking and negative effects on material part properties resulting from the residual stresses [7],[8],[12]. Residual stresses are “stresses within a plastically- or elastically- deformed material that remain within the structure after the load that deformed it is removed [7].” In the DMLS process, the loads are the thermal loads resulting from the heat flux generated from the laser and incident on the metal powder and substrate. The cyclic heating and cooling on the build part creates internal stresses within the parts; these internal stresses can reach the yield stress of the material, leading to cracks and reduction of fatigue life in experimental work [3],[8].

As shown in Figure 1.2 and Figure 2.1 below, the main mechanism for the residual stress generation is due to the high thermal gradients (Thermal Gradient Mechanism) surrounding the moving laser spot [13].

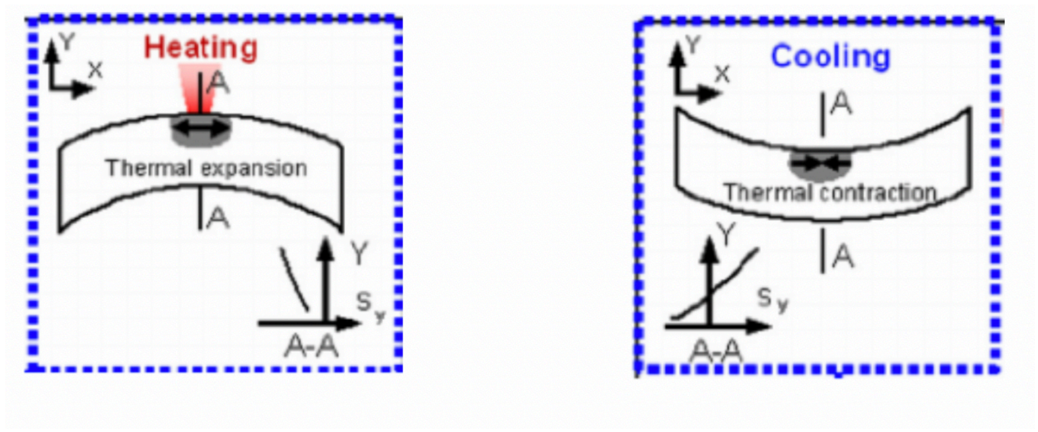


Figure 2. 1 Heating and Cooling from Laser and Subsequently Induced Stresses [3]

From the high heat on the upper layers, and the slow cooling from heat conduction on the previously scanned layers, a high temperature gradient builds across the build part [13]. As the laser scans and heats the top layer, the directly heated portion of the layer tries to expand and experiences tensile stresses; as the previously scanned layers begin to cool, the layers experience contraction and, therefore, compressive stresses [7], [14]. The largest tensile stresses are found to be at the top and bottom of the part, while there are compressive stresses of lesser degree are found throughout the cross-section of the fully built part [12]–[14]. Once the part is removed from the substrate and other support structures, the stresses are relaxed but bending deformation occurs [13]; the tensile stresses built up on the top surface cause curling across the part when it is not anchored with supports to the base plate [15]. Deformations are worse in overhanging features [7] and around the edges, such as when dealing with a plate [9].

Residual stress build-up and the subsequent part deformation is highly dependent on the number of layers required to build the final part; as the number of layers increases, so does the total deflection and final residual stresses measured [9], [13]. In a single layer scan, the stresses would dissipate with the cooling, but this is not the case with multiple layers [7]. The magnitude of residual stresses have also been found to decrease with thicker base plates [13]. The accumulation of the residual stresses in a part being built with DMLS also depends on scanning tracks and directions. The residual stress measurements are higher – sometimes as much as double—in the direction parallel to the scan vectors [10], [14], [15]. The scan vectors are the scan tracks the laser takes across the part during the DMLS process. The stresses also increase with vector length [10]. With the successive melting of adjacent tracks for one scan, transverse stresses increase due to the thermal expansion of the melt pool being inhibited by the previously scanned vector [10].

Certain process parameters have also been observed to affect residual stress accumulation and deformation in the build parts from DMLS through experiments. A longer laser exposure time on the part was found to increase the peak temperature on the part, which in turn increases the distortions on the build part [12]. Increasing the laser scan speed was found to significantly reduce the residual stresses, but sacrificed the density of the final build part [3],[9].

2.2 Temperature Gradient

Another important aspect to consider in DMLS when looking at part distortions is the temperature gradient mechanism (TGM). By understanding the TGM, and how to measure and define it, the problem of reducing the residual stresses accumulating in the DMLS can begin to be tackled. The temperature gradient mechanism controls many outcomes in DMLS including but not limited to the tensile stresses and part distortion. Researchers have developed a Finite Element (FE) model of DMLS with a moving heat flux source to look into the effects of process parameters on the melting process, as well as gain a better understanding into the temperature gradient mechanism [3]. From the simulation process, whose parameters and details are described in more detail in Section 2.4, many characteristics of the TGM were discovered. One such characteristic is that the temperatures across the part and along the scanning direction drastically decrease when moving from the center of the laser beam towards its edge. In addition, the temperature gradient becomes much steeper with high laser powers. The temperature gradient was found to be much higher in the depth direction of the part than along the width or scanning direction, as each additional layer significantly affects the temperature gradient [3].

Another model was created in a past study in order to understand the transient temperature field in laser melting AM processes. This study backed the understanding that the material undergoes rapid heating and cooling, which increases the thermal stresses [16]. It also supported the idea that each successive layer being scanned raises the temperature in the previous layers, resulting in a steady temperature build up in the previously scanned layers with each new layer added. The upper layers had a significantly higher maximum temperature, which could be due to both the temperature build up as well as the first layer having a lower conductivity than the base plate properties chosen for the model; the upper layers maintain their heat as they are farther from the base, which can act as a heat sink [16].

2.3 Effects of Scanning Pattern in DMLS

Researchers have studied the effects of various process parameters and their effects on the distortion outcomes of DMLS parts. In particular, multiple studies have simulated the DMLS process and analyzed the effects of scanning patterns on the distortion outcomes. One study [17] attempted to minimize distortion of FE simulated laser-processed components by testing different scanning patterns. The first pattern tested in the study was the traditional raster scan along a simulated square of nickel, where raster scan means the laser scans back and forth along adjacent parallel vectors for each layer. This scan pattern created a saddle-shaped distortion, shown in Figure 2.2.

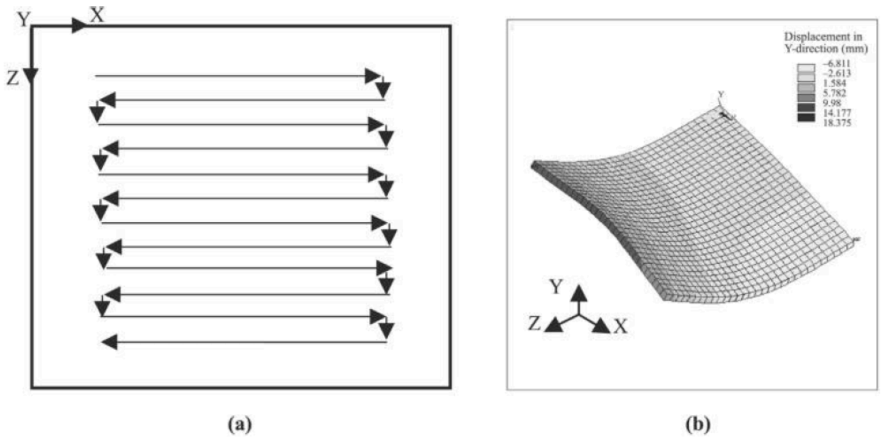


Figure 2. 2 Raster Scan and Subsequent Distortion from [17]

The saddle-shaped distortion is a product of both out-of-plane distortions along both the X- and Z- directions (note that in this paper, the coordinate system is not the usual one which has X and Y defining the plane and Z defining the out of plane direction) [17]. The authors then tested a spiral scanning pattern from the outside of the simulated plate inward towards its center; by creating a scanning pattern that goes in both X- and Z-directions, the concave upward and concave downward distortions negated each other to some extent. The decreased distortion from the spiral scan pattern is shown in Figure 2.3.

This study [17] was one of the first to show that the scanning pattern has an effect on the distortion of built parts in laser-processed components.

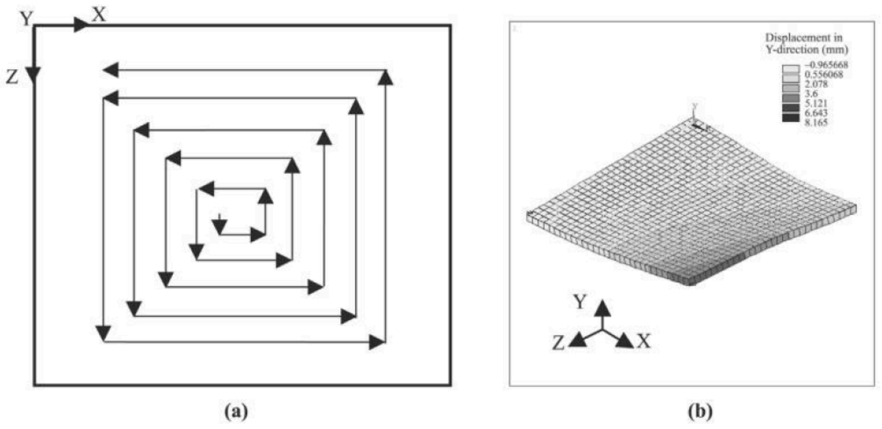


Figure 2. 3 Spiral Scan and Subsequent Distortion from [17]

Another study [10] looked at the scanning pattern effect on residual stress build-up and distortion by looking at a raster scan with unidirectional and alternating directions. The study looked at the start and end temperatures of the scan vectors between each scan strategy in order to understand the differences in residual stress build up; it was found that alternating directional scan strategy had each adjacent scan starting at higher temperatures due to the previous scan just finishing at a location adjacent to the next starting point, as opposed to the unidirectional scan strategy. While the alternating directional scan showed a lower temperature gradient than the unidirectional scan, the temperature gradients for both scans become independent of scan direction after a few seconds. This study found that when looking at the alternating scan strategy, there was a larger area of the scan region under compressive transverse stress than when using the unidirectional scan strategy [10]. The alternating scan had lower magnitude of stresses and plastic strain, but these differences were overall minimal between the two scan strategies. A separate study that looked at different scanning patterns and the subsequent deflection and stresses on the build part found that an XY alternating scanning strategy (one layer is scanned in the x-direction only, while the subsequent layer is scanned in the y-direction) gave the most uniform distribution of stresses for a multi-directional scanned part [18].

Other studies have also looked at the effects of scanning direction on part distortion in DMLS; multiple studies have shown that the longer the scanning vector along each layer, the higher the distortion on the final build part [9], [11], [15], [19]. Shorter scan vectors may be more favorable due to shorter time between layer deposition as well as the heat from previous layers having less time to cool between scans of layers [15]. As the number of layers increases, the deflection in the longitudinal (direction across long edge of build part) scan direction increases but the deflection across the part transversely (direction across the short edge of build part) decreases [9], [18]. One study points out that while the magnitude of stresses does not differ greatly between unidirectional and alternating directional raster scans, the distribution of stress and plastic strain does vary due to differing thermal histories between the two scanning strategies; the alternating scan strategy showed reduced temperature gradients for each scanned vector [10]. This study uses this evidence to show that part stresses in DLMS can be mitigated by using an optimal scan strategy [10].

2.4 Heat Transfer Models

Heat transfer models are crucial in understanding the DMLS process and how the temperature distributions form on the build part throughout the AM process. However, the heat transfer process in DMLS is complex and solving for the temperature field is tedious. Some of the heat transfer model complexity comes from the considerations of the powder phase change to solid, as well as the various boundary conditions that the process exhibits [20]. Several papers have looked at how to accurately model the complex heat transfer processes that occur in DMLS. The most common results, as explained below, use the Fourier heat conduction theory and consider conduction, convection, and radiation heat transfer processes on the build part during build time and cooling [20].

One study looks at the 3D steady state conduction in DMLS and similar processes; it considers a moving rectangular heat source and surface cooling [21]. Points of interest to consider in this heat transfer model are the boundary conditions and governing equations. This heat transfer model follows the steady state three-dimensional heat advection-diffusion model, taking into account the energy-transport phenomena, as shown in Equation 2.1, where T is the Temperature, V is the velocity of the moving heat source and α is the diffusivity of the material [21].

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} - \frac{V}{\alpha}\frac{\partial T}{\partial x} = 0 \qquad 2.1$$

The study considers a block, the top surface of which experiences convection while the sides are considered insulated. Various types of integral transforms, including frequential integral transforms which account for the periodicity of the heating conditions, are completed to find an analytical solution for this specific case.

While the analytical solution found in [21] is not suited for this research (as insulated sides and convection are considered), this study shows that using the heat conduction equation coupled with boundary conditions and incident heat flux can be formulated [21].

Another study looked at numerically modeling the temperature history, as it is difficult to accurately measure the rapidly changing temperatures of the DMLS process experimentally [22]. This paper looked at decoupling the overall model by first solving the heat transfer problem, and then using that solution as input for the thermomechanical problem. The time-dependent heat conduction equation was used.

$$\rho c_p \frac{dT}{dt} = -div(q) + Q \qquad 2.2$$

Where ρ is the material density, c_p is the specific heat, T is temperature, t is time, and Q is the heat source. In this equation, q is the heat flux vector as defined by Fourier's Law $q=-k\nabla T$ where k is the thermal conductivity. The boundary conditions considered for the modelled block were insulation at the side walls, and fixed temperature on the bottom wall, and convective and radiative heat transfer on the top surface. This heat transfer model was solved and the temperature field used in the thermomechanical simulation, which was able to produce predicted stresses and deformations for the DMLS process [23]. Other simulation studies considered the same boundary conditions, as well as using Fourier's law or energy conservation to solve the governing equation for the temperature distribution [23], [24]. These commonalities among general heat conduction equation and boundary conditions are shown through these studies to be applicable to the simulation this study will look at. More details of heat flux and laser equation modeling from previous studies are given in the next section, Section 2.5: Simulation Models.

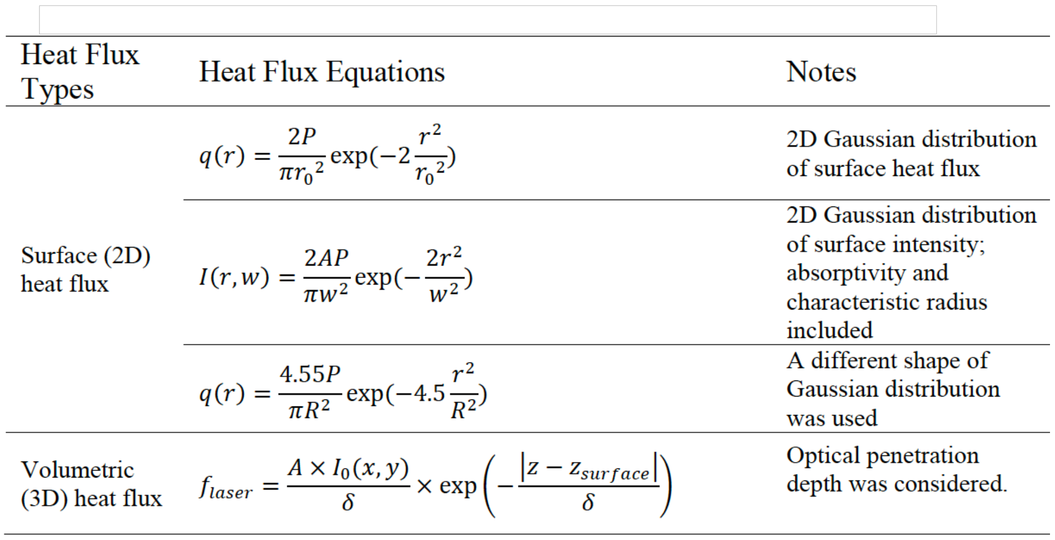
2.5 Simulation Models

Since DMLS is a monetarily costly process to run experiments with, many of the previous studies done looking at the heat and mechanical outcome of the process have been completed through simulations. How these simulation models were set-up is important to understand, as these simulations can be costly in computational time;

depending on the level of detail in mesh and number of layers simulated, a DMLS simulation has been claimed to take anywhere from multiple days to months [11], [25], [26]. This can make it challenging to develop an accurate model with to-scale parameters without sacrificing part or process information that can be crucial to the part response and outcome in DMLS [26]. These models can be implemented using ANSYS, Abaqus with a DFLUX subroutine, or a code created for a specific study [19], [27]–[29].

The DMLS process is considered a coupled process between the thermal responses and the mechanical responses; this coupling is considered weak though, as the thermal history has a significant influence on the mechanical response of the part, but the reverse is not true [11], [12], [22], [28], [30]. Much of the literature states the most important aspects that a thorough DMLS simulation should have. These aspects include a model for the laser thermal input; temperature-dependent material properties to account for heating and cooling on different stages of the process; the layering build-up process and its heating and cooling effects on the overall part; and the mechanical responses to the part from the laser-powder interactions and how the laser moves [3], [28], [30]. Models, such as the one completed in [11], replace the thermo-mechanical coupled simulation with a static mechanical one, but this can sacrifice some of the necessary physics to get accurate results.

When accounting for the thermal input, the laser heat flux incident on the simulated build part must be accounted for; the sintering time in DMLS is exceptionally short, with heat flux coming from the moving laser to each particle for only 1 milliseconds to 0.1 seconds [30] and generating a rapidly changing temperature history [22]. To model the moving laser as a heat source and its incident heat flux, many past simulations have utilized the Gaussian Heat Flux Model [3], [10], [23], [28], [31]–[33]. Table 2.7 outlines a few of the heat flux types and the corresponding Gaussian distributions for heat flux.



Heat Flux Types	Heat Flux Equations	Notes
Surface (2D) heat flux	$q(r) = \frac{2P}{\pi {r_0}^2} \exp(-2\frac{r^2}{{r_0}^2})$	2D Gaussian distribution of surface heat flux
	$I(r,w) = \frac{2AP}{\pi w^2} \exp(-\frac{2r^2}{w^2})$	2D Gaussian distribution of surface intensity; absorptivity and characteristic radius included
	$q(r) = \frac{4.55P}{\pi R^2} \exp(-4.5\frac{r^2}{R^2})$	A different shape of Gaussian distribution was used
Volumetric (3D) heat flux	$f_{laser} = \frac{A \times I_0(x,y)}{\delta} \times \exp\left(-\frac{\lvert z - z_{surface}\rvert}{\delta}\right)$	Optical penetration depth was considered.

Figure 2. 4 Table of Heat Flux Models from [28]

These equations calculate the average heat flux incident on the part as the laser moves across each element [28]. Some simulations have applied the volumetric Gaussian heat source, which accounts for volumetric energy density [10], [34]. The Gaussian heat flux model is preferred over other laser heat flux models, such as Rosenthal's evolution of temperature field [20], due to the fact that the Gaussian model allows for the laser spot size, element size, and thickness of each layer to be of the same magnitude [31]. In one DMLS simulation study, the researchers were able to implement a line heat source over a Gaussian point heat source to try to reduce the computational time [23]. Some studies used dynamic meshing – with a finer mesh under the laser spot and coarser mesh elsewhere – in order to reduce the computational time of calculating the moving heat flux [25], [34]. Simpler meshing under the laser heat source considers a consistent mesh with the laser spot size the size of one element [32].

When considering the heat transfer modes occurring in the DMLS process and how to quantify the cooling of the simulated part, most previous studies reviewed implemented time-dependent Fourier's law of heat conduction discussed in Section 2.4: Heat Transfer Models [22], [25], [31], [32], [35]. One aspect important to note that is specific to the simulations is the boundary conditions considered. One boundary condition considered in some simulations is heat loss by radiation from the part [22], [23], [25], [34], but one study found that radiation influence can be ignored since its influence on the overall cooling is much smaller compared to the other heat loss mechanisms of convection and conduction [25]. Convective heat transfer between the part and the ambient air is another boundary condition many previous simulation studies considered [23], [25], [27], [34]. The process is often completed in an inert and still environment to help with preventing reactions such as oxidations [7], and convection may be ignored under this assumption. Conduction is also considered between the layers of sintered metals with insulated sides to simulate the part in contact with surrounding powders [10], [23], [25]. To account for the substrate underneath the part being built, studies have either treated the bottom as insulated [22], [32], or as a heat sink with a prescribed temperature on the bottom most layer of the simulation [12], [23]. When considering all of the heat transfer modes, the previous simulations have emphasized the importance of considering the temperature-dependent process and the different states of metal throughout the process [10], [28].

2.6 Genetic Algorithms and Optimization

The last topic looked at in the literature review for this project was in optimization and Genetic Algorithms. Due to the non-linear and complex nature of the problem, a genetic algorithm would supply the means to find an optimal scanning pattern in DMLS by means of a simulation and without sacrificing key elements of the optimization problem.

Genetic Algorithms come from Darwin's theory of evolution and the "survival of the fittest" and can be employed to solve both linear and nonlinear problems [36]. Genetic algorithms include a starting population, represented as "genes," a crossover method, and a mutations method [36], [37]. Literature shows that it is important to start with a well distributed population in order to broaden the search space, avoid local optima, and obtain better results [36], [37]. The number of generations for each run of the GA increases with the population size [37]. After the initial population is generated, with either binary or valued genes, each parent selected in the selection process undergoes a crossover; in this action, the genes from each parent are swapped at different sections to create new, unique children [36]–[38].

The crossover technique is of importance in this literature review, as an order-based crossover is needed to generate scanning paths as no path can be repeated. In an order-based crossover, several random positions are selected in a parent string similar to a k-point crossover, but the selected positions of genes in parent 1 are imposed onto those of parent 2 and vice versa to keep the order of genes non-repeating for the children [37]–[39]. A sample order-based crossover is below, adapted from an example in [38]. P1 and P2 are the parents, C1 and C2 are the children, and the selected crossover point is marked by double lines.

P1 = (1 2 3 4 || 6 9 8 5 7)

P2 = (2 1 9 8 || 5 6 3 7 4)

After the crossover point is selected for both parents, the genes to the right of the crossover point in one parent are identified in the other parent. In P1, the genes to the left of the crossover point are 1, 2, 3 and 4. These are located in parent P2, and the remaining genes are marked with a star. Similarly, the genes to the left of the crossover point in parent P2 are 1, 2, 9 and 8. These are located in Parent P1, and the remainder are replaced by stars.

P1 = (1 2 * * * 9 8 * *)

P2 = (2 1 * * * * 3 * 4)

The genes not selected with a * in either parent keep their position in the respective child, and the rest of the genes of the child are filled in with the genes from the other parent. Thus, the obtained children would be:

C1 = (**1 2** 5 6 3 **9 8** 7 4)

C2 = (**2 1** 6 9 8 5 **3** 7 **4**)

Another crossover aspect found to be important to this project found is a uniform crossover, where a random real number determines which genes each child will get from the two parents selected [38]. This ensures the gene selection is uniform across both parents.

Order-based crossovers are found commonly in Travelling Salesmen Problems, which uses a specific type of Genetic Algorithm. TSP problems find the shortest route which traverses every city in a path only once [37], [39]. In the past TSP problems investigated, a two-point order-based crossover was used; two-point crossover increases computational time, but helps avoid desired characteristics from a parent not being passed down to a generated child [37]. A two-point crossover selects two points along the genes of the parents in which to crossover the genes at; child 1 inherits the head and tail genes from Parent 1, and the center genes from Parent 2, and vice-versa for child 2 [37]. An example of a two-point crossover is shown in Figure 2.9, where the double lines between genes indicate the two crossover points selected. It should be noted that the location of the double lines for the crossover is selected at random.

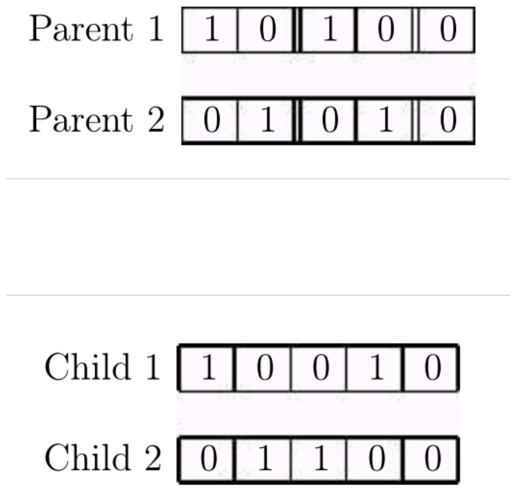


Figure 2. 5 Two-point crossover example [37]

Applying this type of GA crossover and knowing its success in these past studies helps show its usefulness moving forward with finding an optimal scanning pattern in DMLS.

CHAPTER 3

DMLS SIMULATION

3.1 Simulation Introduction

Based on the previously reviewed literature, this study aims to assess the issues of the high temperature gradients and subsequent deformations through optimizing the scan pattern the laser takes in a DMLS build [9], [10], [18]. Using the previous analysis and results on residual stresses induced on build parts in DMLS, heat transfer models, and simulation set-ups as detailed in Chapter 2, the first part of this research looks at the development of a simulation of the DMLS process. The simulation aims to simulate the moving heat source from the laser, the layer build-up process, and the effects of heat transfer on the simulated build part. Using the generated simulation, an optimization technique is then be applied in the second part in order to find a scanning pattern that reduces thermal gradients.

3.2 Simulation Overview

In order to measure the thermal gradients and distortions from the DMLS process in a research setting, a simulation was created using MATLAB and Abaqus/Standard and Abaqus/CAE. The MATLAB code was originally generated by post-doctoral researchers Jennifer Snipes and S. Ramaswami; the purpose of the code is to generate the necessary input files that can be submitted to Abaqus/Standard to run the simulation. The results from these simulation runs are then designed to be read and processed by Abaqus/CAE.

A schematic of the DMLS process is shown in Figure 3.1; this figure helps to show how the powder bed, solidified metal, and laser scanner all relate. This figure may be useful to refer to when considering the heat transfer methods and boundary conditions discussed in later sections of this chapter.

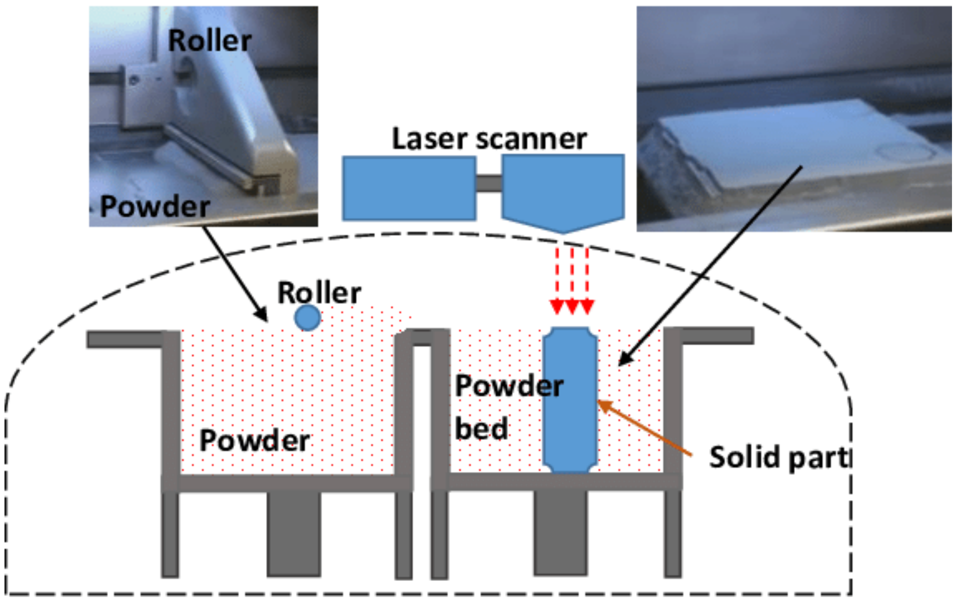


Figure 3. 1 Schematic of DMLS Process [40]

This chapter describes the parameters, the geometry, and the heat transfer, and explains how the input file is written using the MATLAB code. In addition, preliminary results comparing non-optimized scanning patterns are given, together with possible routes to defining the temperature gradient in the process.

3.3 Parameter Description

3.3.1 Process and Laser Parameters

The process and material parameters used in the simulation are described here. Below in Table 3.1 and 3.2 are the process parameters and laser parameters, respectively, that were set for the simulation. Since the main goal is to develop a qualitative approach to the problem, these values were later adjusted in an effort to exaggerate the thermal gradients developed in the process, and so to arrive at an optimizable metric.

Bed Temperature	298 K
Initial Temperature	298 K
T_{inf}	298 K

Table 3. 1 Process Parameters

Laser Power	2500 W
Scan Speed	0.05 m/sec
Laser Radius	.0025 m

Table 3. 2 Laser Parameters

3.3.2 Material Properties

The material properties chosen for the simulation are those of Titanium Ti-6Al-4V (Grade 5), Annealed. The material properties are shown in Table 3.3 [41].

Density	4430 kg/m^3
Modulus of Elasticity	113.8E9 Pa
Poisson's Ratio	0.342
Coefficient of Expansion from 20 to 950 C	9.7E-6 m/m-C

Table 3. 3 Material Properties for Ti-6Al-4V [41]

Since the DMLS sinters powdered metal, which then cools into solid metal, the thermal conductivity varies from powder state to the solid metal state. Within both of these states, the variation of thermal conductivity and specific heat with the part temperature must be accounted for. The thermal conductivity values are shown in Table 3.4 and the specific heat values are shown in Table 3.5 [28].

Temperature [K]	Thermal Conductivity [W/mK]
Powder	
293	0.2
1879	19.4
1928	28.3
Bulk	
299.85	7.20
373.00	8.15
473.00	9.44
773.00	13.32
1149.85	18.20
1273.00	19.79
1773.00	26.26
1928.00	28.27
2399.00	37.00
2699.85	42.00

Table 3. 4 Thermal Conductivities of Ti-6Al-4V [28]

Temperature [K]	Specific Heat [J/kgK]
293	580
478	620
698	670
923	760

1143	930
1273	936
1473	1016
1673	1095
1928	1126

Table 3. 5 Specific Heat of Ti-6Al-4V [28]

In order to correctly model heat transfer phenomena on the simulated build part, latent heat is also included in this simulation. Using the solidus and liquidus temperatures shown in the conductivity data, 1878K and 1928K respectively, the latent heat used in this model is 286 W/mK [42]. All material properties are taken from the input file and used by Abaqus to complete the coupled temperature-displacement analysis.

3.4 Geometry Description

The geometry chosen for this simulation is a block with a fin attachment, as shown in Figures 3.2 and 3.3.

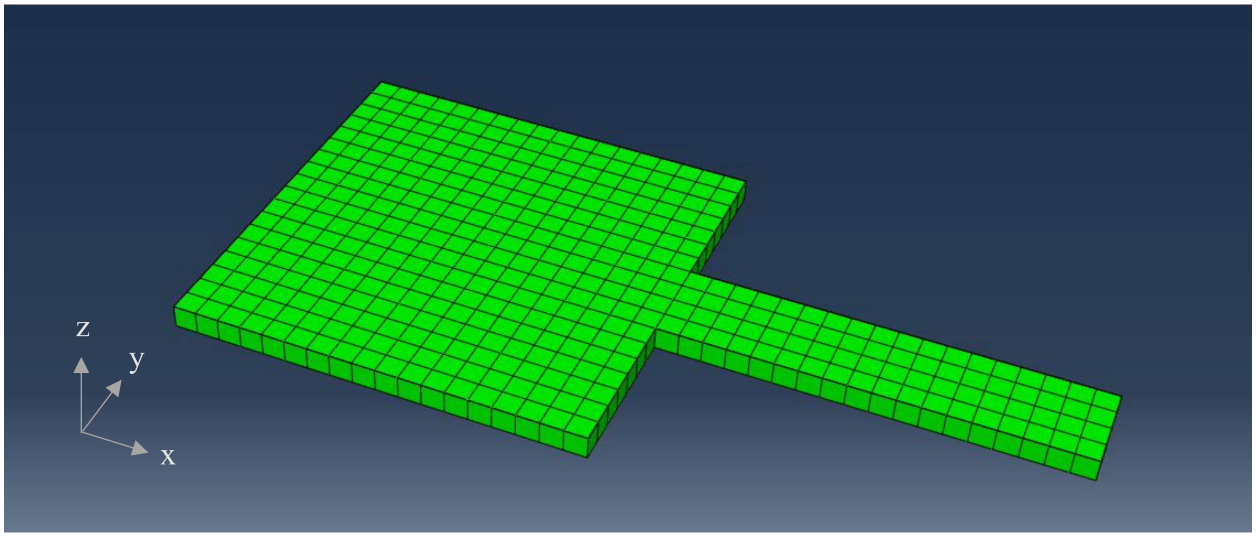


Figure 3. 2 One Layer Fin/Block Geometry from Abaqus

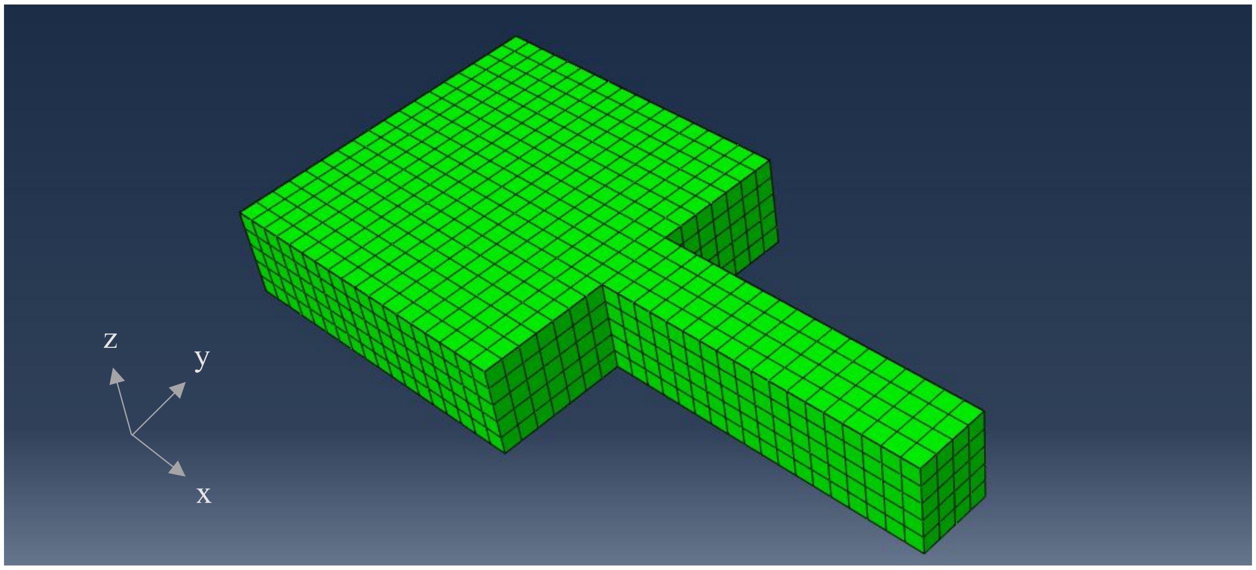


Figure 3. 3 Multi-Layer Fin/Block Geometry from Abaqus

The intention behind setting the above geometry as the one used in initial simulation testing, was to look at both the effects of heating and cooling cycles on a square and deformation effects on the edges, as well as to potentially see the issues in deformations in thinner features within some DMLS build parts.

Each element within the part is given the same dimensions in the x-,y-, and z-directions. This size can be changed within the MATLAB script to specify a coarser or finer mesh. When changing the element size, the number of total elements created is also changed. The simulation converged on using 5mm as the element size in the pre-optimization analysis and final optimization tests. More discussion on mesh selection is included in Section 3.5.1.

As stated, the block with fin attachment is set to have a specific number of elements along each edge. These numbers for a 5mm element size are shown on a labeled block with fin attachment shown in Figure 3.4 below.

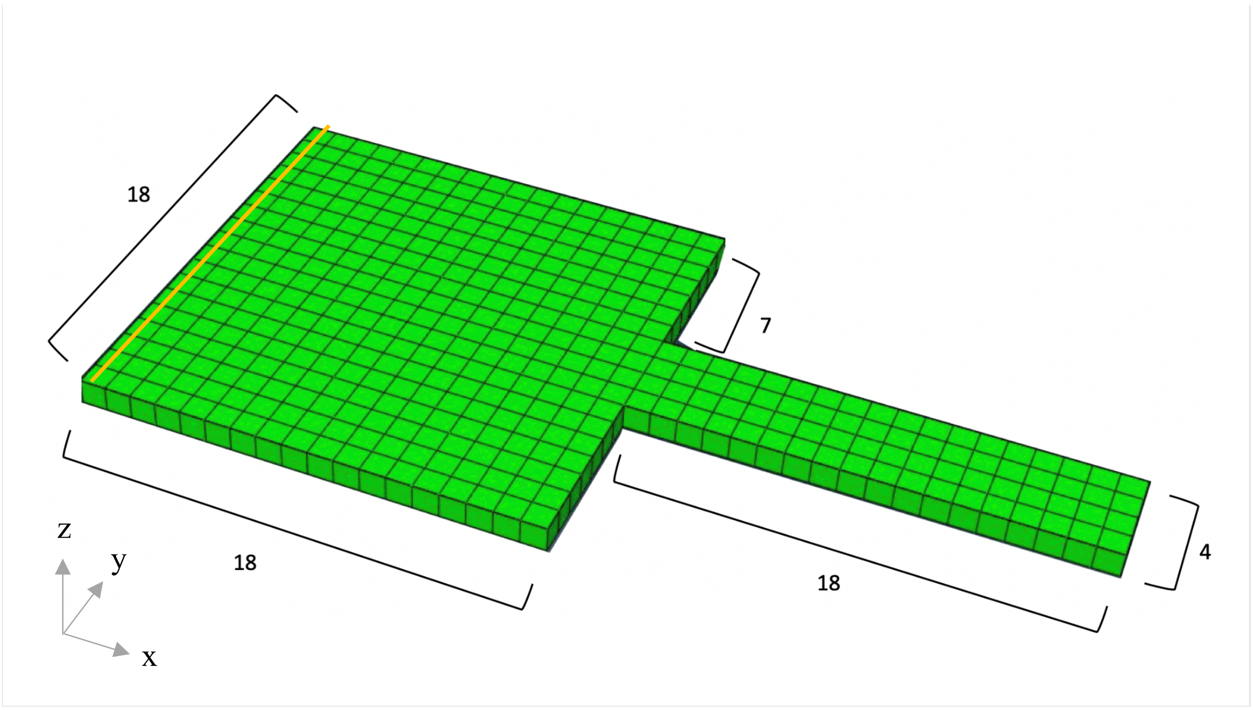


Figure 3. 4 Geometry with Element Numbers

Below is the section of MATLAB code written that defines this specific block and fin geometry; dimensions are in meters:

```
% Finer mesh

mesh.del_x = 5e-3;
mesh.del_y = 5e-3;

mesh.del_z = 5e-3;




% So, when it comes to the interior block scan, I want to have 2^4
element
% rows in y direction.  So that is 16 interior + 2 that will be in
% perimeter. (Per Jennifer Snipes)


mesh.num_elem_z = 9;

mesh.num_elem_y = 18;

mesh.num_elem_x_with_fin = 36;

mesh.num_elem_x_without_fin = 18;


mesh.elem_rows = 1 : mesh.num_elem_y;

mesh.elem_rows_with_fin = 8:11;


mesh.elem_rows_without_fin = ...
    setdiff(mesh.elem_rows, mesh.elem_rows_with_fin);


mesh.num_nodes_z = mesh.num_elem_z + 1;

mesh.num_nodes_y = mesh.num_elem_y + 1;

mesh.num_nodes_x_with_fin = mesh.num_elem_x_with_fin + 1;

mesh.num_nodes_x_without_fin = mesh.num_elem_x_without_fin + 1;
```

From this section of code, it can be seen that the number of elements in each direction is defined and then used to calculate the number of element rows both on and off the fin are calculated, as well as the number of nodes per layer in each direction.

Each row of elements starting from the back end (left side) of the block are considered scanning vectors, with each element in the highlighted row in Figure 3.3 being the starting positions of each vector. More details on the scanning vectors in relation to the chosen geometry are provided in Chapter 4.

3.5 Heat Transfer: One Element

3.5.1 Description of an element and boundary conditions

The model in this simulation consists of elements having a mesh size of 5 x 5 x 5 mm. The order of magnitude of this size of elements is comparable to past numerical simulations [3], [17], [27], [43]. While typically mesh refinement is done until results do not change, mesh refinement in this simulation increases the computational cost greatly; a five-layer scan with 5mm element size takes about 10 hours, and finer meshes only increased this time. Since this study requires numerous runs in the Genetic Algorithm (discussed in Chapter 4), it was important to keep the computational costs down to allow for more efficient optimization. The element type chosen in Abaqus/Standard is C3D8RT, which is an 8-node thermally coupled element with trilinear displacement and temperature outputs [44]. Every element in the simulation is initially considered powder, but changes to solid once the simulated laser has scanned the element. All elements properties are considered isotropic in all properties as mentioned in Section 3.2.2. Elastic deformation is accounted for and coupled temperature-displacement is found on each element at each Abaqus time step.

Boundary conditions are set for both exterior and interior elements. The first condition is that the bottom surface of the first layer of elements has a fixed temperature; this condition helps simulate the conduction between the layers of the build part and the substrate, which acts as a heat sink during the scanning and cooling periods. Conduction occurs between the elements above the first layer, as well. Radiation is considered negligible in this simulation because of the small size of the melt pool, which is assumed to be the same size as the laser beam [17], and the very quick cooling once the laser moves away. Also ignored is convection; due to the small size of the molten pool, the heat loss by convection from the top surface is small compared to the heat loss by conduction from the part. The simulation also assumes an inert chamber [7], with no gas motion. This assumption of no gas motion is important to consider when addressing convection, as without the movement of the surrounding air, significant convection cannot happen between the build part and the chamber air.

A statistical analysis was done to evaluate the significance of convection on the temperature profile of a build part in the DMLS process. Two single-layer scans were completed in Abaqus/CAE, meaning no field variables were accounted for, so the thermal conductivity values remained that of powder for the entire scan. One single-layer scan considered convection on the build part surfaces, and the other did not. The average NT11 (where NT11 is the nodal temperatures across the part's surface) values were found for both runs, and an independent sample t-test was done to evaluate if there was a significant difference between the two runs [45]. The results are shown below in Table

3.6. The degrees of freedom chosen was 202, as there were 202 time-steps in the scan. The null hypothesis taken was μ_1-μ_2 = 0, or that there is no difference in the mean values of the NT11 data from both runs. The confidence level chosen was 90%, or α=.01.

	Single Layer Scan Without Convection	Single Layer Scan With Convection
Avg. NT11	909.6471 K	903.2841 K
T Test		
x_1	909.6471	N = 203
x_2	903.2841	SE = 24,9269
s_1	253.1272	x_1-x_2 =6.36301
s_2	249.1209	t = .2552

Table 3. 6 T-Test Results for Comparing Convection vs No Convection in Simulation One-Layer Scan

In the table above, x_1 and x_2 are the mean values from each population, s_1 and s_2 are the standard deviations from both populations, and SE is the standard error for the differences in samples. All of these are used for calculating the test statistic for the t-test. As can be seen from this table, the t value found was small, signaling that the two populations are very similar. The critical value of t found from a t-distribution graph with n=202, $\alpha/2$ = .005 (use for a two-tailed test) was 2.576 [45]. The test t value found from the two means was .255. Since the test t value is less than the critical value of t, it can be said at a 90% confidence level that the difference between mean NT11 values of the two populations – the two scans, one with and one without convection – are not statistically different. Therefore, this helps support the assumption made for this simulation of no convective heat transfer being included on the simulated build part.

The sides of the exterior elements also experience conduction, to mimic the fact that the newly solidified powder on the perimeter of the build part is still in contact with the rest of the powder in the bed, with which conduction occurs. Volume shrinkage of the elements is not considered in this model. With these assumptions, the numerical model of the DMLS process is simplified, yet sufficient to evaluate the hypothesis of this research.

The last boundary conditions set on the elements is a displacement boundary condition. A corner node on the bottom face is set to have zero displacement. Its nearest neighboring nodes in the element along the x-,y-, and z-directions are also fixed in their respective directions. This idea is taken from a past simulation model with the goal of preventing rotation and translation [17]. Specifics of the laser incident on the element are detailed in the next sub-section, 3.5.2: Description of Laser.

A schematic of a boundary and interior element is shown below in Figure 3.5.

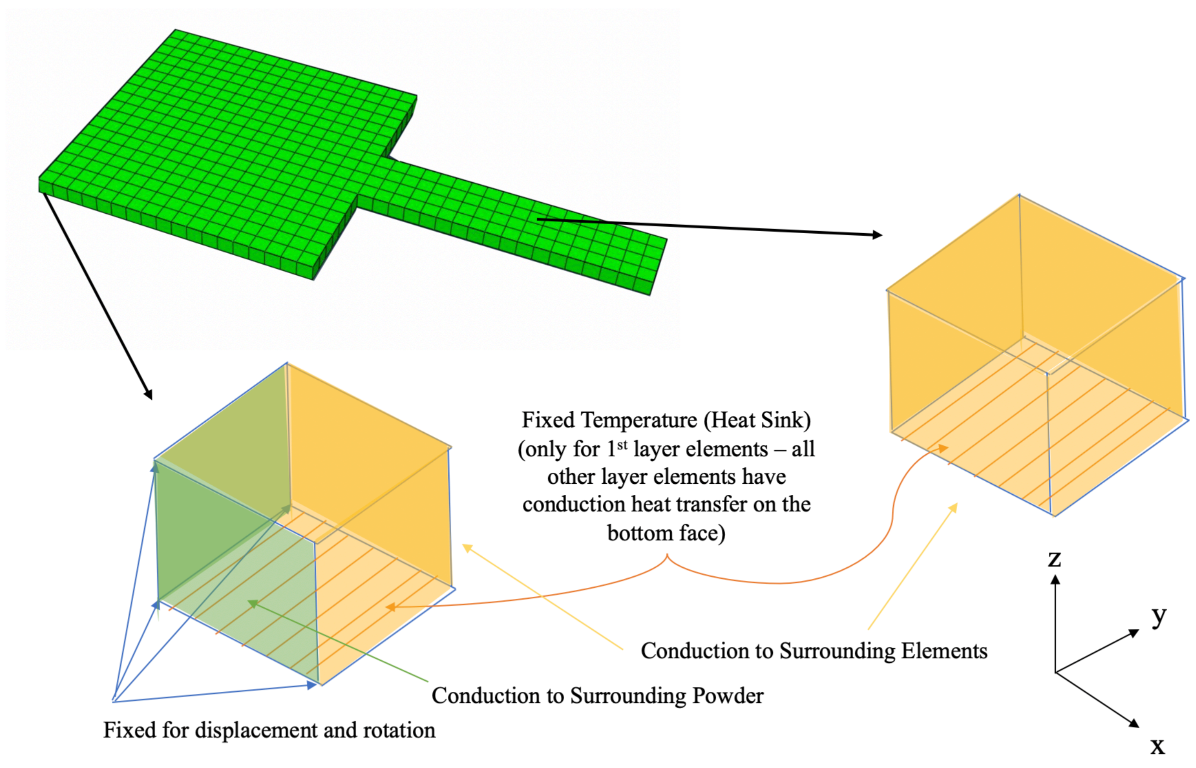

Fixed Temperature (Heat Sink) (only for 1st layer elements – all other layer elements have conduction heat transfer on the bottom face)
Conduction to Surrounding Elements
Conduction to Surrounding Powder
Fixed for displacement and rotation
z
y
x

Figure 3. 5 Schematic of Elements with Boundary Conditions

Below, the section of MATLAB code defining the initial and boundary conditions, respectively, for bottom layer elements respectively is shown:

```
fprintf(fid_aba, '%s \r\n','****');
fprintf(fid_aba, '%s \r\n','**');
fprintf(fid_aba, '%s \r\n','*INITIAL CONDITION, TYPE=TEMPERATURE');
fprintf(fid_aba, '%s%s \r\n','Nset_All, ', num2str(T_init));
fprintf(fid_aba, '%s \r\n','**');
% Field variable = 0 for powder, 1 for bulk
fprintf(fid_aba, '%s \r\n','*INITIAL CONDITION, TYPE=FIELD');
fprintf(fid_aba, '%s%s \r\n','Nset_All,',num2str(FV.powder));
fprintf(fid_aba, '%s \r\n','**');
fprintf(fid_aba, '%s \r\n','****');
```

```
    if (current_layer == 1)
        % Set bottom of first layer to bed temperature
        fprintf(fid_aba, '%s \r\n', '*BOUNDARY');
        fprintf(fid_aba, '%s,%5i,%5i,%10i \r\n', mesh.Nset_bottom.name,
11, 11, T_inf);
        fprintf(fid_aba, '%s \r\n', '*BOUNDARY, TYPE=DISPLACEMENT');
        fprintf(fid_aba, '%5i,%5i,%5i,%5i \r\n', corner_node, 1, 3, 0);
        fprintf(fid_aba, '%5i,%5i,%5i,%5i \r\n', ...
            corner_node_neighbor_x, 2, 3, 0);
        fprintf(fid_aba, '%5i,%5i,%5i,%5i \r\n', ...
            corner_node_neighbor_y, 1, 1, 0);
        fprintf(fid_aba, '%5i,%5i,%5i,%5i \r\n', ...
            corner_node_neighbor_y, 3, 3, 0);

        fprintf(fid_aba, '%s \r\n','*OUTPUT, FIELD, NUMBER
INTERVAL=1');
        fprintf(fid_aba, '%s \r\n','*ELEMENT OUTPUT');
        fprintf(fid_aba, '%s \r\n','EVOL');
    end
```

To calculate the heat flux for each element, the DFLUX subroutine was used in Abaqus/Standard. This user subroutine is used “to apply distributed fluxes in fully coupled thermal-stress analysis [46].” This simulation assumes a non-uniform heat flux distribution, and calculates the distributed flux using the element number and face, the type of distributed flux, and the reference flux magnitude [46], [47]. In this case, the reference flux magnitude was set to 1W/m^2; the flux calculated for each element is only multiplied by one unit of heat flux per unit surface area. Since this simulation uses non-uniform heat flux distributions, the heat flux data for the DFLUX user subroutine comes from the amplitude data calculated at each time step. Using a reference magnitude of one allows for the heat flux to come from the amplitude data instead of being a set value throughout the simulation, such as in uniform heat flux instances [47]. In this case, the amplitude data consists of the magnitudes of heat flux incident on the part and its position along the part for each time step of the simulation.

3.5.2 Description of Laser

A key part to having a working simulation of DMLS is having the model of the laser accurately apply heat flux to each element while moving. As laid out in Section 3.2.1, the laser spot size is set to be the same size as the element [32]; in this case, the radius of the laser is defined as half the width of each square element. The laser definition in this simulation assumes the part to have an even number of rows and assumes that laser does not finish scanning a layer on one of the scanning vectors that makes up the fin.

The laser beam approximation in this simulation follows a Gaussian beam profile; these are used in many models as laid out in Chapter 2 [3], [10], [20], [23], [25], [31]–[33]. Equations 3.1 and 3.2 model from [25] the Gaussian beam profile and are used in the simulation to solve for the heat flux at each point in the part and the average heat flux, respectively.

$$q(r) = \frac{2P}{\pi r_0^2} e^{-\frac{2r^2}{r_0^2}}$$

3. 1

$$q_m = \frac{1}{\pi r_0^2}\int_0^{r_0} q(2\pi r)dr = \frac{0.865P}{\pi r_0^2}$$

3. 2

Where:

- P = laser power
- r_0 = laser beam radius
- r = radial distance of a point from the center of the laser beam center

The radial distance of a point on the build part from the laser beam center was calculated using Cartesian distance equations and the points along the top surface in relation to the laser beam radius. As the laser hits an element, the heat flux incident on that element accounts for all surrounding points on the top surface of the part. The laser beam's radius is assumed to be half the width of an element; this allows for the Gaussian beam heat flux distribution calculated to cover each element. While the Gaussian beam distribution covers the area of a circle and the elements are considered square, Equations 3.1-3.2 calculate the distribution such that the heat flux incident can be calculated on each element and surrounding area. The average heat flux in Equation 3.2 is used to calculate the heat flux incident on each element using the DFLUX subroutine, and the amplitudes are recorded and submitted with the input file on the simulation when run in Abaqus/Standard.

3.5.3 Energy Balance on an Element

From the boundary conditions and heat flux equations stated in Sections 3.4.1 and 3.4.2, the governing equations for the energy balance and heat transfer on an element are deduced; these equations pair with the schematic laid out in Figure 3.4.

The governing heat transfer equation used to approximate the heat transfer on an element – and so the entire build part – comes from Fourier's three- dimensional time-dependent heat conduction equation, shown in Equation 3.3 [21]. This form of the equation takes into account a moving heat source and surface cooling [21]. The boundary conditions applied are laid out in Section 3.4.1, and the initial conditions are shown in Equations 3.5-3.10. As shown in Chapter 2, this equation is used often in modelling the heat transfer phenomena that occur during the DMLS process.

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} - \frac{V}{\alpha}\frac{\partial T}{\partial x} = 0$$

3. 3

Where:

- T = Temperature
- V = Velocity of laser
- α = Thermal diffusivity of the material

Equation 3.3 indicates that diffusivity plays an important role in the heat diffusion when the velocity of the moving heat source is greater than zero. As the velocity tends to infinity, though, the temperature becomes independent of the direction in which the heat source is moving [21]. It should also be noted that this equation is the same as the standard unsteady state three-dimensional heat conduction equation; if V is replaced with $\frac{dx}{dt}$, then the equation on takes on the traditional form [48], as shown below in Equation 3.4.

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial x^2} = \frac{1}{\propto}\frac{\partial T}{\partial t} \qquad 3.4$$

Boundary Conditions:

Boundary Element:

$$x, y, z|_{corner\ bottom\ element\ nodes} = 0 \qquad 3.5$$

$$T(z = 0) = T_{bed} \qquad 3.6$$

$$q = -k\nabla T \qquad 3.7$$

Internal Element:

$$T(z = 0) = T_{bed} \qquad 3.8$$

$$q = -k\nabla T \qquad 3.9$$

Initial Condition for all elements: $T(t = 0, x, y, z) = T_{inf}$ 3.10

Where:

- q = heat flux
- k = thermal conductivity
- T = Temperature
- t = time

The DMLS process consists of unsteady heat transfer modes as the laser moves across the build part. However, an energy balance can be done on a single element for each time step. Figure 3.5 shows the energy balance on both a boundary element an interior element while the laser is incident on it. The heat loss to substrate and heat conduction to surrounding elements corresponds to the equations 3.4 – 3.10 respectively, for the boundary conditions above.

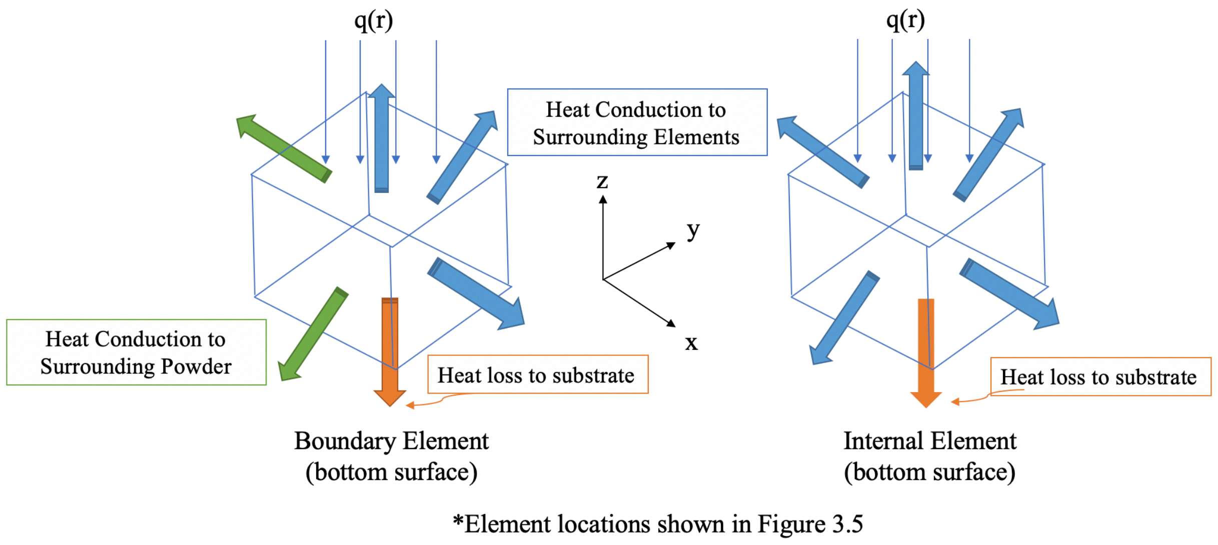

q(r)
q(r)
Heat Conduction to Surrounding Elements
z
y
x
Heat Conduction to Surrounding Powder
Heat loss to substrate
Heat loss to substrate
Boundary Element (bottom surface)
Internal Element (bottom surface)
*Element locations shown in Figure 3.5

Figure 3. 6 Energy Balance on Boundary and Internal Elements

With the heat flux equations in 3.4.2, the heat transfer equations above, and the energy balance on a single element, the heat transfer and temperature history of an element in the build part in DMLS is sufficiently modelled in the simulation.

3.6 MATLAB Code Description/Input File

The MATLAB code written for this DMLS simulation begins with initializing process parameters and generating the mesh. In generating the mesh, the number of layers, the size of the elements, and the number of elements is dictated, and the element and node sets are generated for the set number of layers. The code then sets the laser parameters and time parameters that are used throughout the code to determine the durations of the various stages of the simulation; the code writes the boundary conditions used in this simulation as well, as described in Section 3.4. Then, the perimeter of the part is “scanned” in the code, with the heat fluxes and amplitudes being calculated and recorded in a file to submit to Abaqus. The interior is then scanned; depending on whether a raster or random scan is required, the code generates the vector scanning order and as with the perimeter scan, the heat fluxes and amplitudes are calculated and appended to the files being submitted to Abaqus.

For this simulation, raster scans follow adjacent vector scans, while a random scan follows an unordered array of vector scans. A schematic of the scanning patterns is included below in Figure 3.4. The numbers above the scanning arrows correspond to the track number of the scan in the sample of four scanning vectors. This is the only difference in code between the raster and random scans that are run.

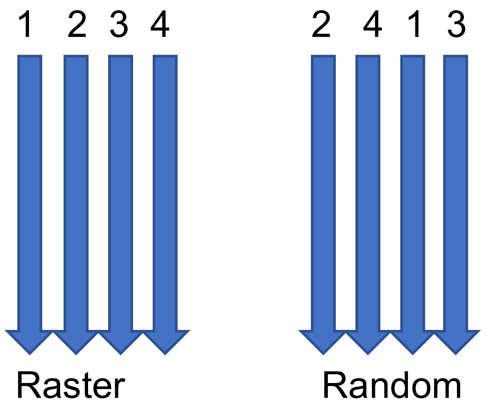


Figure 3. 7 Schematic of Raster vs. Sample Random Scan

After the heat flux data is calculated and recorded for the simulated scanning, the MATLAB code then writes the input file that is submitted into Abaqus. The heat flux data that MATLAB calculates is used as the input heat flux loads for the temperature-displacement analysis completed in Abaqus.

The MATLAB code begins writing the input file by including all the element and node files, then writing the material properties and initial conditions. The rest of the input file is written in four main steps, with heat flux and amplitude data being added when a step involves laser scanning. The steps written to the input file are described below.

Step 1: Material Removal

In this step, all material from the simulated part is removed. At the beginning of the input file, all element and node sets are included in the part. However, the simulation should only add the requested number of layers, and do so one at a time after the previous layer has been scanned. This step "removes" all the material that adding all the elements and nodes would have created in Abaqus from the start of the input file, so that the subsequent steps can dictate which element and node layer sets should be added and evaluated.

Step 2: Adding Layer

In this step, a layer is added to the build part in the simulation. The layer is added by adding the elements and node sets from the specific layer as determined in the mesh generation. If a layer is being added beyond the first layer, this step ensures the flux on the top surface for the previous layer is set to zero and the field variables are set "bulk," so all previous layers are considered a solid part having bulk properties.

Step 3: Scanning Layer

In in this step of the input file that the top layer is heated according to the heat flux data calculated and the scanning pattern dictated earlier in the code. A coupled temperature-displacement analysis is set for Abaqus to complete. Once scanning is complete, the heat flux is set to zero once scanning is complete, and the field variables are changed to give the recent layers bulk properties. Steps 2 and 3 are repeated until the number of layers dictated is reached.

Step 4: Cooling Part

The last step written to the input file is the cooling step, in which time is added to allow the part to cool, and a coupled temperature-displacement analysis is set for Abaqus to complete. All output files are generated from Abaqus for post-processing of the simulation.

3.7 Defining the Temperature Gradient

In order to address and minimize the thermal gradients and deformations on the build part, three possible objective functions for the optimization were evaluated based on the preliminary results found in section 3.8: Preliminary Results from Raster and Random Scans. These different objective functions represent ways to characterize and compare the thermal gradients between the raster scan and random scan by utilizing the nodal temperature outputs of the Abaqus DMLS simulation.

i. Average of Max NT11 across top surface of part

The average Max NT11 takes the average, over the entire scan time, of the maximum nodal temperatures across the top surface of the build part over the entire scan time. The maximum values averaged are found in Abaqus using the maximum envelope of the nodal temperature curves across the top surface of the build part at each time step. An example of the maximum envelope calculation done in Abaqus to find the maximum nodal temperatures at each time step of the simulation is shown below in Figure 3.8. Each curve on the plot represents the variation in temperature in temperature at one node on the top surface. This figure shows the temperature profile over all time steps of each node for the single-layer raster scan simulation; the top red line that follows the maximum peaks of the nodes is the maximum envelope, and these maximum values are used as the maximum nodal temperatures at each time step over the entire simulation.

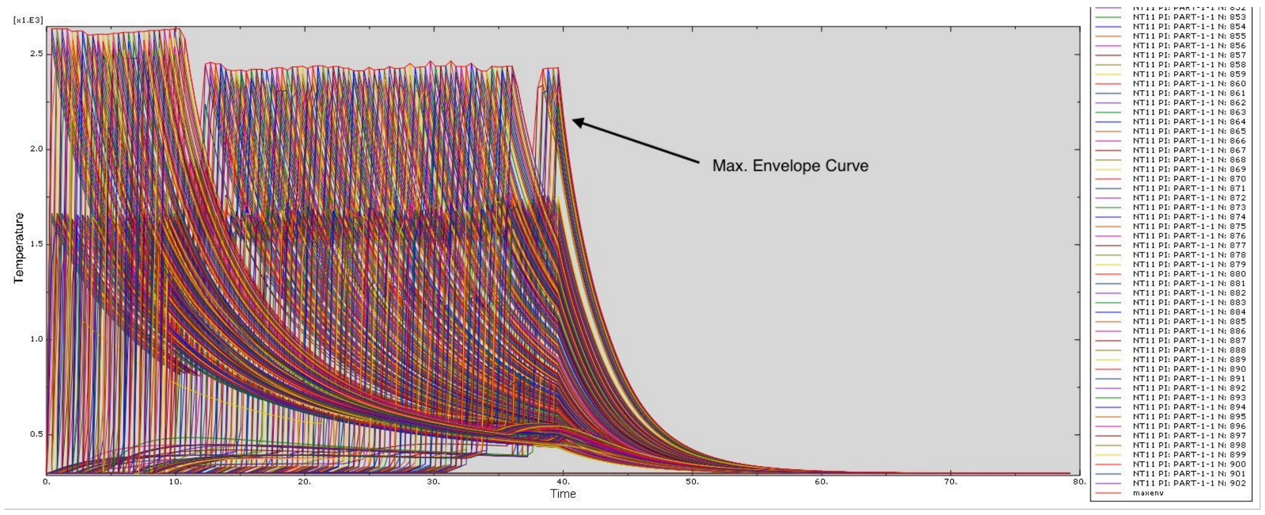


Figure 3. 8 Maximum Envelope Plot for 1-Layer Simulation

This objective function is relevant because it helps to see over the course of the scan what the average temperature across the top surface of the part is, as this could be useful when comparing temperature gradients between scans. Using this as an objective function in the genetic algorithm will also help optimize the scan to have a lower average peak temperature, thus minimizing the temperature gradient.

ii. Absolute Max of NT11 across entire part surface

The absolute max of NT11 finds the maximum, over the entire scan time, of the maximum envelope calculated by Abaqus of the nodal temperatures across the surface of the build part at each time step of the simulated scan. The maximum envelope is the envelope from all the NT11 curves for each node generated by Abaqus, so finding the absolute maximum across all of the nodes. This objective function is relevant because it helps see which scan results in the highest peak temperature during the scan time; if used in the optimization, reducing the peak temperature that occurs during the scan can help to reduce the temperature gradients over the entire part.

iii. Average of the Average NT11 across entire part surface

The average of the Average NT11 takes the average, over the entire scan, of the average nodal temperatures across the build part calculated by Abaqus at each time step of the simulated scan. Using the average nodal temperatures has the advantage of accounting for the lower end of the part's temperatures during scan time, in addition to the maxima. This objective function could provide insight into which scan pattern generates overall lower temperatures, and so lower thermal gradients.

Since stresses are directly related to the temperature gradients in a build part in laser sintering as shown in Chapter 2 (a selection of references to this include [10], [13], [18], [49]), it is important to have the temperature gradient appropriately defined for the analysis portion of the optimization. The first two metrics help to see a difference in peak temperatures on the build part and how they vary across the final scanned surface; minimizing the peak temperatures would help to minimize the uneven heating and cooling on the build parts. The third metric gives insight into the average temperatures across the final scanned surface; lowering them could also lower the peak temperatures.

An investigation into the differences between the raster scan and the un-optimized random scan, in Section 3.8, helped to decide which metric is best to define a temperature gradient in the simulation to use in the optimization.

3.8 Preliminary Results from Raster and Random Scan

Before optimizing the scan path, preliminary results on the differences between raster scans and random scans are analyzed; specifically, the nodal temperatures across the simulated build part after completing all scans are found using Abaqus/CAE and compared. Initially, all raster and random scans were run with the same properties as listed in 3.2.1, but with a scan speed of 0.1 m/s. These initial scans were run for both 1-layer and 5-layers, in order to see differences between the heating and cooling from the two scan modes as layers were built up. Computationally, these runs took about 30 minutes and 6 hours, respectively on Abaqus/Standard. It should be noted that the random scanning pattern is changed for each random scan, as the code generates a new random scanning path each time a new Abaqus input file is generated. Figures 3.9 and 3.10 show the nodal temperatures across the surface (NT11) for 1-layer raster and random scans, and Figures 3.11 and 3.12 show the nodal temperatures across the surface for 5-layer raster and random scans. Also included in these figures is a look at the center node (node 668) on the bottom surface of the block – this idea is taken from a past study looking at defining the temperature gradients [3], which looked at the thermal history of the center node on the top surface of the bottom layer. Looking at the center node helps to emphasize the heating and cooling cycles the part undergoes in DMLS.

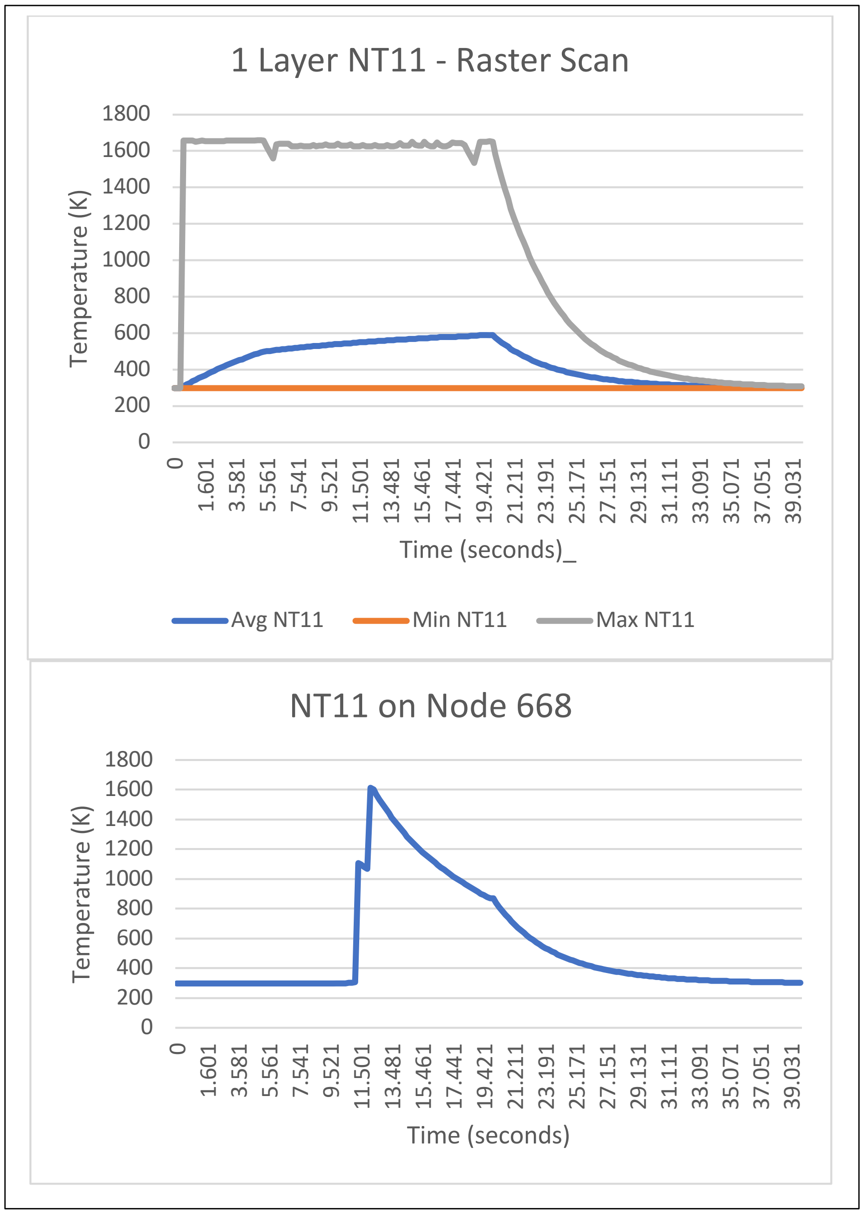


Figure 3. 9 Raster 1-Layer Scan NT11 – Original Parameters with Scan Speed = .1m/s

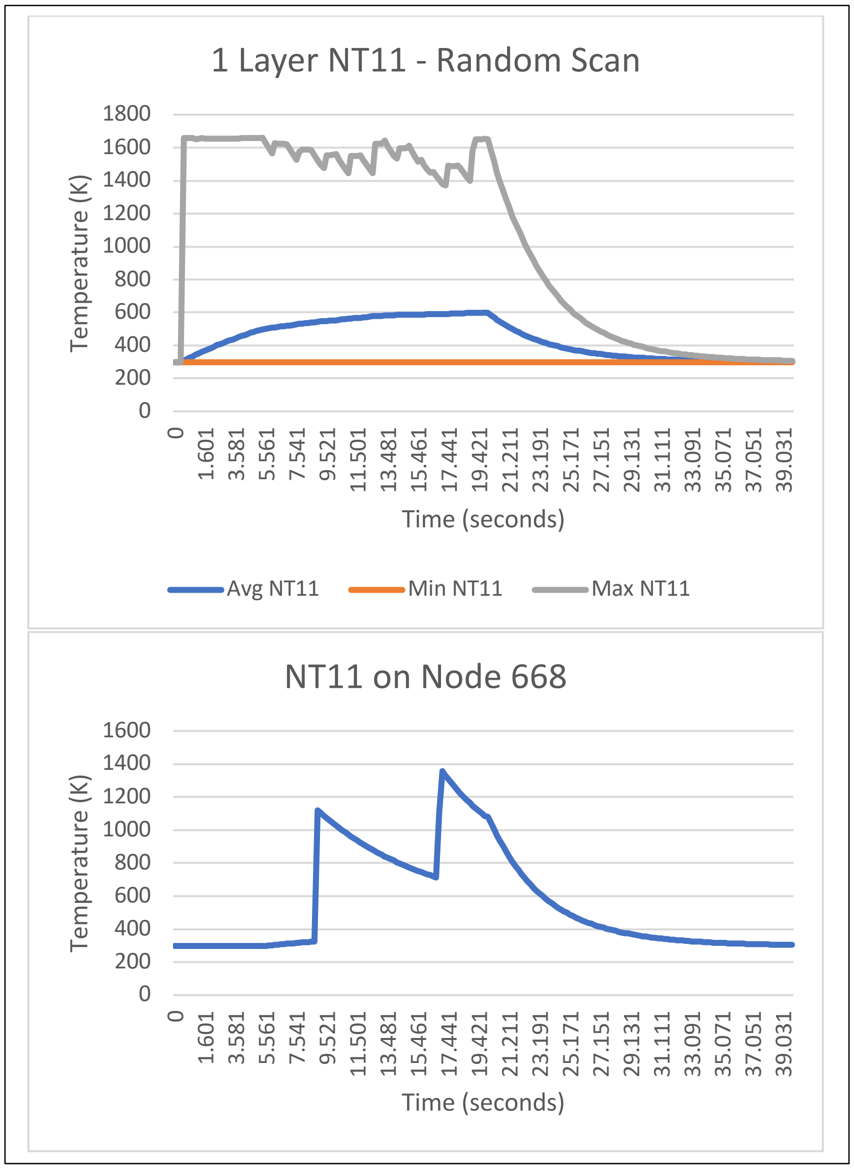


Figure 3. 10 Random 1- Layer Scan NT11 – Original Parameters with Scan Speed = .1m/s

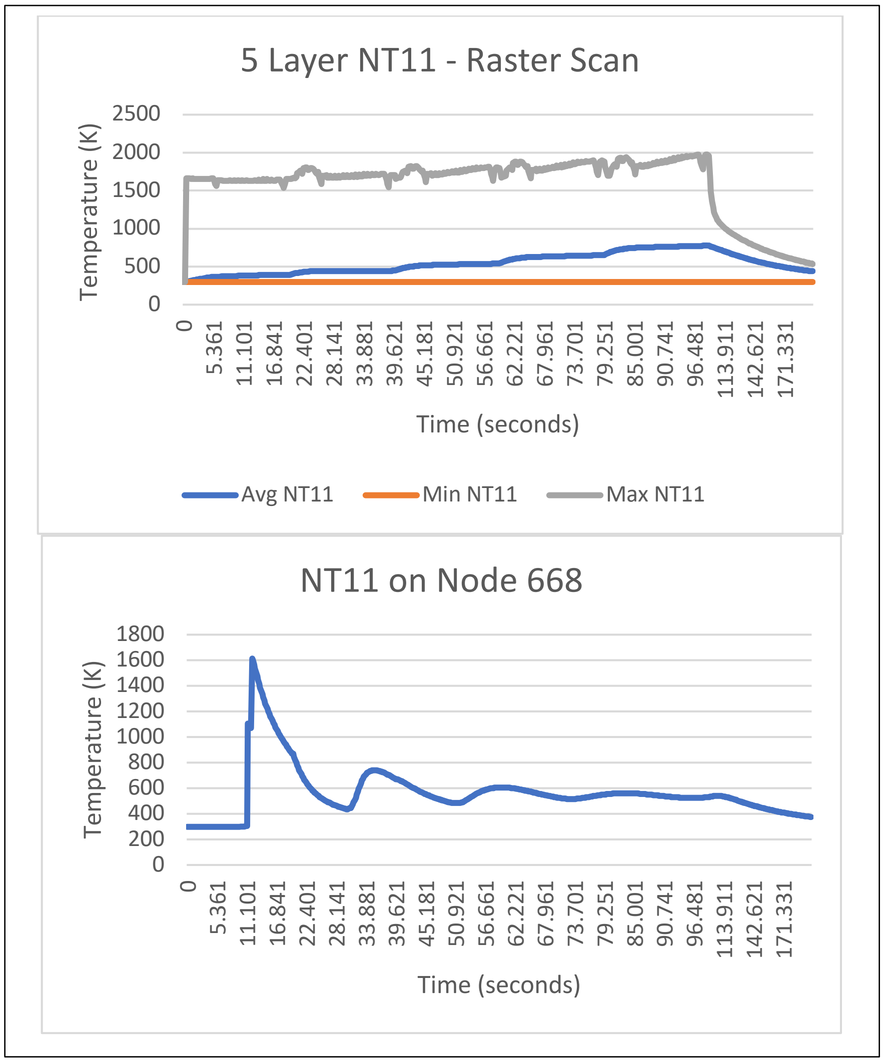


Figure 3. 11 Raster 5- Layer Scan NT11 – Original Parameters with Scan Speed = .1m/s

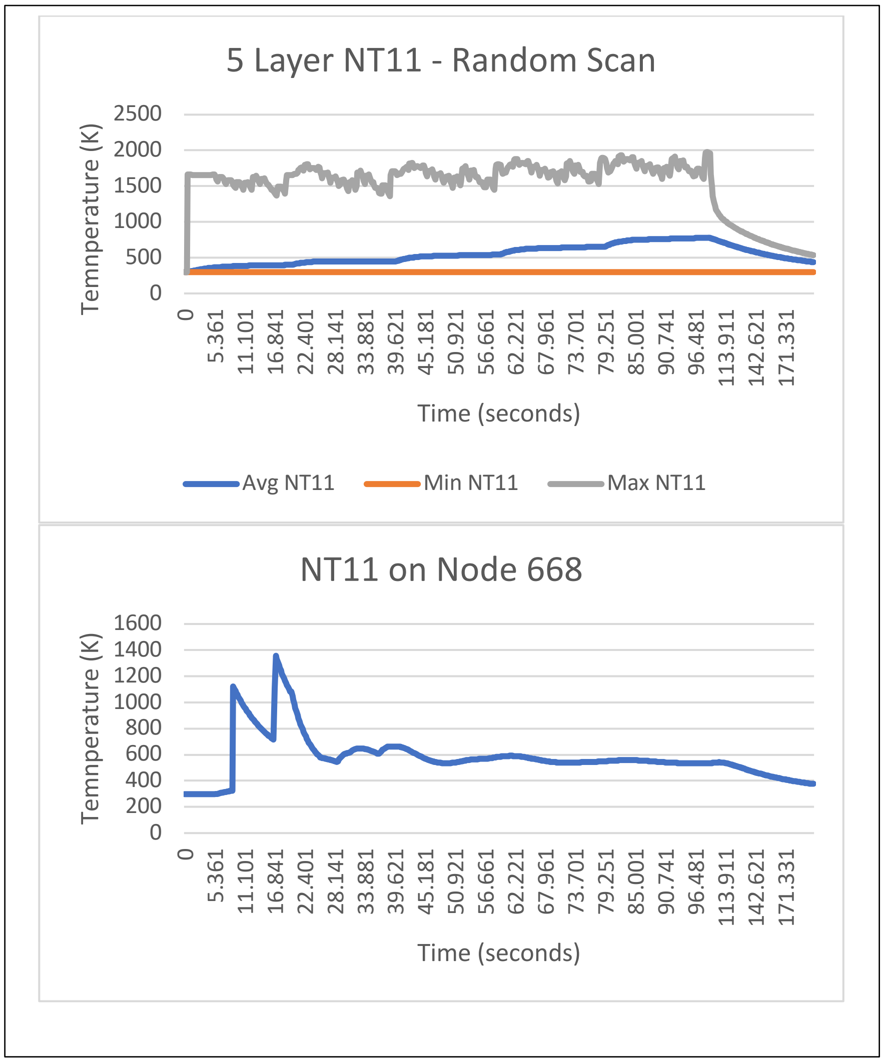


Figure 3. 12 Random 5- Layer Scan NT11 – Original Parameters with Scan Speed = .1m/s

The graphs for both sets of 1- and 5-Layer scans show that the simulation running at the set parameters and speed produces reasonable and realistic temperature results; the magnitude of the peak temperatures reached matches that found in past simulation studies [12], [20], [22], [27], [43].

For both the 1-layer scans (Figures 3.9-3.10), the maximum temperature reached from both scans is approximately 1660 K and they both share nearly identical average temperature profiles across the scan surface for the scan time. However, differences between the two scanning patterns in the thermal history can be seen in both the average maximum NT11 over the scanning period as well as the maximum temperature reached by the center node. The graphs show that the random scanning pattern drops to cooler NT11 values over the course of the scanning period (occurring from ~4 seconds to 20 seconds) when looking at the maximum NT11 curve , as opposed to the raster scan, which heats up to and stays close to the maximum NT11 reached. For both the 1- and 5-layer scans, the center node's maximum temperature reached is about 200K less using a random scanning pattern versus the raster scanning pattern. These results show that while overall differences are not significant, there are key thermal history differences between the raster and random scanning patterns that can be used to minimize thermal gradients on the simulated DMLS build parts.

The stresses calculated by Abaqus for the runs at the initial parameters were also looked at, to better assess the validity of the simulation. An initial 5-layer raster scan simulation was completed, and the Von Mises stress over the course of the entire scan was found in Abaqus. These measurements are shown in Figure 3.13.

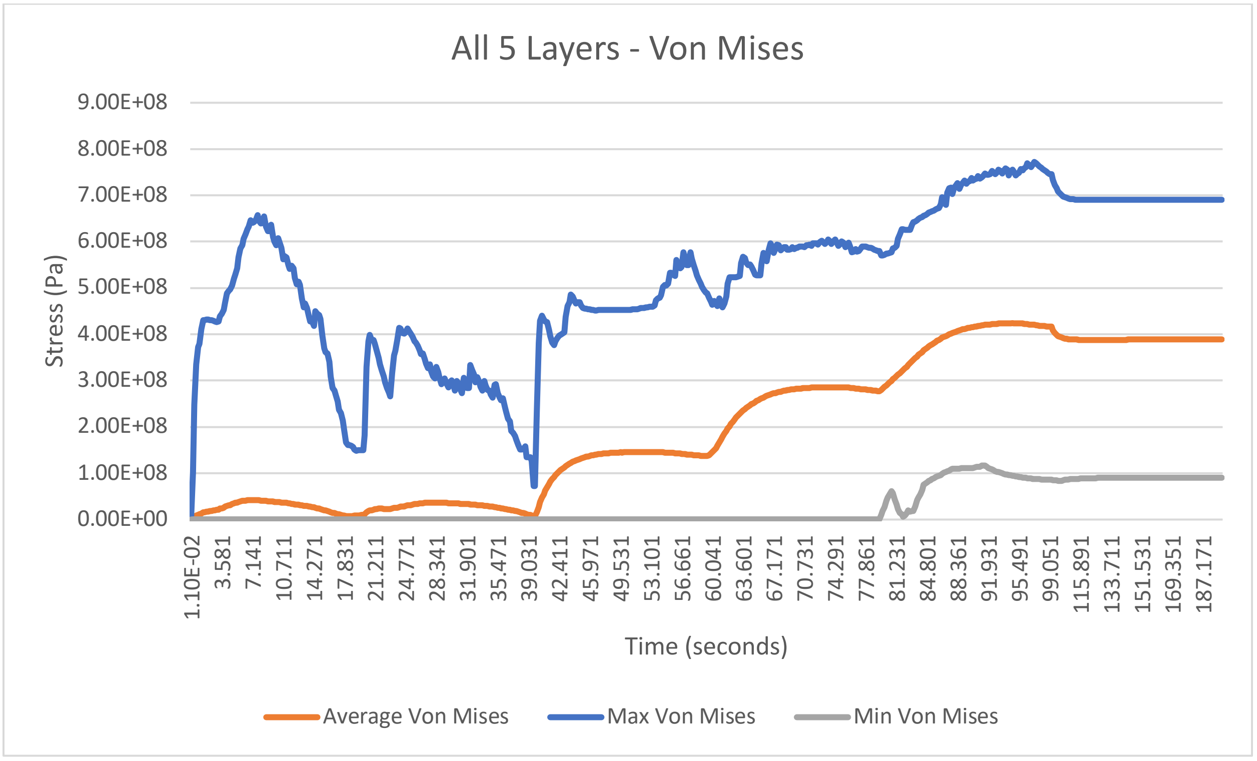


Figure 3. 13 Von Mises Stress for 5-Layer Raster Scan

As can be seen from the graph, the maximum Von Mises stress reaches approximately 790 MPa. In addition, the stresses are also shown to build up with each additional layer, as indicated by the 5 “hills” along the average Von Mises stress line; these stress measurements and the stress build-up are similar to what has been seen in past studies [9], [13]. The tensile and compressive yield stresses for Ti-6Al-4V are 880 MPa and 970 MPa respectively [41]; as can be seen, the more layers added in a DMLS part, the closer the stress values reach the yield stress of the material.

From the initial scan results with minimal difference in thermal history between random and raster scans, the next steps were to try to exaggerate these differences. One outcome of this research is to develop an approach to optimize the scanning pattern, which means finding an objective function that varies with scanning pattern to be used in the optimization. By exaggerating the thermal histories, the objective functions should become clearer and more useful in helping to magnify the effects of the scanning pattern on the thermal gradients of a build part in DMLS. In an attempt to exaggerate existing differences in thermal history existing between the raster and random scans, the simulation parameters were adjusted. First, the thermal conductivity values were multiplied by 10 to see if increasing the rate of heat transfer would increase the thermal history differences more than shown in the initial run. These results are shown in Figure 3.14.

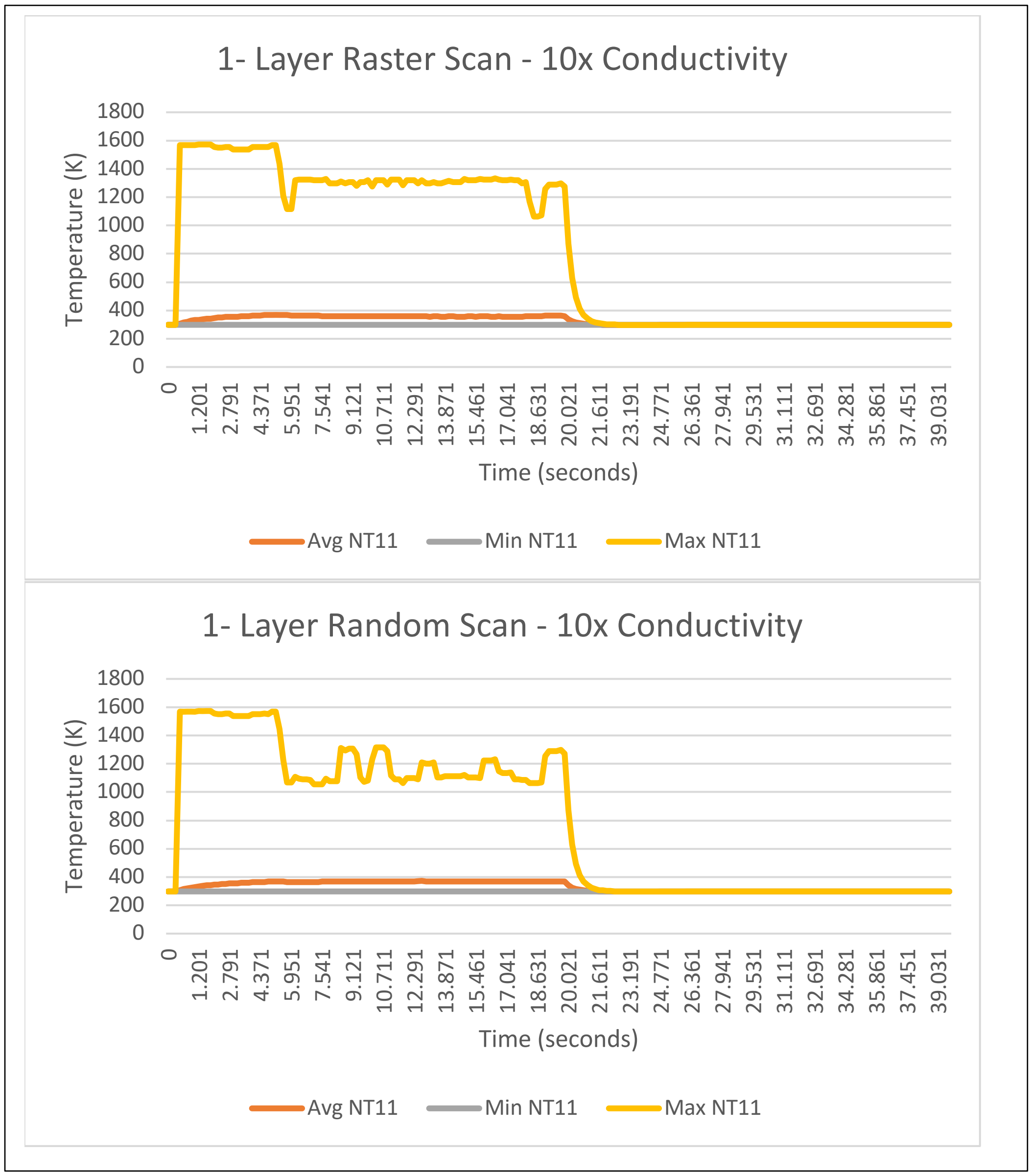


Figure 3. 14 Raster and Random Scans at Scan Speed = .1 m/s and 10x normal conductivity

As expected, the part cooled much quicker than before for both scans, but the differences in the maximum NT11 seen between the two for normal conductivity simulations are still similar overall; while at lower temperature values, the random scans still maintain a lower average maximum NT11 – almost identical to the normal conductivity runs differences – during scanning than the raster scan.

Another test run to see simulation differences in random and raster scans was a decrease in speed while maintaining the other parameters the same as normal. The speed was halved to 0.05 m/sec, to see if the differences found in the average maximum NT11 during scanning could be exaggerated from the normal and 10x conductivity value runs.

The NT11 results for half-speed simulation runs for raster and random scans for both 1- and 5-layer scans are shown in Figures 3.15 and 3.16.

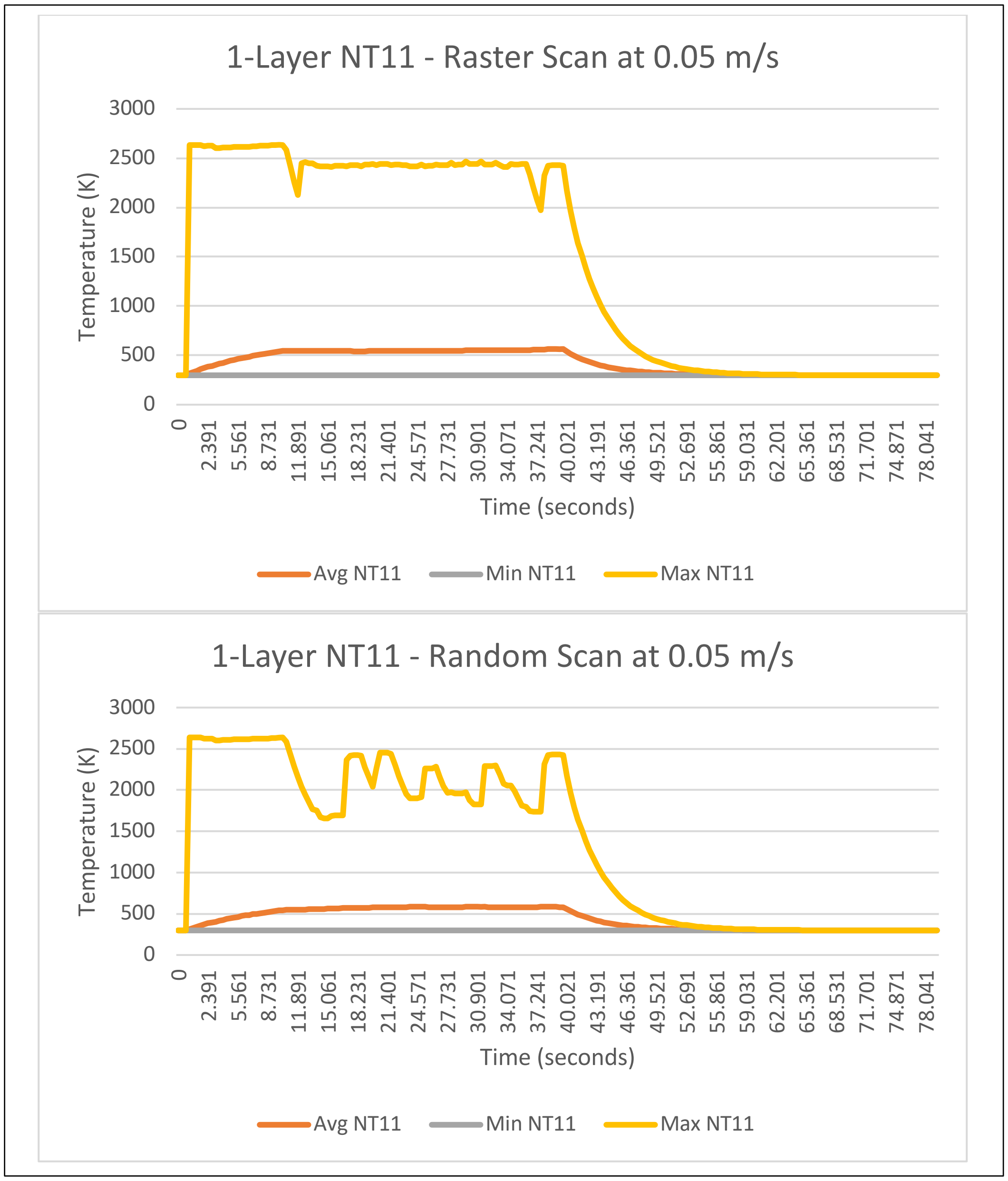


Figure 3. 15 NT11 Results for 1-Layer Random and Raster Scan at original parameters at Scan Speed = 0.05m/s

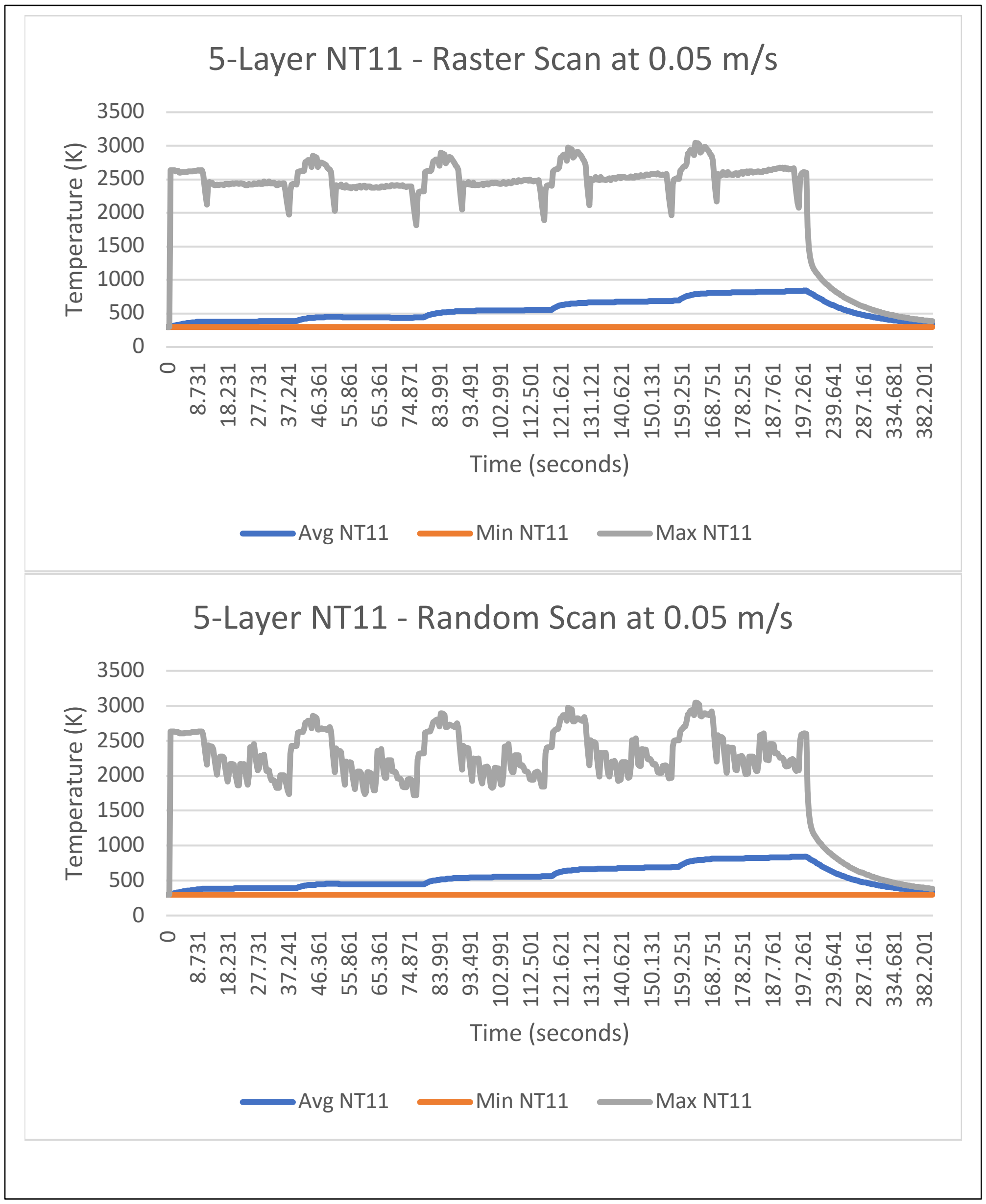


Figure 3. 16 NT11 Results for 1-Layer Random and Raster Scan at original parameters at Scan Speed = 0.05m/s

Figures 3.15 and 3.16 indicate that decreasing the speed increases the maximum temperature reached in the part to about 2600K and 2900K respectively for both 1- and 5-layer random and raster scans, which is to be expected because the laser spends more time heating each element on the simulated part. The same major differences appear in between the random and raster scans, namely in the average maximum NT11 for the part during the scan period (occurring ~11 seconds to 43 seconds for 1-layer scan and ~11 seconds to 197 seconds for 5-layer scan). A closer look in comparison of the maximum NT11 values between both scans during the simulation time is taken in Figure 3.17 for 1-layer scan and 3.18 for 5-layer scan..

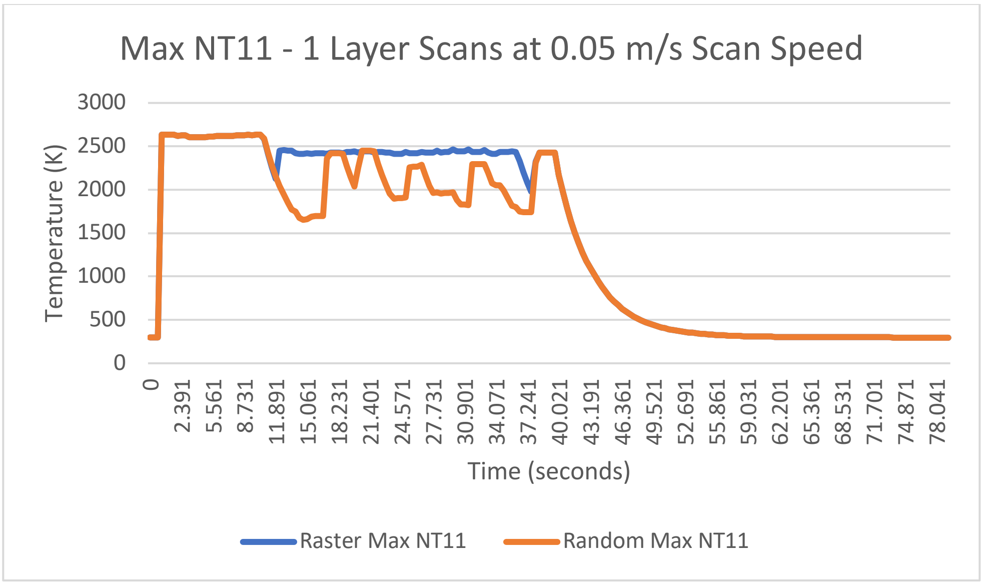


Figure 3. 17 Maximum NT11 over course of simulation for both 1-Layer scanning types at Scan Speed = 0.05 m/s

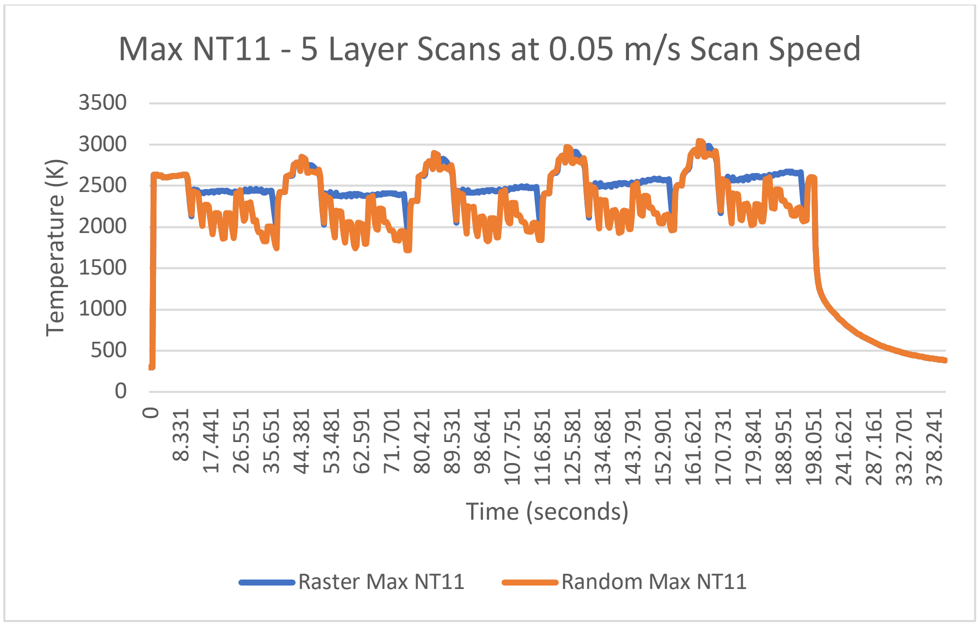


Figure 3. 18 Maximum NT11 over course of simulation for both 5-Layer scanning types at Scan Speed = 0.05 m/s

Figures 3.17 and 3.18 show that the random scanning pattern lowers the average maximum nodal temperature across the part surface during the simulation. The objective function that most clearly shows an improvement in thermal gradients between raster and random scans is the average of the maximum NT11. Specifically, the random scan pattern has an average maximum NT11 value of 1340 K over the course of the 1-layer simulation, while the raster scan has an average maximum NT11 value of 1461.3259 K; the random scan keeps the average maximum surface temperature roughly 100 K cooler over the scan period than the raster scan. For the 5-layer simulation, the random scan pattern has an average maximum NT11 value of 2042K, while the raster scan's average maximum NT11 value was 2214K over the scan time. In the case of 5-layers, the random scan keeps the average maximum surface temperature nearly 200K cooler over the scan period than the raster scan. When comparing the two, the power was 2500W for both but the speed was slowed down from initial simulations of 0.1 m/sec to 0.05 m/sec so as to exaggerate any differences in temperature gradients for the both types of scans; the idea of adjusting speed and power to increase peak temperatures reached on the part is found in past laser sintering studies [20]. Using a slower scan speed to help exaggerate this difference, the differences found in average maximum NT11 provides a feasible objective function in the minimization of the thermal gradients over the build part in the DMLS simulation created.

CHAPTER 4

OPTIMIZATION OF SCAN PATTERN

4.1 Description of Genetic Algorithm

A Genetic Algorithm was coded using MATLAB to optimize the DMLS scanning pattern. The genetic algorithm was chosen as the optimizer to ensure diversity of scanning path orders; genetic algorithms are capable of solving both linear and non-linear unconstrained problems, and explore all possible regions of a state space to find a viable solution [36].

The genetic algorithm code consists of a main script and 6 functions. The first function called by the main script defines the initial population; in this case, the initial population is random permutation arrays of the starting coordinates of paths along the part built in the input file. For the specific shape of the base and fin modeled in this research, the initial population of designs would consist of arrays with a random permutation of the numbers 1 through 16, the number of possible starting path positions.

6	3	16	11	7	14	8	5	15	1	2	4	13	9	10	12
16	6	14	12	1	13	5	9	15	11	10	3	7	2	8	4
8	2	3	13	1	6	10	9	15	14	16	5	11	12	4	7
7	6	12	14	3	10	8	15	11	5	4	16	1	13	2	9
4	5	13	10	14	8	6	12	16	3	15	9	7	1	11	2

Figure 4. 1 Sample Initial Population with Population Size of 5

Figure 4.2 shows how the population arrays relate to the build part and the starting positions. The random permutation of numbers 1 through 16 correspond to the vectors on the build part, not including the perimeter (as this is scanned first and separate from the random interior scan).

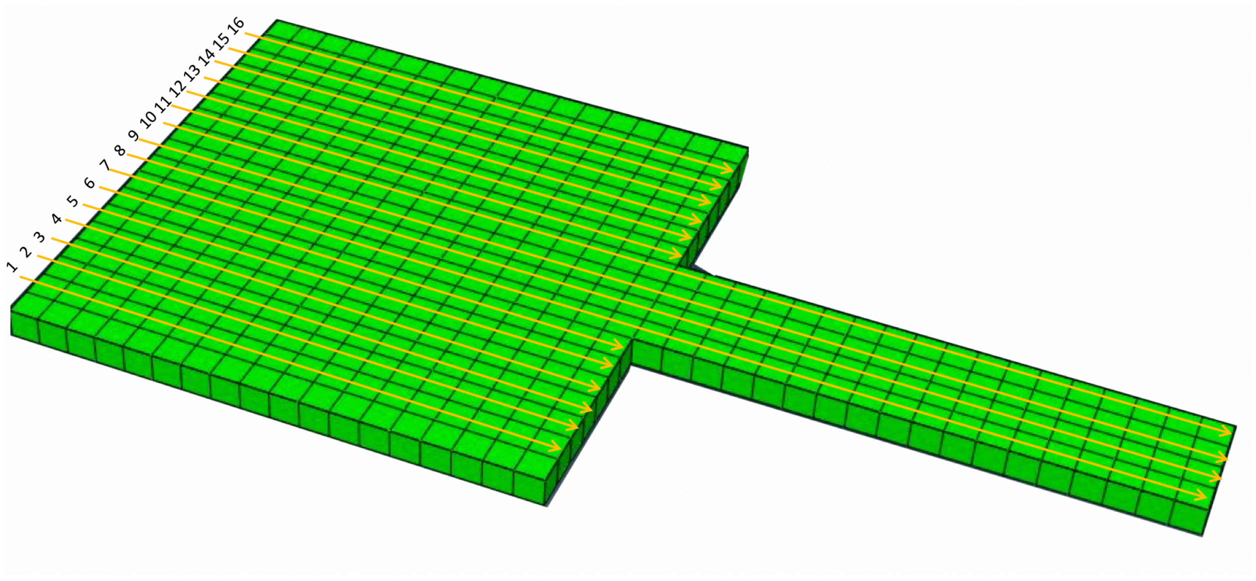


Figure 4. 2 Visual of Scanning Vectors used in GA Population

After the initial population is created, the initial "fitness" of these path arrays is found. In order to calculate the "fitness" of a scanning path array, the scanning patterns from the initial population are written into the input file for Abaqus. These input files are then submitted to Abaqus/Standard for analysis. Using the Abaqus2Matlab program [50], the "fitness" function is written to pull post-processing data from the Abaqus/Standard output files into MATLAB. For the purposes of this research, the nodal temperature data at the nodes after the dictated number of scans is extracted for each population member from the Abaqus output database (.odb) file and pulled into MATLAB. For the selection step of the GA, tournament selection was chosen; four designs from the initial population were randomly chosen twice and their fitness's compared, with the top two designs from each random selection of designs advancing through the algorithm as Parent 1 and Parent 2. Below is a schematic of how the tournament selection works; note that the fitness values, population size, and desired fitness are all arbitrary for the purpose of explaining the genetic algorithm.

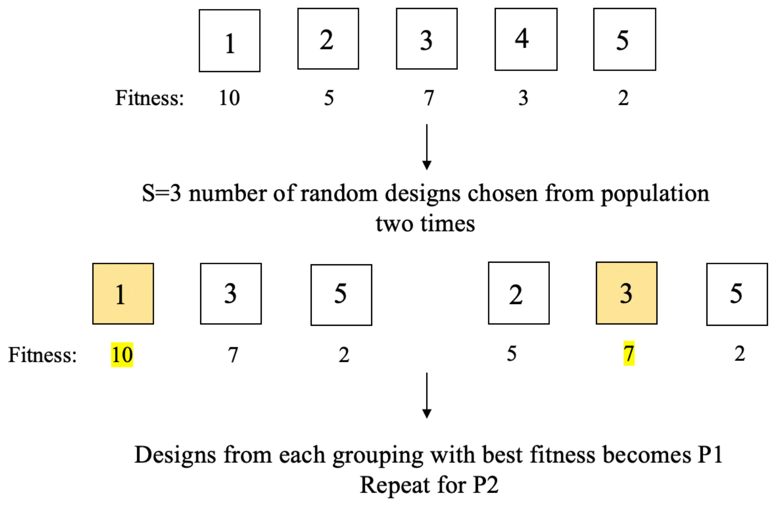
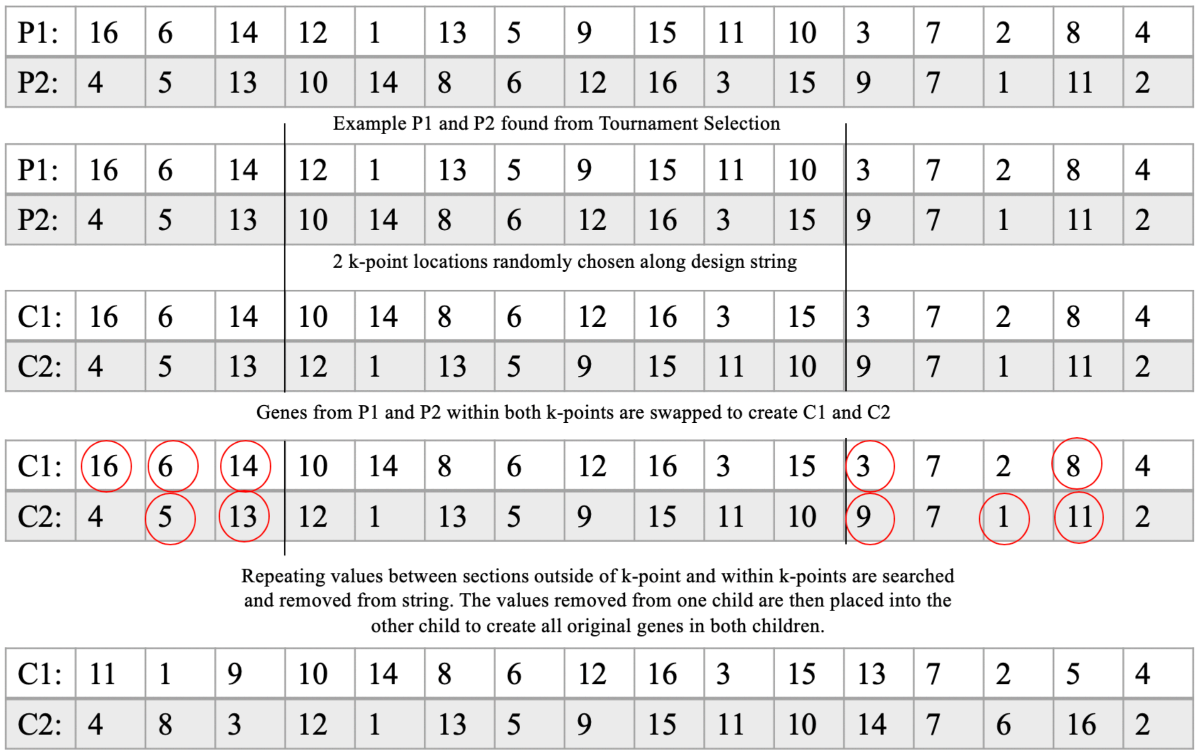

1
2
3
4
5
Fitness: 10 5 7 3 2
S=3 number of random designs chosen from population two times
1
3
5
2
3
5
Fitness: 10 7 2 5 7 2
Designs from each grouping with best fitness becomes P1
Repeat for P2

Figure 4. 3 Sample Tournament Selection from Example Design Population = 3

In the above figure, the total population is considered five. The number of random designs for the tournament selection is set at three; three random members are chosen twice from the total population of five. In this example case, maximum fitness' are considered desirable. From the two sets of three randomly selected population members, the most fit individual (or as stated in this example, the members with the highest fitness value) are selected to become the two parents to crossover and create two children.

The two parents are then crossed over to generate two children. An order-based crossover was implemented following previously explored order-based crossovers (OBX) [38] and outlined coding logic [51]; the crossover logic pulls from the Travelling Salesman Problem [39] to ensure crossovers with no repeating genes in either child. OBX was used to ensure that no paths in the design strings would be repeated after the crossover, which could cause the laser to scan some paths twice and others not at all. An example of this crossover method is shown below:



P1:	16	6	14	12	1	13	5	9	15	11	10	3	7	2	8	4
P2:	4	5	13	10	14	8	6	12	16	3	15	9	7	1	11	2

Example P1 and P2 found from Tournament Selection

P1:	16	6	14	12	1	13	5	9	15	11	10	3	7	2	8	4
P2:	4	5	13	10	14	8	6	12	16	3	15	9	7	1	11	2

2 k-point locations randomly chosen along design string

C1:	16	6	14	10	14	8	6	12	16	3	15	3	7	2	8	4
C2:	4	5	13	12	1	13	5	9	15	11	10	9	7	1	11	2

Genes from P1 and P2 within both k-points are swapped to create C1 and C2

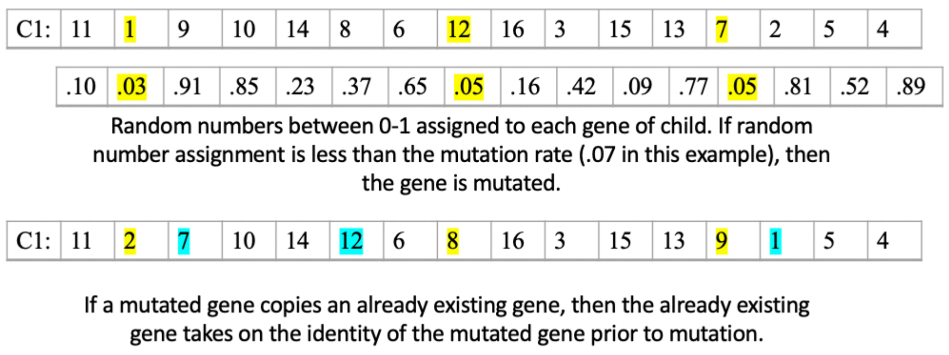
C1:	16	6	14	10	14	8	6	12	16	3	15	3	7	2	8	4
C2:	4	5	13	12	1	13	5	9	15	11	10	9	7	1	11	2

Repeating values between sections outside of k-point and within k-points are searched and removed from string. The values removed from one child are then placed into the other child to create all original genes in both children.

C1:	11	1	9	10	14	8	6	12	16	3	15	13	7	2	5	4
C2:	4	8	3	12	1	13	5	9	15	11	10	14	7	6	16	2

Figure 4. 4 Order based Crossover shown with sample P1 and P2

From the order-based crossover, the two children's design strings are created. To finalize the new designs, the children are mutated. The main code contains a set mutation rate under one; each path position (or gene) in the cross-over design arrays are assigned a random number from zero to one, and if the gene's assignment is below the mutation rate, then the gene is mutated. If a position in child's mutated array is the same as an already existing path in the design array, then the duplicated path takes on the original path assignment of the mutated gene prior to mutation.



C1:	11	1	9	10	14	8	6	12	16	3	15	13	7	2	5	4

.10	.03	.91	.85	.23	.37	.65	.05	.16	.42	.09	.77	.05	.81	.52	.89

Random numbers between 0-1 assigned to each gene of child. If random number assignment is less than the mutation rate (.07 in this example), then the gene is mutated.

C1:	11	2	7	10	14	12	6	8	16	3	15	13	9	1	5	4

If a mutated gene copies an already existing gene, then the already existing gene takes on the identity of the mutated gene prior to mutation.

Figure 4. 5 Example Mutation with Sample Child

The selection, crossover, and mutation process repeat with the population for as many children dictated in the code. Post mutation, the population has grown and genetic diversity between arrays has increased. The fitness of each design—both from the set of initial design as well as the new ones— is evaluated, and the designs with the worst fitness's are trimmed off and the population size returns to the original size of the initial population. Convergence can be checked in the code in multiple ways; convergence can be considered reached if the number of generations reaches the dictated maximum number of generations, if the number of function call exceeds the dictated maximum number of function calls, if the differences in average fitness's of the population falls under a certain tolerance, or if there is no difference in average fitness from one generation to the next. Convergence would signify an optimal scanning pattern has been obtained. If convergence has not been met yet, then the code writes the newly obtained population path arrays to the MATLAB code which generates the Abaqus input file, and the algorithm is run again.

4.2 The Genetic Algorithm in MATLAB

The Genetic Algorithm, with steps as described above in 4.1, was coded to take any number of scanning paths in the DMLS process and optimize the path order to limit the temperature gradients. The code links with Abaqus by writing the chosen path from each generation into the input file and running the simulation; then, the temperature results are analyzed using Abaqus post-processing capabilities and the Abaqus2Matlab program [50]. The temperature results are used to the evaluate the "fitness" of each path.

The exact parameters for the GA used are included in this section. Four different convergence criteria were created, to allow for changing the criteria for convergence depending on which was desired in any given run. The four convergence criteria created were:

- if the number of generations created by the GA reaches the maximum number set (in this case, 30)
- if the number of GA calls exceeded the maximum number set (in this case, 2000)
- if the average fitness between two consecutive generations are the same
- if the top fitness of the current generation is the same as the top fitness of the past generation

These options come from past genetic algorithm literature [52]. It was decided for the GA runs to terminate if either first the maximum number of generations were met or if first the average fitness between two consecutive generations was below a termination epsilon. Due to high computational costs of the GA, the maximum number of generations was set to 30 and ε=0.01. A lower tolerance epsilon was chosen to help with convergence as the code is so time intensive, and 30 generations could take multiple days to reach.

When picking a population size for the GA, it was important to pick a large enough population size to allow for diversity in the problem without having the computational costs of the GA becoming too high [52]. Given the nature of the problem and the single variable being assessed in the GA, it was decided that 10 would be an appropriate starting size for population.

The selection technique chosen for this GA was tournament selection; it is often viewed as a selection technique with high efficiency and easy implementation [52], [53]. In tournament selection, random individuals are chosen from the larger population and are set to "compete" against each other. The members with the highest fitness values go on to generate the children [52]. The number of individuals to be randomly selected in the tournament selection in this case was four, as to give enough competing selection without adding too much computational time.

The two-point order-based crossover was implemented in the GA. As described in the literature review, the two-point crossover helps to improve the chances of desired traits getting passed on to the children of each generation [37]. Implementing a two-point crossover also allowed for the crossovers to remain order-based, so the children did not have any repeating paths.

The mutation rate – the rate at which chromosomes are mutated for members of a population in a given generation [52] – chosen was 0.01; this mutation rate prevents the GA from becoming a random search, but still allows for mutations to occur that will prevent the GA from getting trapped in local optima [52].

As described in Section 3.8, the objective function chosen to minimize the thermal gradients on a build part in DMLS was the average maximum NT11 values across the simulated part's surface. This objective function was the goal fitness function used in the GA. Utilizing the Abaqus2Matlab toolbox's function ReadFil(output_file.fil, Record Key), Using the Record Key 201, MATLAB is able to read the temperature values from the .fil file. The maximum envelope of the temperatures across the part surface was recorded (as was explained in Section 3.8 for how Abaqus find maximum NT11) and then averaged. This value was used as the fitness for a certain scanning pattern and was used in the trimming of the population. Once the children were created and the entire generation evaluated, those scanning patterns that produced the least fit individual were cut from the population for the next generation, to keep the population size down to 10 and maintain only the fittest scanning patterns.

There is one note in regard to the temperature gradient analysis done directly in Abaqus and that done using the Abaqus2Matlab toolbox. In Abaqus, both the nodal temperatures and the elemental temperatures are available output options. In Chapter 3, the nodal temperatures were evaluated for initial comparisons of raster and random scans since the nodal temperatures are more accurate and do not require Abaqus to interpolate any values [44]. However, the record key available in the Abaqus2Matlab toolbox available for temperature pulls the elemental temperatures [50]; This is an averaged temperature over the whole element rather that a single node temperature. This introduced a complexity if we were to attempt to regenerate the nodal temperatures from these elemental temperatures, but for purposes of minimizing the temperature values used to define the temperature gradient, minimizing the elemental temperatures in the GA should accomplish the same goal and also minimize the nodal temperatures of the part.

Another thing to note is how the fitness function was calculated in the GA in MATLAB. While the average max NT11 was looked at in Abaqus as a temperature gradient definition, the fitness function in the GA had to be able to be calculated multiple times, over each generation. It was decided that in the GA in MATLAB it would be most effective to average all the temperatures over all the time steps and take the maximum of these average for all elements. By minimizing this average, the GA will be able to find the lowest difference between the high and low temperatures, and thus a lower temperature gradient. This metric in MATLAB also creates a fitness function The initial results for the output averaged maximum temperatures from MATLAB for the raster scan discussed in Chapter 3 are included below and are used in Section 4.5 in comparing temperature gradients generated from the optimized scanning path to the raster scan along with the post-processing values of the average maximum NT11 values from Abaqus. The Abaqus run with the optimized scan pattern return the correct nodal temperatures which are significantly higher than the elemental temperatures.

Temperature Output (maximum average elemental temperatures) of Raster Scan from MATLAB – One Layer	526.2899 K

Temperature Output (maximum average elemental temperatures) of Raster Scan from MATLAB – Five Layer	788.0681 K

Table 4. 1 MATLAB Elemental Temperature Output for 1- and 5-Layer Raster Scan

4.3 Running the GA on Clemson's Palmetto Cluster

Due to high computational needs and long processing times, the Genetic Algorithm code tied to Abaqus was executed on Clemson's Palmetto Cluster. In order to effectively use the GA, each population in the GA should be around 10; this means 10 simulations must be run on Abaqus and analyzed for their fitness in each generation. 30 generations were chosen as the convergence criteria for this GA. That comes out to 300 Abaqus simulations. If a five-layer scan is being completed with each Abaqus simulation being run in sequence, that would mean about 2,400 hours of computational time needed. The Genetic Algorithm code was edited by Grigori Yourganov of the CITI Group to help parallelize the Abaqus runs from a population onto the Cluster, in order to help lower wall-time and increase the code's efficiency. The edits made to the code involve sending each Abaqus run in a population to a different node in the Palmetto Cluster, to allow for all members of the populations to be run on Abaqus simultaneously and cut down on the total time needed for the GA to cycle through on generation.

In order to help shorten the computational time from MATLAB, much of the code was rewritten into Linux in order to run batch scripts and avoid using the MATLAB interface. This code is included in Appendix C. This allowed for easier parallel processing of each Abaqus run, as well as taking out some of the computational time of having to also run the MATLAB jobs for the GA in parallel as well. The most computationally costly portion of the GA code is the reading of the results in MATLAB of the Abaqus output file; converting this step onto Linux help to facilitate the input/output step and reduce the length of time. This portion of the code, however, is the still bottleneck when it comes to the speed that the GA processes each generation at. Four batch scripts were written, for the different stages of the Genetic Algorithm; initialization of new generation, simulation runs for each member of the generation, crossover and children, and trimming. Within the batch scripts, the MATLAB scripts are called and run utilizing the Cluster specifications called in each batch script. Each script called the necessary amount of memory from the processing nodes depending on which part of the GA was being completed.

4.4 Optimization Results

Three Genetic Algorithm runs were completed first for optimizing the scan pattern on the one-layer simulation. All three single-layer scan runs were done under the same parameters. An initial population of 10 was chosen with 4 parents being selected via tournament selection and 4 children being created in each generation; the total population was trimmed back to 10 after comparing fitness values among the population prior to the next generation. The other GA parameters remained the same as described in the previous section. The first run took a total of approximately 52 hours and found the following optimal scanning pattern and corresponding temperature gradient in table 4.3 under Run

1; the convergence occurred when the difference of average fitness between generations was near zero. The convergence happened after the 7th generation's fitness values were evaluated. The evolution of optimal results can be seen in Figure 4.6. The second run took a total of approximately 48 hours and found the following optimal scanning pattern and corresponding temperature gradient in table 4.3 under Run 2. The evolution of the second run's optimal results can be seen in Figure 4.7. For the second single-layer optimization run, the convergence happened after the 10th generation; there was no difference in any of the scanning patterns of members of the population at this point. The third run took approximately 30 hours to complete and found the following optimal scanning pattern and corresponding temperature gradient in table 4.3 under Run 3. The evolution of the third run's optimal results can be seen in Figure 4.8. In the third run, the convergence happened after the 6th generation, at which point there was no difference in scanning patterns of the members of the total population. The decrease in time for Single-Layer Run 2 and Single-Layer Run 3 is most likely due to less computational traffic on the Palmetto Cluster, allowing all runs in the GA to skip computing queues that may have existed during Single-Layer Run 1.

One-Layer Simulation Optimization Results:	
Run 1	

Optimal Scan Pattern:	9 13 1 8 5 10 3 11 6 14 16 4 12 7 2 15
MATLAB Average Max. Temperature Output:	520.0048 K
ABAQUS Temperature Gradient Output:	1321.0255 K
Run 2	
Optimal Scan Pattern:	8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13
MATLAB Average Max. Temperature Output:	520.8018 K
ABAQUS Temperature Gradient Output:	1377.84039 K
Run 3	
Optimal Scan Pattern:	10 3 2 1 8 14 4 13 12 16 7 11 15 5 9 6

MATLAB Average Max. Temperature Output:	516.9315 K
ABAQUS Temperature Gradient Output:	1367.6019 K

Table 4. 2 Single Layer Full Simulation GA Optimal Results

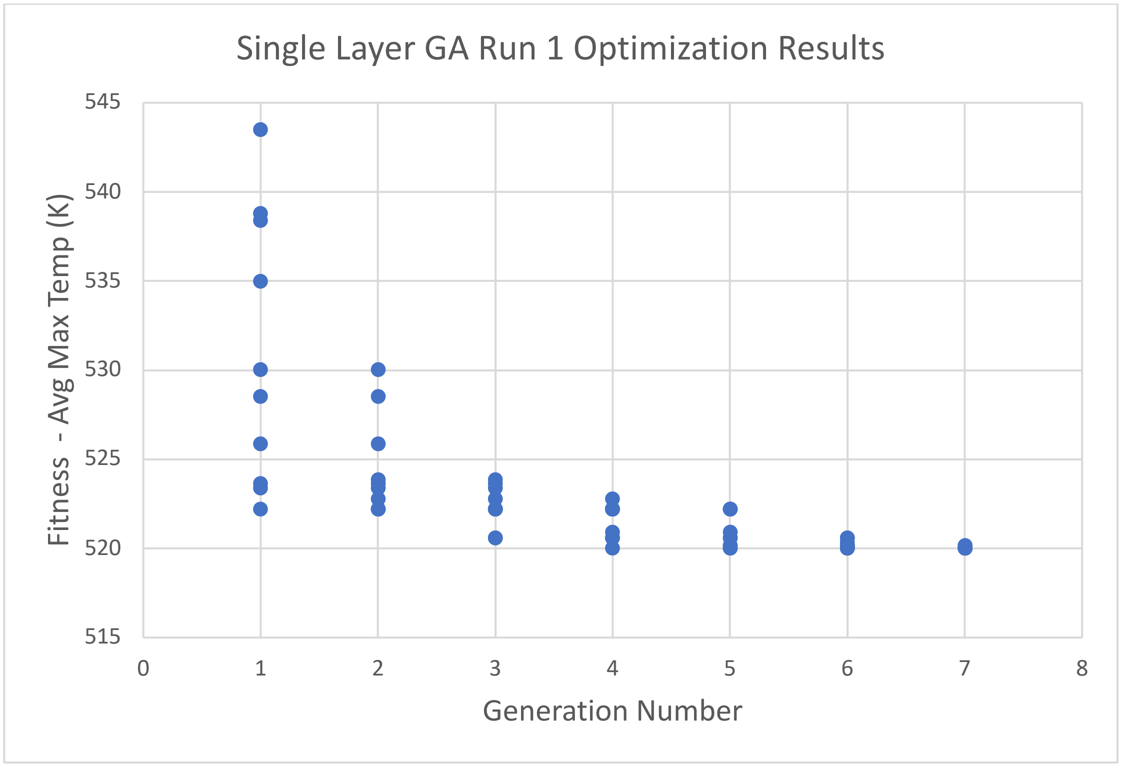


Figure 4. 6 Single Layer Full GA Optimal Solution Evolution: Run 1

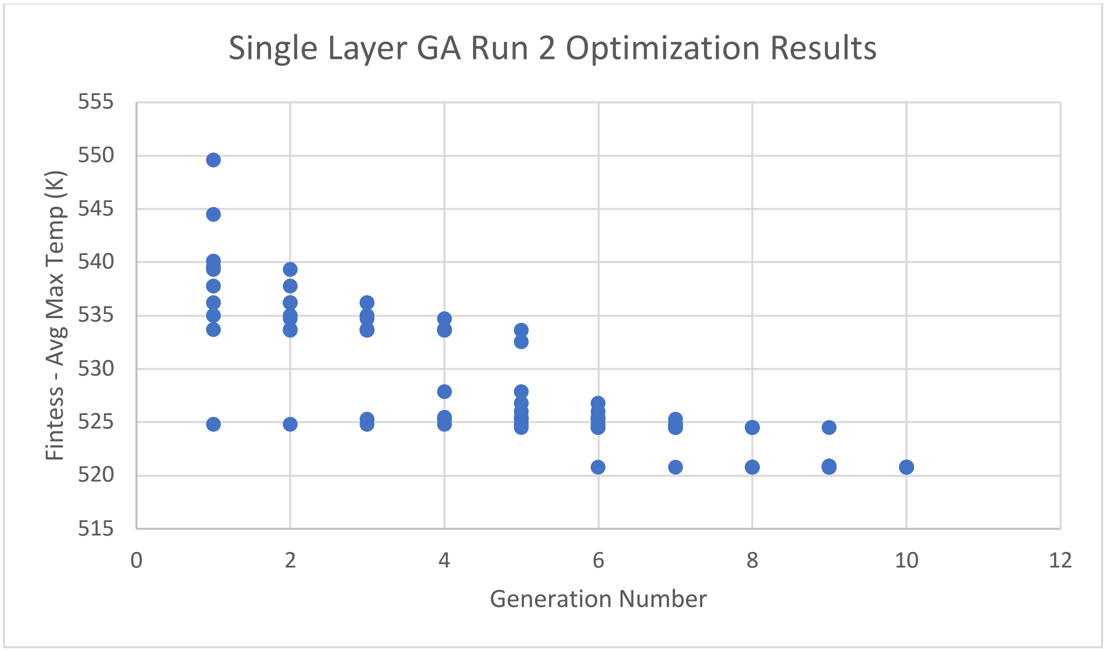


Figure 4. 7 Single Layer Full GA Optimal Solution Evolution: Run 2

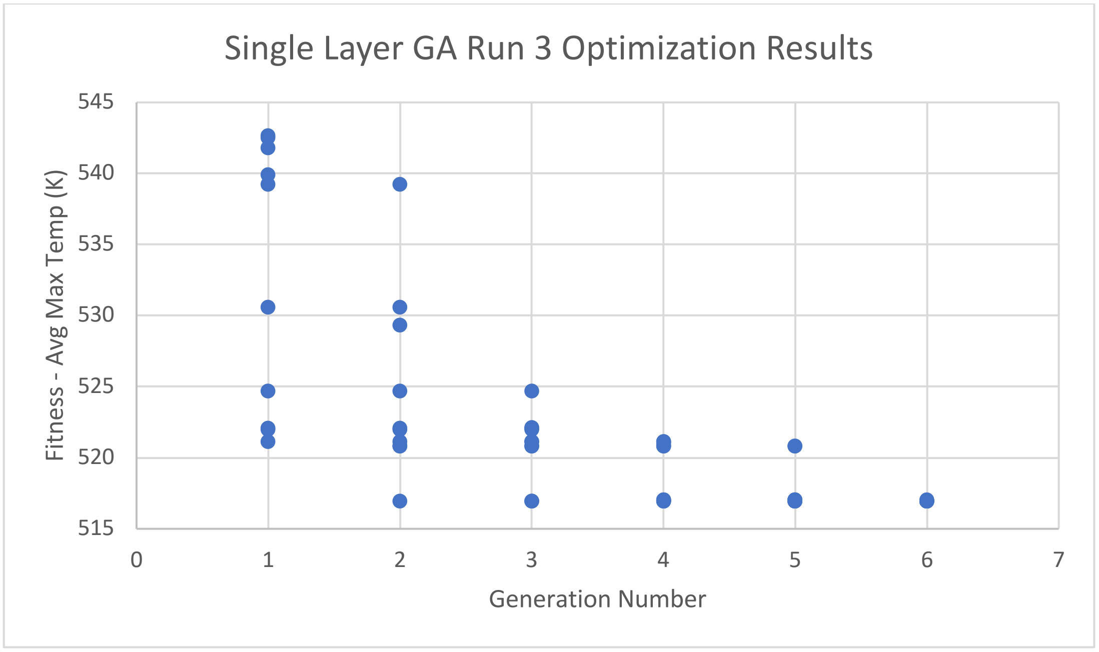


Figure 4. 8 Single Layer Full GA Optimal Solution Evolution: Run 3

For Single-Layer Run 1, the optimal scanning pattern was found first in the 4th generation, and persisted through the 7h generation, with the 1st generation being considered the initial population generated as in the test run of the single layer optimization. For Single-Layer Run 2, the optimal scanning pattern was found in first in the 6th generation and persisted through the 10th generation. For Single-Layer Run 3, the optimal scanning pattern was found in the 2nd generation and persisted through the 6th generation. The GA evolution of all three runs shows that all runs started with a widespread initial population, but that after the first crossover and trim was completed, the population spread began to become less. A discussion on the differences between the different runs and more in-depth comparison to original raster scan results is included in the following section, 4.5 Comparison of Optimized Results to Raster Scan.

After successful single-layer GA optimization runs, a five-layer simulation scanning pattern GA optimization run was completed. It is important to look at the scanning pattern optimization on multi-later simulation, as the multi-layer build parts are more realistic to industry as well as where the high temperature gradients become more of an issue, as discussed in the Chapter 2 Literature Review. The five-layer simulation GA ran under the same parameters as the single-layer scan did, but with five-layers being scanned in the simulation. The first 5-layer GA run took roughly 5 days to run; some generations took roughly 20 hours to process, due to the increased computational time of a 5-layer simulation as well as possible queues on the Palmetto Cluster. The optimal scanning pattern, corresponding fitness, and the evolution of the GA results is shown below in Table 4.3 and Figure 4.9, respectively.

Five-Layer Simulation Optimization Results:

Scan Pattern:	15 6 10 2 8 7 11 4 3 16 12 13 1 9 5 14
MATLAB Temperature Gradient Output:	685.6925 K
ABAQUS Temperature Gradient Output:	2018.1599 K

Table 4. 3 Five Layer Full Simulation GA Optimal Results

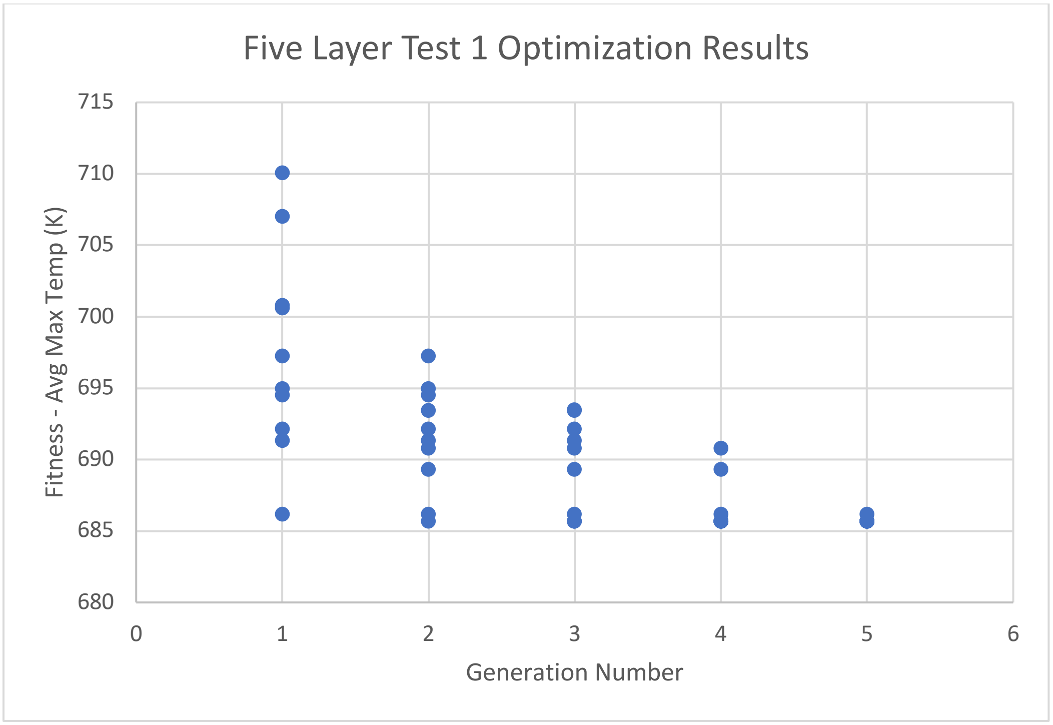


Figure 4. 9 Five Layer Full GA Optimal Solution Evolution: Run 1

The five-layer GA converged after 5 generations, with the initial population being considered the 1st generation; the code converged after the difference in average fitness between generations was near zero. The optimal scanning pattern first appeared in the 2nd generation and persisted through the 5th generation. The optimal scanning pattern found for the five-layer scan is the same scanning pattern on each layer; the scanning pattern was not set to vary on each layer. The five-layer optimization run has similar patterns to the single-layer optimization runs when looking at the optimal solution evolution, indicating that the GA's uses can be extended to various size parts. When looking at the optimal scanning patterns found for all three runs, it can be seen that the scanning patterns switch between the outer vectors (towards the edge of the part) and the inner vectors (towards the center of the part) every scanning position, suggesting this dispersal of scanning positions helps to even out the heating and cooling across the surface of the build part. This pattern is seen both between the three single-layer GA solutions as well as between the single-layer and five-layer GA solutions, with some adjacent scanning vectors appearing in Single-Layer Runs 2 and 3 and the Five-Layer run.

Using the genetic algorithm for solving this optimization problem was important as this problem can hold many solutions and has a more complex search space, which are characteristics of optimization problems that GA's are efficient at handling [52]. With 16 different scanning vectors and an unconstrained problem, an exhaustive search could have been completed; the initial population scanning orders could have been switched more to try to find a true optimum. However, computational limits made this challenging. However, it can be seen from the above results that using a GA with a DMLS simulation creates a feasible approach to optimizing the scanning pattern of the DMLS process to reduce the temperature gradients. This approach can be generalized for any shape, as well, making it a versatile tool.

A full list of the genetic algorithm runs and the results—the scanning patterns and corresponding fitness— from each generation are included in Appendix D.

4.5 Comparison of Optimized Results to Raster Scan

A summary of the optimization results is below. The temperature gradients from the optimal scanning patterns are compared directly with the original raster scan temperature gradient results in Table 4.5.

Temperature Gradients of Original and Optimized Simulation Runs	
Original 1-Layer Run, Raster	Optimized 1-Layer Run
MATLAB Output: 526.2899 K Abaqus Output: 1461.3259 K	RUN 1: MATLAB Output: 520.0048 K Abaqus Output: 1321.0255 K TEMPERATURE GRADIENT REDUCTION (Abaqus Results Reduction): 140.3004 K
	RUN 2: MATLAB Output: 520.8018 K Abaqus Output: 1377.84039 TEMPERATURE GRADIENT REDUCTION (Abaqus Results Reduction): 83.4855 K

	RUN 3: MATLAB Output: 516.9315 K Abaqus Output: 1367.6019 TEMPERATURE GRADIENT REDUCTION (Abaqus Results Reduction): 93.7240 K
Original 5-Layer Run, Raster	Optimized 5-Layer Runs
MATLAB Output: 788.0681 K Abaqus Output: 2214.4722	MATLAB Output: 685.6925 K Abaqus Output: 2018.1599 K TEMPERATURE GRADIENT REDUCTION (Abaqus Results Reduction): 196.3123 K

Table 4. 4 Comparison of Raster Scan Results vs. Optimized Scan Pattern Results

When looking at the single-layer optimized scanning patterns, it can be seen that the GA found three different optimal scanning patterns. The first GA run produced a scanning pattern that generates a lower temperature gradient overall when looking at the

Abaqus output; the drop in the temperature gradient is greater in the in the first run's scanning pattern. The MATLAB output of all three runs are very close, suggesting that there are multiple optimal scanning patterns that can achieve the same fitness values. The second run's temperature gradient is higher, though, than the first by roughly 50K and third run's is higher by roughly 40K. The second and third GA runs might have run into a local minimum, hence why the scanning patterns do not produce as low of a temperature gradient as the first run's scanning pattern does. Overall, the single-layer optimization runs show that scanning patterns outside of the raster scan can affect the temperature evolution on a DMLS part during the build and allow for lower temperature gradients than what the current raster scan generates. The multiple runs also show the possible existence of multiple optimal scanning patterns for a DMLS build part.

The five-layer optimized scanning pattern sees a greater reduction in temperature gradient from an optimal scanning pattern than the single-layer optimization did; this was expected as the residual stress build up occurs more severely in higher numbers of layers, as the uneven heating and cooling becomes more of an issue [9], [13]. At the found optimum scanning pattern, on average, the maximum temperature across the build part was 200K less at each time step (barring the initial time steps when the part was at the initial temperature condition). Due to high computational costs, only one five-layer run was completed to compare with the initial raster scan results; as the two single-layer GA runs suggest, a different scanning pattern could exist for the five-layer scan that could possibly further reduce the temperature gradient along the build part in the simulation. However, the results from the five-layer GA run show that a scanning pattern does exist that will minimize the thermal gradients on the build part. As is discussed in Chapter 5:

Future Work, more comprehensive GA runs can be competed with higher population sizes and high number of parents/children which will help generate more robust optimization results for more in-depth build simulations.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Research Conclusions

Overall, this research shows that it is possible implement a DMLS simulation and optimize the temperature profiles. Chapter 3 explained in detail the set-up and process that the DMLS simulation takes, and Chapter 4 explains the set-up and execution of the Genetic Algorithm written that minimizes the temperature gradient across a build part through adjusting the order of scanning pattern from the laser in DMLS. Through the GA, an optimal scanning pattern was discovered for both a single-layer build and a five-layer build. This optimal scanning pattern sequence shows reduced temperatures across the build part when compared to the traditional raster can. Since the temperature gradients are lowered, the post-build deformations should also be reduced as the steep temperature gradients are what lead to the deformations, as discussed in Chapter 2.

To complete the optimization of scanning pattern, this research explored different possible metrics for temperature gradient on the build part to be used at the objective function. It was determined that the average maximum nodal temperature was a viable metric to minimize in the GA, as it showed the most variation among different scanning patterns. The aim of this research was to lay groundwork for optimizing the laser scanning path in DMLS; this was achieved through a working simulation and genetic algorithm, which is able to find a scanning pattern that minimizing temperature gradients induced from the laser on the build part during the build process.

5.2 Future Work

While this research showed that optimizing the scan pattern will result in a lower temperature gradient, and therefore, less thermal stresses, much more work can be done to improve on the results and broaden the applications of this research. Updating the simulation to match more closely the in-situ process of DMLS would be beneficial to increasing the validity of the optimization results. These updates could include convection and radiation effects, as well as smaller element sizes to increase the accuracy of the heat flux data from the moving laser. These updates could be made with the availability of more computational power, as these updates would increase the computational time of the simulation and the overall GA.

Changes to the GA that could help further this research would be to design it to optimize the scan pattern for each layer individually. Currently, the multi-layer scans have the same vector scanning order for each layer. However, each layer of a multi-layer scan could exhibit a different scan pattern, and this could help to further minimize the overall temperature gradients on a build part and make the GA more robust in its results. In order to achieve this, the simulation would have to be adjusted so that each time a new layer is dictated to be scanned, the scanning order is also changed. The GA would also have to be adjusted to look at and optimize the scanning patterns of each individual layer of the multi-layer scan.

Another aspect that should be considered in future work is the application of the simulation and GA to other shapes; simulating and testing on other shapes would prove the diversity of the GA within the realm of DMLS applications, as well as possibly show any patterns in the optimal scanning pattern that reduces the temperature gradients.

Testing on other shapes can also explore the effects scanning pattern in DMLS might have on various types of features, such as thinner overhangs or rounded edges.

Ultimately, taking the optimization from simulation to experiment would greatly increase the optimal scanning pattern GA application. Future work should apply the optimal scanning pattern found in the GA to the physical scanning and building of a part in DMLS in order to validate both the simulation being ran as well as validate the application of the optimal scanning pattern to an in-situ build. This validation could help show the physical improvements on build part deformations from using an optimized scanning pattern, would help strengthen the case for the GA's application to real world use.

APPENDICES

APPENDIX A. MATLAB SCRIPT FOR DMLS SIMULATION ABAQUS INPUT FILE

Appendix A includes the 9 MATLAB functions that calculates and generates the node and element mesh data and input files, the heat flux input data and input files, and the laser scan position data and input files. The overall simulation Abaqus input file is generated at the end as well, calling the individual input files for mesh and heat flux data that are generated. The majority of the code was written by Jennifer Snipes and Ramaswami Subrahmanian for their post-doctoral research, but adjustments were made for this specific project. Also included after the code is a simulation input file that would be generated from the code.

A1_main.m

```
%% Housekeeping


clc
clear
close all
fclose all;
tic;

%% For graphs

line_width = 5;


%% Generate meshes

% This flag should be set to 'true' when running this program for the
first
% time after any mesh parameter
% (e.g. the number of layers in the z-direction, mesh.num_elem_z)
% has been set in the program P01_generate_mesh.m .
generate_mesh = true;
disp('Generating mesh ...')
run P01_generate_mesh.m


%% Process parameters
```

```
run P02_set_laser_params.m

T_bed = 298;    % K

T_init = 298;    % K
T_inf = 298;     % K

% laser_power = 50;   % W
% laser_radius = 300e-6; % micron -> m

ref_flux_magnitude = 1;% W/m^2


% estimated for heat transfer from a vertical flat plate,
% assuming constant heat flux, and calculating property values
% at 950 K (= (300 + 1600)/2), where 1600 K  is a little below the
melting
% point of stainless steel.
%%%Heat_transfer_coefficient = 10;    % W/(m^2.K)


%% Time parameters


% For removing material
initial_time_inc_Stage1 = 1e-3;
duration_Stage1 = 1e-3;

% For heating one layer of material
initial_time_inc_Stage2 = 0.01;

scan_speed = 0.05 ; % m/sec

del_t = mesh.del_x / scan_speed;



% For cooling material
initial_time_inc_Stage3 = 0.01;



%% Field variables for powder and bulk

FV.powder = 0;
FV.bulk = 1;


%% Trace and scan perimeter

disp('Scanning Perimeter ...')
```

```
run P03a_trace_perimeter.m
run P03b_perimeter_scan.m


%% Scan interior

disp('Scanning Interior ...')

run P04_random_scan.m

% from which we can now obtain these variables
duration_Stage2_per_layer=random_scan.flux_ampl_data(end,1);
duration_Stage3 = duration_Stage2_per_layer * mesh.layers_considered;



%% Generate data for heat flux corresponding to perimeter+interior scan


top_surf_flux_ampl_data = [ ...
    perim_scan.flux_ampl_data;  random_scan.flux_ampl_data
    ];



%% Write heat flux data to files

% Call a script to write Ampl lines
disp('Writing Ampl data files ...')
run P05b_write_ampl_data_files.m

% Call a script to write Dflux lines
disp('Writing DFlux data files ...')
run P05c_write_DFlux_lines.m




%% ***** Write .inp file *****
disp('Writing input file ...')

%% Open file and print heading

fid_aba = fopen('00_Additive_mfg.inp', 'w');

fprintf(fid_aba, '%s \r\n', '*HEADING');
fprintf(fid_aba, '%s \r\n', 'Simulation of thermal distortion and
stress in additive manufacturing');


%% Nodes

fprintf(fid_aba, '%s \r\n', '****');
fprintf(fid_aba, '%s \r\n', '**');

filename = '01_Nodes.inp';
fprintf(fid_aba, '%s \r\n', '*NODE, NSET = Nset_All');
fprintf(fid_aba, '%s%s \r\n', '*INCLUDE, INPUT = ', filename);

fprintf(fid_aba, '%s \r\n', '**');
fprintf(fid_aba, '%s \r\n', '****');


%% Elements

fprintf(fid_aba, '%s \r\n', '****');
fprintf(fid_aba, '%s \r\n', '**');

filename = '02_Elements.inp';
fprintf(fid_aba, '%s \r\n', '*ELEMENT, ELSET = Elset_All, TYPE =
C3D8RT');
fprintf(fid_aba, '%s%s \r\n', '*INCLUDE, INPUT = ', filename);

fprintf(fid_aba, '%s \r\n', '**');
fprintf(fid_aba, '%s \r\n', '****');


%% Node set for bottom layer

fprintf(fid_aba, '%s \r\n', '****');
fprintf(fid_aba, '%s \r\n', '**');

cmd_line = ['*NSET, NSET = ', mesh.Nset_bottom.name, ', GENERATE'];
fprintf(fid_aba, '%s \r\n', cmd_line);
fprintf(fid_aba, '%5i, %5i, %1i \r\n', mesh.Nset_bottom.first_node, ...
    mesh.Nset_bottom.last_node, 1);

fprintf(fid_aba, '%s \r\n', '**');
fprintf(fid_aba, '%s \r\n', '****');


%% Element sets for xy-layers

fprintf(fid_aba, '%s \r\n', '****');
fprintf(fid_aba, '%s \r\n', '**');

% Element sets for layers
filename = '03_Elsets_layers.inp';
fid_elsets = fopen(filename, 'w');

for k = 1 : mesh.num_elem_z

    cmd_line = ['*ELSET, ELSET = ', mesh.Elset_layer_array(k).name, ',
GENERATE'];
    fprintf(fid_elsets, '%s \r\n', cmd_line);
    fprintf(fid_elsets, '%5i, %5i, %1i \r\n',
mesh.Elset_layer_array(k).first_elem, ...
        mesh.Elset_layer_array(k).last_elem, 1);
```

```
end
clear k  cmd_line
fclose(fid_elsets);

% Include the Elsets file in the main file
cmd_line = ['*INCLUDE, INPUT=' filename];
fprintf(fid_aba, '%s \r\n', cmd_line);

fprintf(fid_aba, '%s \r\n', '**');
fprintf(fid_aba, '%s \r\n', '****');


%% Node sets for xy-layers

fprintf(fid_aba, '%s \r\n', '****');
fprintf(fid_aba, '%s \r\n', '**');

% Node sets for layers
filename = '04_Nsets_layers.inp';
fid_Nsets = fopen(filename, 'w');

for k = 1 : mesh.num_elem_z
    Nset_name = mesh.Nset_layer_array(k,1).name;
    Elset_name = mesh.Nset_layer_array(k,1).Elset;

    cmd_line = ['*NSET, NSET = ', Nset_name, ', ELSET = ', Elset_name];
    fprintf(fid_Nsets, '%s \r\n', cmd_line);
end
clear k  cmd_line
clear Nset_name  Elset_name
fclose(fid_Nsets);

% Include the Nsets file in the main file
cmd_line = ['*INCLUDE, INPUT=' filename];
fprintf(fid_aba, '%s \r\n', cmd_line);

fprintf(fid_aba, '%s \r\n', '**');
fprintf(fid_aba, '%s \r\n', '****');


%% Solid section

fprintf(fid_aba, '%s \r\n', '****');
fprintf(fid_aba, '%s \r\n', '**');
fprintf(fid_aba, '%s \r\n', '*SOLID SECTION, ELSET = Elset_All,
MATERIAL = Ti-6Al-4V');
fprintf(fid_aba, '%s \r\n', '**');
fprintf(fid_aba, '%s \r\n', '****');


%% Material

% Unless otherwise stated, properties used are:
% Properties of Titanium Ti-6Al-4V (Grade 5), Annealed
% From MatWeb, www.matweb.com, accessed January 26, 2017
% Note the melting point is not specified in the model.
% MatWeb gives it as 1604 - 1660 deg. C.
fprintf(fid_aba, '%s \r\n', '****');
fprintf(fid_aba, '%s \r\n', '**');
fprintf(fid_aba, '%s \r\n', '*MATERIAL, NAME = Ti-6Al-4V');
fprintf(fid_aba, '%s \r\n', '*DENSITY');
fprintf(fid_aba, '%s \r\n', '4430'); % kg/m^3
fprintf(fid_aba, '%s \r\n', '*ELASTIC');
fprintf(fid_aba, '%s \r\n', '113.8E9, 0.342'); % E in Pa, nu

% For temperature from 20 to 950 deg. C
fprintf(fid_aba, '%s \r\n', '*EXPANSION');
% BELOW IS INCORRECT VALUE, used to test and evaluate how code is
working
%fprintf(fid_aba, '%s \r\n', '9.7E-3'); % m/(m.degC)

% BELOW IS THE CORRECT VALUE, ABOVE IS DIFFERENT VALUE TO TEST
fprintf(fid_aba, '%s \r\n', '9.7E-6'); % m/(m.degC)

% Source of data for thermal conductivity and specific heat:
%{
3-DIMENSIONAL FINITE ELEMENT MODELING OF SELECTIVE LASER
MELTING TI-6AL-4V ALLOY
C.H. Fu, Y.B. Guo

https://sffsymposium.engr.utexas.edu/sites/default/files/2014-089-
Fu.pdf
accessed May 12, 2017
%}

% 1/2 CONDUCTIVITY
%{
fprintf(fid_aba, '%s \r\n', '*CONDUCTIVITY, DEPENDENCIES=1');
% for powder
fprintf(fid_aba, '%s%s \r\n', ' 0.1,  293, ', num2str(FV.powder)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '9.7, 1878, ', num2str(FV.powder)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '14.15, 1928, ', num2str(FV.powder)); %
W/m.K
fprintf(fid_aba, '%s \r\n', '**');
% for bulk
fprintf(fid_aba, '%s%s \r\n', ' 3.6,  299.85, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', ' 4.075,  373.00, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', ' 4.72,  473.00, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '6.66,  773.00, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '9.10, 1149.85, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '9.895, 1273.00, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '13.13, 1773.00, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '14.135, 1928.00, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '18.50, 2399.00, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '21.00, 2699.85, ', num2str(FV.bulk)); %
W/m.K
%}

%{
% DOUBLE CONDUCTIVITY
fprintf(fid_aba, '%s \r\n', '*CONDUCTIVITY, DEPENDENCIES=1');
% for powder
fprintf(fid_aba, '%s%s \r\n', ' 0.4,  293, ', num2str(FV.powder)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '38.8, 1878, ', num2str(FV.powder)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '56.5, 1928, ', num2str(FV.powder)); %
W/m.K
fprintf(fid_aba, '%s \r\n', '**');
% for bulk
fprintf(fid_aba, '%s%s \r\n', ' 14.4,  299.85, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', ' 16.3,  373.00, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', ' 18.88,  473.00, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '26.64,  773.00, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '36.4, 1149.85, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '39.58, 1273.00, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '52.52, 1773.00, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '56.54, 1928.00, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '74.00, 2399.00, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '84.00, 2699.85, ', num2str(FV.bulk)); %
W/m.K
%}


% CORRECT CODE FOR CONDUCTIVITY - COMMENTED OUT FOR DIFFERENT TRIAL
fprintf(fid_aba, '%s \r\n', '*CONDUCTIVITY, DEPENDENCIES=1');
% for powder
fprintf(fid_aba, '%s%s \r\n', ' 0.2,  293, ', num2str(FV.powder)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '19.4, 1878, ', num2str(FV.powder)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '28.3, 1928, ', num2str(FV.powder)); %
W/m.K
fprintf(fid_aba, '%s \r\n', '**');
% for bulk
fprintf(fid_aba, '%s%s \r\n', ' 7.20,  299.85, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', ' 8.15,  373.00, ', num2str(FV.bulk)); %
W/m.K
```

```
fprintf(fid_aba, '%s%s \r\n', ' 9.44,  473.00, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '13.32,  773.00, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '18.20, 1149.85, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '19.79, 1273.00, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '26.26, 1773.00, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '28.27, 1928.00, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '37.00, 2399.00, ', num2str(FV.bulk)); %
W/m.K
fprintf(fid_aba, '%s%s \r\n', '42.00, 2699.85, ', num2str(FV.bulk)); %
W/m.K
%{
Table data
 7.20 26.85
 8.15 100.00
 9.44 200.00
13.32 500.00
18.20 876.85
19.79 1000.00
26.26 1500.00
28.27 1655.00
37.00 2126.00
42.00 2426.85
%}
%}

fprintf(fid_aba, '%s \r\n', '*SPECIFIC HEAT');
% fprintf(fid_aba, '%s \r\n', '526.3'); % J/kg.K
fprintf(fid_aba, '%s \r\n', ' 580,  293'); % J/kg.K
fprintf(fid_aba, '%s \r\n', ' 610,  478'); % J/kg.K
fprintf(fid_aba, '%s \r\n', ' 670,  698'); % J/kg.K
fprintf(fid_aba, '%s \r\n', ' 760,  923'); % J/kg.K
fprintf(fid_aba, '%s \r\n', ' 930, 1143'); % J/kg.K
fprintf(fid_aba, '%s \r\n', ' 936, 1273'); % J/kg.K
fprintf(fid_aba, '%s \r\n', '1016, 1473'); % J/kg.K
fprintf(fid_aba, '%s \r\n', '1095, 1673'); % J/kg.K
fprintf(fid_aba, '%s \r\n', '1126, 1928'); % J/kg.K
%{
Table data:
580 20
610 205
670 425
760 650
930 870
936 1000
1016 1200
1095 1400
1126 1655
%}

%{
Source of latent heat value:
```

```
https://materialsdata.nist.gov/dspace/xmlui/bitstream/handle/11115/166/
Thermophysical%20Properties.pdf?sequence=3
accessed June 6, 2017

Using solidus and liquidus temperatures defined as part of conductivity
data, for consistency
%}
% CORRECT LATENT VALUE = 286
% TEST LATENT VALUE = 500
fprintf(fid_aba, '%s \r\n', '*LATENT HEAT');
fprintf(fid_aba, '%4i, %4i, %4i \r\n', 500, 1878, 1928); % W/m.K

fprintf(fid_aba, '%s \r\n', '**');
fprintf(fid_aba, '%s \r\n', '****');


%% Amplitude

fprintf(fid_aba, '%s \r\n', '** Include data for amplitude of heat flux
to each element');
% filename =
'scratch1/cvbuck/scans/heat_flux/ampl/00_ampl_inc_file.inp';
filename = '/scratch1/cvbuck/dmp2/00_ampl_inc_file.inp';
% filename = './heat_flux/ampl/00_ampl_inc_file.inp';
cmd_line = ['*INCLUDE, INPUT=', filename];
fprintf(fid_aba, '%s \r\n', cmd_line);

fprintf(fid_aba, '%s \r\n', '**');
fprintf(fid_aba, '%s \r\n', '** Include data for amplitude of zero flux
to a layer');
cmd_line = '*AMPLITUDE, NAME=AMP_Zero';
fprintf(fid_aba, '%s \r\n', cmd_line);

begin_time = 0;
end_time = initial_time_inc_Stage2;
fprintf(fid_aba, '%5i,%5i,%9.5f,%5i \r\n', begin_time, 0, end_time, 0);


%% Initial Conditions

%{
The melting point of AISI 304 stainless steel ranges from 1400 to 1455
deg.
C, according to MatWeb,
http://www.matweb.com/search/DataSheet.aspx?MatGUID=abc4415b0f8b490387e
3c922237098da&ckck=1
acessed January 21, 2017.
%}
fprintf(fid_aba, '%s \r\n','****');
fprintf(fid_aba, '%s \r\n','**');
fprintf(fid_aba, '%s \r\n','*INITIAL CONDITION, TYPE=TEMPERATURE');
fprintf(fid_aba, '%s%s \r\n','Nset_All, ', num2str(T_init));

fprintf(fid_aba, '%s \r\n','**');
fprintf(fid_aba, '%s \r\n','*INITIAL CONDITION, TYPE=FIELD');
fprintf(fid_aba, '%s%s \r\n','Nset_All,',num2str(FV.powder));
fprintf(fid_aba, '%s \r\n','**');
fprintf(fid_aba, '%s \r\n','****');


%% Stage 1 - Remove material

fprintf(fid_aba, '%s \r\n', '****');
fprintf(fid_aba, '%s \r\n', '**');

fprintf(fid_aba, '%s \r\n', '*STEP, NAME=Material_Removal_Step');
fprintf(fid_aba, '%s \r\n', '*COUPLED TEMPERATURE-DISPLACEMENT');
fprintf(fid_aba, '%12.8f,%12.8f \r\n', initial_time_inc_Stage1,
duration_Stage1);
fprintf(fid_aba, '%s \r\n', '*MODEL CHANGE, REMOVE');
fprintf(fid_aba, '%s \r\n', 'Elset_All');
fprintf(fid_aba, '%s \r\n', '*END STEP');

fprintf(fid_aba, '%s \r\n', '**');
fprintf(fid_aba, '%s \r\n', '****');




%% Stage 2 - Add material layer by layer, heating the top each time


% Apply heat flux  starting at x_min, y_min
fprintf(fid_aba, '%s \r\n', '****');
fprintf(fid_aba, '%s \r\n', '**');

% Separate input files included for heating each layer
for current_layer = 1 : mesh.layers_considered
    %% Add this layer to the model
    cmd_line = ['*STEP, INC=35000, NAME = Adding Layer ', ...
        num2str(current_layer)];
    fprintf(fid_aba, '%s \r\n', cmd_line);

    fprintf(fid_aba, '%s \r\n', '*COUPLED TEMPERATURE-DISPLACEMENT');
    fprintf(fid_aba, '%12.8f,%12.8f \r\n', initial_time_inc_Stage2,
initial_time_inc_Stage2);

    fprintf(fid_aba, '%s \r\n', '*MODEL CHANGE, ADD');
    fprintf(fid_aba, '%s \r\n',
mesh.Elset_layer_array(current_layer).name);


    % Fix nodes at and near corner of bottom surface
    if (current_layer == 1)
        % Set bottom of first layer to bed temperature (commented)
        fprintf(fid_aba, '%s \r\n', '*BOUNDARY');
        fprintf(fid_aba, '%s,%5i,%5i,%10i \r\n', mesh.Nset_bottom.name,
11, 11, T_inf);
```

```
        % Using an idea from following reference to minimize
translation
        % and rotation of part:
        % On the bottom face, fix a corner node in x,y,z.
        % Fix its nearest neighbor along 'x' in y,z.
        % Fix its nearest neighbor along 'y' in x,z.
        %
        % REFERENCE:
        %{
        K. Dai and L. Shaw,
        "Distortion minimization of laser-processed components
        through control of laser scanning patterns",
        Rapid Prototyping Journal, 8(5), 270 276, 2002.
        %}

        fprintf(fid_aba, '%s \r\n', '*BOUNDARY, TYPE=DISPLACEMENT');
        fprintf(fid_aba, '%5i,%5i,%5i,%5i \r\n', corner_node, 1, 3, 0);
        fprintf(fid_aba, '%5i,%5i,%5i,%5i \r\n', ...
            corner_node_neighbor_x, 2, 3, 0);
        fprintf(fid_aba, '%5i,%5i,%5i,%5i \r\n', ...
            corner_node_neighbor_y, 1, 1, 0);
        fprintf(fid_aba, '%5i,%5i,%5i,%5i \r\n', ...
            corner_node_neighbor_y, 3, 3, 0);

        fprintf(fid_aba, '%s \r\n','*OUTPUT, FIELD, NUMBER
INTERVAL=1');
        fprintf(fid_aba, '%s \r\n','*ELEMENT OUTPUT');
        fprintf(fid_aba, '%s \r\n','EVOL');
    end

    % When adding a layer past the first,
    % ensure the flux on the top surface of the previous layer
    % is now zero.
    if (current_layer > 1)
        cmd_line = '*DFLUX, OP=NEW, Amplitude=AMP_Zero';
        fprintf(fid_aba, '%s \r\n', cmd_line);
        data_line = [ ...
            mesh.Elset_layer_array(current_layer-1).name,', S2, ', ...
              num2str(ref_flux_magnitude) ...
            ];
        fprintf(fid_aba, '%s \r\n', data_line);


        % Reset field variable so that previous layer
        % now has bulk properties
        cmd_line = '*FIELD, OP=MOD';
        fprintf(fid_aba, '%s \r\n', cmd_line);

        Nset_name = mesh.Nset_layer_array(current_layer-1).name;
        data_line = [ ...
            Nset_name,', ', num2str(FV.bulk) ...
            ];
        fprintf(fid_aba, '%s \r\n', data_line);


        % Check that heat flux on top surface of layer just scanned is
now zero
```

```
        fprintf(fid_aba, '%s \r\n','*OUTPUT, FIELD, NUMBER
INTERVAL=1');
        fprintf(fid_aba, '%s \r\n','*ELEMENT OUTPUT');
        fprintf(fid_aba, '%s \r\n','FLUXS, FV, EVOL');
        fprintf(fid_aba, '%s \r\n','**');

    end

    fprintf(fid_aba, '%s \r\n', '*END STEP');
    fprintf(fid_aba, '%s \r\n', '**');
    fprintf(fid_aba, '%s \r\n', '****');


    %% Apply heat flux using random scanning pattern

    fprintf(fid_aba, '%s \r\n', '****');
    fprintf(fid_aba, '%s \r\n', '**');

    cmd_line = ['*STEP, INC=35000, NAME = Scanning Layer ', ...
        num2str(current_layer)];
    fprintf(fid_aba, '%s \r\n', cmd_line);

    fprintf(fid_aba, '%s \r\n', '*COUPLED TEMPERATURE-DISPLACEMENT');
    fprintf(fid_aba, '%12.8f,%12.8f \r\n', initial_time_inc_Stage2,
duration_Stage2_per_layer);

    filename
=['/scratch1/cvbuck/dmp2/DFlux_Layer_',num2str(current_layer),'.inp'];
    % filename =
['scratch1/cvbuck/Scans/heat_flux/DFlux/DFlux_Layer_',num2str(current_l
ayer), '.inp'];
    % filename =
['./heat_flux/DFlux/DFlux_Layer_',num2str(current_layer), '.inp'];
    cmd_line = ['*INCLUDE, INPUT=', filename];
    fprintf(fid_aba, '%s \r\n', cmd_line);

    % Convective heat transfer from sides

%     fprintf(fid_aba, '%s \r\n', '*FILM, OP=NEW');
%
%     if (current_layer == 1)
%         fprintf(fid_aba, '%s,%10s,%10i,%10i \r\n', ...
%             mesh.Elsets_side_y_min(1,1).name, ...
%             'F3', T_inf, Heat_transfer_coefficient);
%
%         fprintf(fid_aba, '%s,%10s,%10i,%10i \r\n', ...
%             mesh.Elsets_side_y_max(1,1).name, ...
%             'F5', T_inf, Heat_transfer_coefficient);
%
%         fprintf(fid_aba, '%s,%10s,%10i,%10i \r\n', ...
%             mesh.Elsets_side_x_min(1,1).name, ...
%             'F6', T_inf, Heat_transfer_coefficient);
%
%         fprintf(fid_aba, '%s,%10s,%10i,%10i \r\n', ...
%             mesh.Elsets_side_x_max(1,1).name, ...
%             'F4', T_inf, Heat_transfer_coefficient);
%     else
%         fprintf(fid_aba, '%s,%10s,%10i,%10i \r\n', ...
%             mesh.Elsets_names_side_y_min_combined{current_layer-1,
1}, ...
%             'F3', T_inf, Heat_transfer_coefficient);
%
%         fprintf(fid_aba, '%s,%10s,%10i,%10i \r\n', ...
%             mesh.Elsets_names_side_y_max_combined{current_layer-1,
1}, ...
%             'F5', T_inf, Heat_transfer_coefficient);
%
%         fprintf(fid_aba, '%s,%10s,%10i,%10i \r\n', ...
%             mesh.Elsets_names_side_x_min_combined{current_layer-1,
1}, ...
%             'F6', T_inf, Heat_transfer_coefficient);
%
%         fprintf(fid_aba, '%s,%10s,%10i,%10i \r\n', ...
%             mesh.Elsets_names_side_x_max_combined{current_layer-1,
1}, ...
%             'F4', T_inf, Heat_transfer_coefficient);
%     end


    % Output
    fprintf(fid_aba, '%s \r\n','*OUTPUT, FIELD, NUMBER INTERVAL=100');
    fprintf(fid_aba, '%s \r\n','*NODE OUTPUT');
    fprintf(fid_aba, '%s \r\n','U,NT');

    fprintf(fid_aba, '%s \r\n','*ELEMENT OUTPUT');
    fprintf(fid_aba, '%s \r\n','S, PEEQ, THE, TEMP, EVOL');

    fprintf(fid_aba, '%s \r\n','*OUTPUT, HISTORY, NUMBER INTERVAL=2');
    fprintf(fid_aba, '%s \r\n','*NODE OUTPUT, NSET = Nset_All');
    fprintf(fid_aba, '%s \r\n','U');




    %% Write restart file if scanning last layer
    if (current_layer == mesh.layers_considered)
        cmd_line = '*RESTART, WRITE, NUMBER INTERVAL=2, TIME
MARKS=YES';
        fprintf(fid_aba, '%s \r\n', cmd_line);
    end


    %% End scanning step

    fprintf(fid_aba, '%s \r\n', '*END STEP');
    fprintf(fid_aba, '%s \r\n', '**');
    fprintf(fid_aba, '%s \r\n', '****');

end
clear cmd_line
```

```
%% Still stage 2 - Zero the flux on the top surface of the last layer
...
% ... and set field for last layer
fprintf(fid_aba, '%s \r\n', '****');
fprintf(fid_aba, '%s \r\n', '**');
cmd_line = '*STEP, INC=35000, NAME = Set Flux Zero';
fprintf(fid_aba, '%s \r\n', cmd_line);

fprintf(fid_aba, '%s \r\n', '*COUPLED TEMPERATURE-DISPLACEMENT');
fprintf(fid_aba, '%12.8f,%12.8f \r\n', initial_time_inc_Stage2,
initial_time_inc_Stage2);

cmd_line = '*DFLUX, OP=NEW, Amplitude=AMP_Zero';
fprintf(fid_aba, '%s \r\n', cmd_line);
Elset_name = mesh.Elset_layer_array(current_layer).name;
data_line = [ ...
    Elset_name,', S2, ', ...
    num2str(ref_flux_magnitude) ...
    ];
fprintf(fid_aba, '%s \r\n', data_line);

% Reset field variable so that last layer
% now has bulk properties
cmd_line = '*FIELD, OP=MOD';
fprintf(fid_aba, '%s \r\n', cmd_line);

Nset_name = mesh.Nset_layer_array(current_layer).name;
data_line = [ ...
    Nset_name,', ', num2str(FV.bulk) ...
    ];
fprintf(fid_aba, '%s \r\n', data_line);

% Check that heat flux on top surface of layer just scanned is now zero
fprintf(fid_aba, '%s \r\n','*OUTPUT, FIELD, NUMBER INTERVAL=1');
% fprintf(fid_aba, '%s%s \r\n','*ELEMENT OUTPUT, ELSET=', Elset_name);
fprintf(fid_aba, '%s \r\n','*ELEMENT OUTPUT');
clear Elset_name
fprintf(fid_aba, '%s \r\n','FLUXS, FV');
fprintf(fid_aba, '%s \r\n','**');

fprintf(fid_aba, '%s \r\n', '*END STEP');
fprintf(fid_aba, '%s \r\n', '**');
fprintf(fid_aba, '%s \r\n', '****');

clear cmd_line  data_line



%% Stage 3 - Allow the part to stand and cool

fprintf(fid_aba, '%s \r\n', '****');
fprintf(fid_aba, '%s \r\n', '**');

cmd_line = '*STEP, INC=35000, NAME = Cooling Part';
fprintf(fid_aba, '%s \r\n', cmd_line);
fprintf(fid_aba, '%s \r\n', '*COUPLED TEMPERATURE-DISPLACEMENT');
fprintf(fid_aba, '%12.8f,%12.8f \r\n', initial_time_inc_Stage3,
duration_Stage3);

% Output
fprintf(fid_aba, '%s \r\n','*OUTPUT, FIELD, NUMBER INTERVAL=100');
fprintf(fid_aba, '%s \r\n','*NODE OUTPUT');
fprintf(fid_aba, '%s \r\n','U,NT');

fprintf(fid_aba, '%s \r\n','*ELEMENT OUTPUT');
fprintf(fid_aba, '%s \r\n','S, PEEQ, THE, TEMP, EVOL');

fprintf(fid_aba, '%s \r\n','*OUTPUT, HISTORY, NUMBER INTERVAL=1');
fprintf(fid_aba, '%s \r\n','*NODE OUTPUT, NSET = Nset_All');
fprintf(fid_aba, '%s \r\n','U');
%%%% THESE THREE LINES WERE ADDED on 10/29/19 TO TRY TO GENERATE.FIL
for runs
fprintf(fid_aba, '%s \r\n', '*FILE FORMAT, ASCII');
fprintf(fid_aba, '%s \r\n', '*NODE FILE, NSET=Nset_Layer_1');
% change to output on 11/11 to try to fix memory issue
fprintf(fid_aba, '%s \r\n', 'NT'); % if testing nodal temps
% fprintf(fid_aba, '%s \r\n', 'U'); if testing displacement
% fprintf(fid_aba, '%s \r\n', 'RF, U');

% Write restart file
cmd_line = '*RESTART, WRITE, NUMBER INTERVAL=2, TIME MARKS=YES';
fprintf(fid_aba, '%s \r\n', cmd_line);

fprintf(fid_aba, '%s \r\n', '*END STEP');
fprintf(fid_aba, '%s \r\n', '**');
fprintf(fid_aba, '%s \r\n', '****');

clear cmd_line


%% Close file

fclose(fid_aba);
disp('*** Input file written ***')
```

<u>P01_generate_mesh.m</u>

```
%% Flag

view_2d_mesh = 0;
view_3d_mesh = 0;


%% Geometrical and Mesh parameters


% So, when it comes to the interior block scan, I want to have 2^4
element
% rows in y direction.  So that is 16 interior + 2 that will be in
% perimeter.

% Coarser mesh 10 mm
%{
mesh.del_x = 1e-2;
mesh.del_y = 1e-2;
mesh.del_z = 1e-2;

mesh.num_elem_z = 10;
mesh.num_elem_y = 10;
mesh.num_elem_x_with_fin = 20;
mesh.num_elem_x_without_fin = 10;

mesh.elem_rows = 1 : mesh.num_elem_y;
mesh.elem_rows_with_fin = [5, 6];
mesh.elem_rows_without_fin = ...
    setdiff(mesh.elem_rows, mesh.elem_rows_with_fin);
%}

% Finer mesh 5mm

mesh.del_x = 5e-3;
mesh.del_y = 5e-3;
mesh.del_z = 5e-3;

mesh.num_elem_z=9;
mesh.num_elem_y=18;
mesh.num_elem_x_with_fin=36;
mesh.num_elem_x_without_fin=18;

mesh.elem_rows=1:mesh.num_elem_y;
mesh.elem_rows_with_fin=8:11;

mesh.elem_rows_without_fin = ...
    setdiff(mesh.elem_rows, mesh.elem_rows_with_fin);


% Finer mesh 2.5mm
%{
mesh.del_x = 2.5e-3;
mesh.del_y = 2.5e-3;
mesh.del_z = 2.5e-3;

mesh.num_elem_z=18;
mesh.num_elem_y=36;
mesh.num_elem_x_with_fin=72;
mesh.num_elem_x_without_fin=36;

mesh.elem_rows=1:mesh.num_elem_y;
mesh.elem_rows_with_fin=15:22;

mesh.elem_rows_without_fin = ...
    setdiff(mesh.elem_rows, mesh.elem_rows_with_fin);
%}
```

```
% Finer mesh 1mm
%{
mesh.del_x = 1e-3;
mesh.del_y = 1e-3;
mesh.del_z = 1e-3;

mesh.num_elem_z=45;
mesh.num_elem_y=90;
mesh.num_elem_x_with_fin=180;
mesh.num_elem_x_without_fin=90;

mesh.elem_rows=1:mesh.num_elem_y;
mesh.elem_rows_with_fin=35:55;

mesh.elem_rows_without_fin = ...
    setdiff(mesh.elem_rows, mesh.elem_rows_with_fin);
%}

% Finer mesh .5mm
%{
mesh.del_x = 5e-4;
mesh.del_y = 5e-4;
mesh.del_z = 5e-4;

mesh.num_elem_z=90;
mesh.num_elem_y=180;
mesh.num_elem_x_with_fin=360;
mesh.num_elem_x_without_fin=180;

mesh.elem_rows=1:mesh.num_elem_y;
mesh.elem_rows_with_fin=70:110;

mesh.elem_rows_without_fin = ...
    setdiff(mesh.elem_rows, mesh.elem_rows_with_fin);
%}

mesh.num_nodes_z = mesh.num_elem_z + 1;
mesh.num_nodes_y = mesh.num_elem_y + 1;
mesh.num_nodes_x_with_fin = mesh.num_elem_x_with_fin + 1;
mesh.num_nodes_x_without_fin = mesh.num_elem_x_without_fin + 1;

last_elem = mesh.elem_rows(end);
mesh.node_rows = [ mesh.elem_rows, last_elem+1 ];

last_elem = mesh.elem_rows_with_fin(end);
mesh.node_rows_with_fin = [ mesh.elem_rows_with_fin, last_elem+1 ];

mesh.node_rows_without_fin = ...
    setdiff(mesh.node_rows, mesh.node_rows_with_fin);
clear last_elem


num_elem_rows_with_fin = length(mesh.elem_rows_with_fin );
num_elem_rows_without_fin = length(mesh.elem_rows_without_fin);
mesh.num_elem_xy_plane = ...
    (num_elem_rows_with_fin * (mesh.num_elem_x_with_fin)) + ...
    (num_elem_rows_without_fin * mesh.num_elem_x_without_fin);


%% How many layers do we want to add?

mesh.layers_considered = min(1, mesh.num_elem_z);


%% Is defining parameters enough? Do we already have mesh files?
if (generate_mesh == false)
    return
end


%% Define an xy-plane of nodes as a template

mesh.Nodes = [];
num_node_rows_with_fin = length(mesh.node_rows_with_fin );
num_node_rows_without_fin = length(mesh.node_rows_without_fin);
mesh.num_nodes_xy_plane = ...
    (num_node_rows_with_fin * (mesh.num_nodes_x_with_fin)) + ...
    (num_node_rows_without_fin * mesh.num_nodes_x_without_fin);

Nodes_xy_plane = zeros(mesh.num_nodes_xy_plane, 3);
ctr = 0;
for j = 1 : mesh.num_nodes_y
    y = (j-1) * mesh.del_y;

    if (ismember(j, mesh.node_rows_without_fin))
        num_nodes_x_curr = mesh.num_nodes_x_without_fin;

    elseif (ismember(j, mesh.node_rows_with_fin))
        num_nodes_x_curr = mesh.num_nodes_x_with_fin;

    end

    for i = 1 : num_nodes_x_curr
        x = (i-1) * mesh.del_x;

        ctr = ctr+1;
        Nodes_xy_plane(ctr, :) = [ctr, x, y];
    end
end
clear ctr x y
clear i j


% View mesh
if (view_2d_mesh == true)
    fh = figure;
    set(fh, 'OuterPosition', get(0, 'ScreenSize'))
    ph = plot(Nodes_xy_plane(:,2), Nodes_xy_plane(:,3), 'ok');
    set(ph, 'LineWidth', line_width)
    grid on
end
```

```
%% Node set for bottom layer


mesh.Nset_bottom.name = 'Nset_bottom_surf';
mesh.Nset_bottom.first_node = 1;
mesh.Nset_bottom.last_node = mesh.num_nodes_xy_plane;


%% Generate 3D mesh of nodes using 2D template

num_nodes_total = mesh.num_nodes_xy_plane * (mesh.layers_considered +
1);
mesh.Nodes = zeros(num_nodes_total, 4);
for k = 1 : (mesh.num_elem_z + 1)
    start = ((k-1) * mesh.num_nodes_xy_plane) + 1;
    finish = k * mesh.num_nodes_xy_plane;

    mesh.Nodes(start : finish, 1:3) = Nodes_xy_plane;
    % Adjust node numbers
    mesh.Nodes(start : finish, 1) = ...
        mesh.Nodes(start : finish, 1) + ((k-1) *
mesh.num_nodes_xy_plane);
    mesh.Nodes(start : finish, 4) = ...
        mesh.Nodes(start : finish, 4) + ((k-1) * mesh.del_z);
end
clear k start finish

% View mesh
if (view_3d_mesh == true)
    fh = figure;
    set(fh, 'OuterPosition', get(0, 'ScreenSize'))
    ph = plot3(mesh.Nodes(:,2), mesh.Nodes(:,3), mesh.Nodes(:,4),
'ok');
    set(ph, 'LineWidth', line_width)
    grid on
    view(45, 45)
end


%% On the lower face, identify the corner node and its nearest
neighbors in x,y

corner_node = 1;
corner_node_neighbor_x = 2;
if (ismember(1, mesh.node_rows_without_fin))
    corner_node_neighbor_y = mesh.num_nodes_x_without_fin + 1;
elseif (ismember(1, mesh.node_rows_with_fin))
    corner_node_neighbor_y = mesh.num_nodes_x_with_fin + 1;
end


%% Write node data to file
```

```
filename = '01_Nodes.inp';
fid_nodes = fopen(filename, 'w');
fprintf(fid_nodes, '%s \r\n', '** NODE DATA');
fprintf(fid_nodes, '%10i,%12.8f,%12.8f,%12.8f \r\n', mesh.Nodes');
fclose(fid_nodes);


%% ***** Define elements *****

%% Define one xy-layer of elements

Elem_xy_plane = zeros(mesh.num_elem_xy_plane, 9);

% Define element-node connectivity for first element
curr_row = 1;
curr_col = 1;
if (ismember(curr_row, mesh.elem_rows_without_fin))
    num_nodes_curr_row = mesh.num_nodes_x_without_fin;
    num_elem_curr_row = mesh.num_elem_x_without_fin;
else
    num_nodes_curr_row = mesh.num_nodes_x_with_fin;
    num_elem_curr_row = mesh.num_elem_x_with_fin;
end

nodes = zeros(1,8);

nodes(1,1) = 1;
nodes(1,2) = 2;
nodes(1,3) = nodes(1,2) + num_nodes_curr_row;
nodes(1,4) = nodes(1,1) + num_nodes_curr_row;

nodes(1,5:8) = nodes(1,1:4) + mesh.num_nodes_xy_plane;
Elem_xy_plane(1, :) = [1, nodes];


% Define element-node connectivity for the remaining elements
for ctr = 2 : mesh.num_elem_xy_plane
    new_nodes = zeros(1,8);

    if (curr_col < num_elem_curr_row)

        curr_col = curr_col + 1;

        new_nodes(:) = nodes(:) + 1;


    elseif (curr_row < mesh.num_elem_y)

        prev_row = curr_row;
        curr_row = curr_row + 1;

        curr_col = 1;

        if (ismember(curr_row, mesh.elem_rows_with_fin))
            num_elem_curr_row = mesh.num_elem_x_with_fin;
        else
            num_elem_curr_row = mesh.num_elem_x_without_fin;

        end

        % Four cases:
        % 1. Current row has fin, previous row has not
        if (ismember(curr_row, mesh.elem_rows_with_fin)  && ...
                ismember(prev_row, mesh.elem_rows_without_fin))
            new_nodes(1:2) = nodes(1:2) + 2;
            num_nodes_curr_row = mesh.num_nodes_x_with_fin;
        % 2. Current row hasn't a fin, previous row has
        elseif (ismember(curr_row, mesh.elem_rows_without_fin)  && ...
                ismember(prev_row, mesh.elem_rows_with_fin))
            num_extra_nodes = ...
                mesh.num_nodes_x_with_fin - ...
                mesh.num_nodes_x_without_fin;
            new_nodes(1:2) = nodes(1:2) + 2; % + num_extra_nodes
            num_nodes_curr_row = mesh.num_nodes_x_with_fin;
        % 3. Neither current nor previous row has a fin
        elseif (ismember(curr_row, mesh.elem_rows_without_fin)  && ...
                ismember(prev_row, mesh.elem_rows_without_fin))
            before_prev_row = prev_row - 1;
            if ((before_prev_row > 0) && ...
                    (ismember(before_prev_row, mesh.elem_rows_with_fin
)))
                num_extra_nodes = ...
                    mesh.num_nodes_x_with_fin - ...
                    mesh.num_nodes_x_without_fin;
                new_nodes(1:2) = nodes(1:2) + 2 + num_extra_nodes;
            else
                new_nodes(1:2) = nodes(1:2) + 2;
            end

            num_nodes_curr_row = mesh.num_nodes_x_without_fin;

        elseif (ismember(curr_row, mesh.elem_rows_with_fin)  && ...
                ismember(prev_row, mesh.elem_rows_with_fin))
            new_nodes(1:2) = nodes(1:2) + 2;
            num_nodes_curr_row = mesh.num_nodes_x_with_fin;

        end

        new_nodes(3) = new_nodes(2) + num_nodes_curr_row;
        new_nodes(4) = new_nodes(1) + num_nodes_curr_row;

        new_nodes(5:8) = new_nodes(1:4) + mesh.num_nodes_xy_plane;

    else
        break
    end
    Elem_xy_plane(ctr, :) = [ctr, new_nodes];
    % Update 'nodes' to be ready for numbering the next element
    nodes = new_nodes;
end
```

```
clear ctr new_nodes
clear curr_row  prev_row  before_prev_row
clear num_nodes_curr_row  num_extra_nodes



%% Coordinates of centers of element faces on top surface

% x,y-coordinates of elements on top surface of deposited material
% These coordinates are the same for every layer
top_surf_center_data = zeros(mesh.num_elem_xy_plane, 3);

finish = mesh.num_elem_xy_plane;
top_surf_center_data(1 : finish, 1) = (1 : finish)';


for j = 1 : mesh.num_elem_xy_plane
    node = Elem_xy_plane(j,2);
    index = find(mesh.Nodes(:,1) == node);
    x_coord = mesh.Nodes(index,2) + (mesh.del_x/2);
    y_coord = mesh.Nodes(index,3) + (mesh.del_y/2);

    top_surf_center_data(j, 2:3) = [x_coord, y_coord];
end
clear j  node  index  x_coord  y_coord


filename = '00_top_surf_center_data.txt';
fid_surf = fopen(filename, 'w');
fprintf(fid_surf, '%12.8f,%12.8f,%12.8f \r\n', top_surf_center_data');
fclose(fid_surf);


%% Define element-node connectivity for complete mesh


num_elem_total = mesh.num_elem_xy_plane * mesh.layers_considered;

mesh.Elements = zeros(num_elem_total, 9);
for k = 1 : mesh.num_elem_z
    start = ((k-1) * mesh.num_elem_xy_plane) + 1;
    finish = k * mesh.num_elem_xy_plane;

    mesh.Elements(start : finish, :) = Elem_xy_plane;
    % Adjust element and node numbers
    mesh.Elements(start : finish, 1) = ...
        mesh.Elements(start : finish, 1) + ((k-1) *
mesh.num_elem_xy_plane);
    mesh.Elements(start : finish, 2:end) = ...
        mesh.Elements(start : finish, 2:end) + ((k-1) *
mesh.num_nodes_xy_plane);
end
clear k start finish
```

```
%% Define separate element sets for each layer

mesh.Elset_layer_array = struct([]);
for k = 1 : mesh.num_elem_z
    first_elem = ((k-1) * mesh.num_elem_xy_plane) + 1;
    last_elem = k * mesh.num_elem_xy_plane;

    mesh.Elset_layer_array(k,1).name = ['Elset_Layer_', num2str(k)];
    mesh.Elset_layer_array(k,1).first_elem = first_elem;
    mesh.Elset_layer_array(k,1).last_elem = last_elem;
end


%% Using element sets, define separate node sets for each layer

% Yes, there will be overlap between node sets
mesh.Nset_layer_array = struct([]);
for k = 1 : mesh.num_elem_z
    mesh.Nset_layer_array(k,1).name = ['Nset_Layer_', num2str(k)];
    mesh.Nset_layer_array(k,1).Elset = ['Elset_Layer_', num2str(k)];
end


%% Write element data to file

filename = '02_Elements.inp';
fid_elements = fopen(filename, 'w');
fprintf(fid_elements, '%s \r\n', '** ELEMENT DATA');
fprintf(fid_elements, '%10i,%10i,%10i,%10i,%10i,%10i,%10i,%10i,%10i
\r\n', mesh.Elements');
fclose(fid_elements);
```

P02_set_laser_params.m

```
% P02_set_laser_params.m

laser.power = 2500;   % W
laser.radius = mesh.del_x/2; % micron -> m

laser.x_min = mesh.del_x/2;
laser.y_min = mesh.del_y/2;

% This definition assumes that the laser will not end on the fin.
% In turn, the fin is assumed to have an even number of rows, and the
laser
% is assumed to start on the "main" part of the workpiece.
laser.y_max = (mesh.num_elem_y * mesh.del_y) - mesh.del_y/2;

% xmax with no fin
laser.x_max_no_fin = (mesh.num_elem_x_without_fin * mesh.del_x) - ...
    (mesh.del_x / 2);
```

```
% xmax with fin
laser.x_max_with_fin = (mesh.num_elem_x_with_fin * mesh.del_x) - ...
    (mesh.del_x / 2);
```

<u>P03a_trace_perimenter.m</u>

```
% P03a_trace_perimeter.m


%% For graphs

plot_perimeter_pts = false;
plot_moving_perim_pts = false;

%% create array of start and end positions for the laser to trace the
perimeter


% there are 3 cases to be considered here:
% (i) fin starts on the first y-row
% (ii) fin starts and ends in the middle
% (iii) fin ends on the last row



% y positions of the fin
y_fin=(mesh.elem_rows_with_fin * mesh.del_y) - mesh.del_y/2;

% we will assume that the laser trace start in first y-row

% if we start in the fin

x_end_pt(1) = laser.x_min;
y_end_pt(1) = laser.y_min;

% Case(i) fin starts on the first y-row
if mesh.elem_rows_with_fin(1) == 1

    x_end_pt(2)=laser.x_max_with_fin;
    y_end_pt(2)=y_end_pt(1);

    x_end_pt(3)=laser.x_max_with_fin;
    y_end_pt(3)=y_fin(end);

    x_end_pt(4)=laser.x_max_no_fin;
    y_end_pt(4)=y_fin(end);

    x_end_pt(5)=laser.x_max_no_fin;
    y_end_pt(5)=laser.y_max;

    x_end_pt(6)=laser.x_min;
    y_end_pt(6)=laser.y_max;
    x_end_pt(7)=laser.x_min;
    y_end_pt(7)=laser.y_min + mesh.del_y;



    % Case (ii) fin starts and ends in the middle
elseif mesh.elem_rows_with_fin(1) > 1 && mesh.elem_rows_with_fin(end) <
mesh.num_elem_y

    x_end_pt(2)=laser.x_max_no_fin;
    y_end_pt(2)=y_end_pt(1);

    x_end_pt(3)=laser.x_max_no_fin;
    y_end_pt(3)=y_fin(1);

    x_end_pt(4)=laser.x_max_with_fin;
    y_end_pt(4)=y_fin(1);

    x_end_pt(5)=laser.x_max_with_fin;
    y_end_pt(5)=y_fin(end);

    x_end_pt(6)=laser.x_max_no_fin;
    y_end_pt(6)=y_fin(end);

    x_end_pt(7)=laser.x_max_no_fin;
    y_end_pt(7)=laser.y_max;

    x_end_pt(8)=laser.x_min;
    y_end_pt(8)=laser.y_max;

    x_end_pt(9)=laser.x_min;
    y_end_pt(9)=laser.y_min + mesh.del_y;


    % Case (iii) fin ends on the last row
elseif mesh.elem_rows_with_fin(end) == mesh.num_elem_y

    x_end_pt(2)=laser.x_max_no_fin;
    y_end_pt(2)=y_end_pt(1);

    x_end_pt(3)=laser.x_max_no_fin;
    y_end_pt(3)=y_fin(1);

    x_end_pt(4)=laser.x_max_with_fin;
    y_end_pt(4)=y_fin(1);

    x_end_pt(5)=laser.x_max_with_fin;
    y_end_pt(5)=y_fin(end);

    x_end_pt(6)=laser.x_min;
    y_end_pt(6)=laser.y_max;

    x_end_pt(7)=laser.x_min;
    y_end_pt(7)=laser.y_min + mesh.del_y;

end


perimeter.xy = [x_end_pt', y_end_pt'];


%% Add intermediate points between each pair of end points

num_end_pts = size(perimeter.xy, 1);
perimeter.detailed_xy = [];

for i = 1 : (num_end_pts - 1)
    perimeter.detailed_xy = [perimeter.detailed_xy; perimeter.xy(i,:)];

    change_in_x = perimeter.xy(i+1,1) - perimeter.xy(i,1);
    change_in_y = perimeter.xy(i+1,2) - perimeter.xy(i,2);

    if (change_in_x ~= 0)
        num_inter_pts = abs(round(change_in_x / mesh.del_x)) - 1;
        inter_pts = zeros(num_inter_pts,2);
        if (change_in_x > 0)
            inter_pts(:,1) = perimeter.xy(i,1) + ((1 :
num_inter_pts)*mesh.del_x);

        else
            inter_pts(:,1) = perimeter.xy(i,1) - ((1 :
num_inter_pts)*mesh.del_x);

        end
        inter_pts(:,2) = perimeter.xy(i,2);
    elseif (change_in_y ~= 0)
        num_inter_pts = abs(round(change_in_y / mesh.del_y)) - 1;
        inter_pts = zeros(num_inter_pts,2);

        if (change_in_y > 0)
            inter_pts(:,2) = perimeter.xy(i,2) + ((1 :
num_inter_pts)*mesh.del_y);

        else
            inter_pts(:,2) = perimeter.xy(i,2) - ((1 :
num_inter_pts)*mesh.del_y);
        end
        inter_pts(:,1) = perimeter.xy(i,1);
    end
    perimeter.detailed_xy = [perimeter.detailed_xy; inter_pts];

end
clear i  change_in_x  change_in_y
clear inter_pts
perimeter.detailed_xy = [perimeter.detailed_xy; perimeter.xy(end,:)];

num_perim_elem = size(perimeter.detailed_xy,1);
```

```
%% Optional plotting
if plot_perimeter_pts
    fh = figure;
    set(fh, 'OuterPosition', get(0, 'ScreenSize'))
    ph = plot(perimeter.detailed_xy(:,1), perimeter.detailed_xy(:,2),
'b*');
    set(ph, 'LineWidth', 5)
    grid on
    axis equal

    hold on
end


if plot_moving_perim_pts
    fh = figure;
    set(fh, 'OuterPosition', get(0, 'ScreenSize'))

    for i = 1 : length(perimeter.detailed_xy)
        ph = plot(perimeter.detailed_xy(i,1),
perimeter.detailed_xy(i,2), 'b*');
        set(ph, 'LineWidth', 5)
        grid on
        hold on
        axis([laser.x_min - mesh.del_x, laser.x_max_with_fin +
mesh.del_x,...
            laser.y_min - mesh.del_y, laser.y_max + mesh.del_y])
        axis equal
        pause(0.1)
    end
end
```

<u>P03b_perimeter_scan.m</u>

```
% P03b_perimeter_scan.m

%% Procedure for moving laser:
%{
-- A path of elements covering the perimeter of the domain was defined
in
P03a_trace_perimeter.m
 - Use a loop over the number of elements in the path.
 - Before the loop, define the initial position and the row of the
laser.
%}

%% Laser position parameters

tol = mesh.del_x / 100;


%% Duration of scan
```

```
perim_scan.duration_per_layer = num_perim_elem * del_t;
perim_scan.time_array = 0 : del_t : perim_scan.duration_per_layer;

%% For storing heat flux amplitude data

% Array for amplitude of heat flux as a function of time for each
element
perim_scan.flux_ampl_data = ...
    zeros(length(perim_scan.time_array), mesh.num_elem_xy_plane + 1);
perim_scan.flux_ampl_data(:,1) = perim_scan.time_array';


%% Scan the top surface of the sample
time_ctr = 0;
% For each time step
for k = 2 : length(perim_scan.time_array)
    time_ctr = time_ctr + 1;

    laser.x_current = perimeter.detailed_xy(k-1,1);
    laser.y_current = perimeter.detailed_xy(k-1,2);

    % Define flux on top surface
    run P05a_heat_flux_data.m

    % Save the flux data
    % 'time_ctr' defined in calling script
    perim_scan.flux_ampl_data(time_ctr+1, 2:end) = ...
        perim_scan.flux_ampl_data(time_ctr+1, 2:end) + flux_data';
end
clear k
```

P04_random_scan.m

```
%% Procedure for moving laser:

plot_moving_scan=false;

%% Identify rows with and without fin
% Some rows have already been scanned as part of the perimeter. Those
% elements should not be scanned again. These are the rows that now
belong
% to the interior.

random_scan.rows_with_fin = mesh.elem_rows_with_fin(2 : end-1);

% If the fin starts on the first row
if (ismember(1, mesh.elem_rows_with_fin))
    random_scan.rows_without_fin = mesh.elem_rows_without_fin(1 : end-
1);
% If the fin ends on the last row
elseif (ismember(mesh.num_elem_y, mesh.elem_rows_with_fin))
    random_scan.rows_without_fin = mesh.elem_rows_without_fin(2 : end);
```

```
% If the fin is away from the first and last rows
else

    random_scan.rows_without_fin = [ ... %mesh.elem_rows_without_fin(2
: end);
        2 : mesh.elem_rows_with_fin(1), ...
        mesh.elem_rows_with_fin(end) : (mesh.num_elem_y-1) ...
        ];
end


random_scan.inter_rows=[random_scan.rows_with_fin,
random_scan.rows_without_fin];


%% Identify limits of laser movement for interior scan

% Note that the min and maxes here were previously defined for the
% perimeter scan

random_scan.x_min = laser.x_min + mesh.del_x;
random_scan.y_min = laser.y_min + mesh.del_y;

% xmax with no fin
random_scan.x_max_no_fin = laser.x_max_no_fin - mesh.del_x;
% xmax with fin
random_scan.x_max_with_fin = laser.x_max_with_fin - mesh.del_x;

random_scan.y_max = laser.y_max - mesh.del_y;


random_scan.x_length_no_fin=random_scan.x_max_no_fin-random_scan.x_min;
random_scan.x_length_fin=random_scan.x_max_with_fin -...
    (random_scan.x_max_no_fin+ mesh.del_x);

random_scan.num_elem = mesh.num_elem_xy_plane - num_perim_elem;


%% Set parameters for random start positions

% In the following paper
% B. Cheng, et al., Stress and deformation evaluations of scanning
strategy
% effect in selective laser melting, Addit Manuf (2016),
% http://dx.doi.org/10.1016/j.addma.2016.05.007
% there was mention of a technique called island scanning.  We can
borrow
% this idea of having our domain divided into certain number of regions
for
% the scanning paths to take place.

% For now our scanning will continue to always be in the x direction.

Nx_block_regions=1;
Nx_fin_regions=1;
```

```
del_x_block_reg=random_scan.x_length_no_fin/Nx_block_regions;
del_x_fin_reg=random_scan.x_length_fin/Nx_fin_regions;


Ny_block_regions=1;
Ny_fin_regions=1;


% Assuming we will not have something starting at the middle of a
region,
% only at the edges of a region

% for scanning in x, the total number of possible start positions, not
% including the fin, is

N_starts= (length(random_scan.inter_rows)*Nx_block_regions);  %+...
%     (length(random_scan.rows_with_fin)*Nx_fin_regions);


%% Create array of start and end positions

% We will create an array that has the start and end position for each
path
% that the laser could take in the interior of the block.

% So the columns are index xstart  ystart  xend  yend  rand

laser_start_end_array=zeros(N_starts,6);

% In addition, each path will be assigned a random number, so they can
be
% shuffled later.

rand_array=randperm(N_starts)';

ctr=0;
for i=1:length(random_scan.inter_rows)
    for j=1:Nx_block_regions
        ctr=ctr+1;
        rand=rand_array(ctr,1);

        xstart=random_scan.x_min + (j-1)*del_x_block_reg;
        xend=xstart+del_x_block_reg;

        ystart=random_scan.y_min + (i-1)*mesh.del_y;
        yend=ystart;


        laser_start_end_array(ctr,:)=[ctr, xstart, ystart, xend, yend,
rand];

    end
end
```

```
% Now, to "shuffle" the paths we will order them from least to greatest
by
% the rand column
% LINE BELOW FOR RANDOM
% sorted_laser_start_end_array=sortrows(laser_start_end_array,6);

%LINE BELOW FOR RASTER
sorted_laser_start_end_array=laser_start_end_array;

% Since for now the fin is so small we will just let it be scanned at
the
% end and " in order"
laser_fin_start_end_array=zeros(length(random_scan.rows_with_fin),6);

ctr=0;
index=size(sorted_laser_start_end_array,1);
for i=1:length(random_scan.rows_with_fin)
    for j=1:Nx_fin_regions
        ctr=ctr+1;
        rand=1;

        xstart=(random_scan.x_max_no_fin + mesh.del_x) + (j-
1)*del_x_fin_reg;
        xend=xstart+del_x_fin_reg;

        ystart=laser.y_min + (random_scan.rows_with_fin(1,i)-
1)*mesh.del_y;
        yend=ystart;

        index=index+1;
        laser_fin_start_end_array(ctr,:)=[index, xstart, ystart, xend,
yend, rand];

    end
end


sorted_laser_start_end_array=[sorted_laser_start_end_array; ...
    laser_fin_start_end_array];


save laser_scan_positions.txt   sorted_laser_start_end_array  -ascii



%% Duration of scan

random_scan.duration_per_layer = random_scan.num_elem * del_t;


start_time = perim_scan.time_array(end) + del_t;
```

```
end_time = perim_scan.time_array(end) + random_scan.duration_per_layer;

random_scan.time_array = ...
    start_time : del_t : end_time;


%% For storing heat flux amplitude data

% Array for amplitude of heat flux as a function of time for each
element
random_scan.flux_ampl_data = ...
    zeros(length(random_scan.time_array), mesh.num_elem_xy_plane + 1);



%% Scan the top surface of the sample

% 'time_ctr' updated in P03b_perimeter_scan.m

curr_time=perim_scan.time_array(end);

for k=1:size(sorted_laser_start_end_array,1)

    % Not sure if we need right now, but in case
    time_ctr = time_ctr + 1;

    curr_time=curr_time+del_t;


    index = time_ctr - num_perim_elem;
    random_scan.flux_ampl_data(index, 1)=curr_time;


    xstart=sorted_laser_start_end_array(k,2);
    ystart=sorted_laser_start_end_array(k,3);

    xend=sorted_laser_start_end_array(k,4);
    yend=sorted_laser_start_end_array(k,5);

    laser.x_current = xstart;
    laser.y_current = ystart;




    % Define flux on all elements in the top surface for the time
    % corresponding to the current laser position .
    run P05a_heat_flux_data.m

    % Save the flux data
    random_scan.flux_ampl_data(index, 2:end) = flux_data';


% optional plotting
```

```
if plot_moving_scan

        ph = plot(laser.x_current, laser.y_current, 'b*');
        set(ph, 'LineWidth', 5)
        grid on
        hold on
%         axis([laser.x_min - mesh.del_x, laser.x_max_with_fin +
mesh.del_x,...
%             laser.y_min - mesh.del_y, laser.y_max + mesh.del_y])
        axis equal
        pause(0.1)

end
%

    % if we didn't reach the end of the strip yet

    while laser.x_current <= xend

        % update time
        time_ctr = time_ctr + 1;
        index = time_ctr - num_perim_elem;

        curr_time=curr_time+del_t;
        random_scan.flux_ampl_data(index, 1)=curr_time;

        % update position
        laser.x_current = laser.x_current + mesh.del_x;


        run P05a_heat_flux_data.m
        random_scan.flux_ampl_data(index, 2:end) = flux_data';

        if plot_moving_scan

        ph = plot(laser.x_current, laser.y_current, 'b*');
        set(ph, 'LineWidth', 5)
        grid on
        hold on
%         axis([laser.x_min - mesh.del_x, laser.x_max_with_fin +
mesh.del_x,...
%             laser.y_min - mesh.del_y, laser.y_max + mesh.del_y])
        axis equal
        pause(0.1)

        end

    end


    disp('Exited while loop successfully')


end
```

P05a_heat_flux_data.m

```
%% Define heat flux history data for top surface


%% If needed, define a matrix containing top surface center data

if (generate_mesh == false)
    filename = '00_top_surf_center_data.txt';
    fid_surf = fopen(filename, 'r');
    data = textscan(fid_surf, '%12.8f,%12.8f,%12.8f');
    top_surf_center_data = [data{1}, data{2}, data{3}];
end
clear data


%% Amplitude data
% Write lines for *AMPLITUDE to separate files
% one file for each element


Cartesian_distance_data = [ ...
    top_surf_center_data(:,2) - laser.x_current, ...
    top_surf_center_data(:,3) - laser.y_current  ...
    ];
r = sqrt((Cartesian_distance_data(:,1).^2) + ...
    (Cartesian_distance_data(:,2).^2));

% Use the distance data to calculate flux on each element
% Source:
%{
        Eq. 5 in paper
        "Comparison of 3DSIM thermal modelling of selective laser
melting
        using new dynamic meshing method to ANSYS"
        K. Zeng, D. Pal, H. J. Gong, N. Patil and B. Stucker
        Materials Science and Technology 2015 VOL 31 NO 8 945-56

        laser_power = 50;   % W
        laser_radius = 300e-6; % micron -> m
%}
flux_fn = 2 * laser.power / (pi * (laser.radius^2));
flux_data = flux_fn * exp(-2 * (r.^2) / (laser.radius^2));
```

P05b_write_ampl_data.m

```
% write_ampl_data_files.m

%% Write Amplitude data files

% Write a single input file containing lines
% to include the heat-flux amplitude data
%filename = './heat_flux/ampl/00_ampl_inc_file.inp';
% THIS FILE PATH WAS CHANGED !!!
filename =
'/scratch1/cvbuck/Scans/heat_flux/ampl/00_ampl_inc_file.inp';
fid_ampl = fopen(filename, 'w');

% For each element
for i = 1 : mesh.num_elem_xy_plane

    element_number_str = num2str(top_surf_center_data(i,1));
    % Save the heat-flux data for the element to a separate array
    current_heat_flux_data = ...
        [top_surf_flux_ampl_data(:,1), top_surf_flux_ampl_data(:,i+1)];

    % Write separate files for the heat-flux amplitude data
    % for each element
    filename = ...
        ['/scratch1/cvbuck/dmp2/ampl_Element_', element_number_str,
'.inp'];
    fid_ht_fl = fopen(filename, 'w');

    cmd_line = ['*AMPLITUDE, NAME=AMP_Element_',element_number_str];
    fprintf(fid_ht_fl, '%s  \r\n',cmd_line);
    fprintf(fid_ht_fl, '%9.5f,%10.5f \r\n',current_heat_flux_data');

    fprintf(fid_ht_fl, '%s \r\n', '****');
    fprintf(fid_ht_fl, '%s \r\n', '****');

    % Write the line in the "including" file
    % to include the amplitude file for the element
    cmd_line = ['*INCLUDE, INPUT=', filename];
    fprintf(fid_ampl, '%s \r\n', cmd_line);
    fprintf(fid_ampl, '%s \r\n', '****');

    fclose(fid_ht_fl);
end
clear cmd_line
fclose(fid_ampl);
```

P05c_wrtie_DFlux_lines.m

```
%% DFLUX lines
```

```
% Write lines for *DFLUX to separate files
% one file for each layer
for k = 1 : mesh.num_elem_z
    % Open file for writing
%     filename = ['./heat_flux/DFlux/DFlux_Layer_',num2str(k), '.inp'];
%     filename =
['/scratch1/cvbuck/Scans/heat_flux/DFlux/DFlux_Layer_',num2str(k),
'.inp'];
    filename = ['/scratch1/cvbuck/dmp/DFlux_Layer_',num2str(k),
'.inp'];
    fid_ht_fl = fopen(filename, 'w');

    element_amp_numbers = top_surf_center_data(:,1);
    element_numbers = top_surf_center_data(:,1);
    finish = mesh.num_elem_xy_plane;
    element_numbers(1 : finish) = ...
        element_numbers(1 : finish) + ...
        (k-1) * mesh.num_elem_xy_plane;


    for i = 1 : size(top_surf_center_data, 1)


        element_amp_number_str = num2str(element_amp_numbers(i,1));
        element_number_str = num2str(element_numbers(i,1));
        % Write *DFLUX lines to file
        cmd_line = ['*DFLUX, Amplitude=AMP_Element_',
element_amp_number_str];
        fprintf(fid_ht_fl, '%s \r\n', cmd_line);
        data_line=[element_number_str,', S2, ',
num2str(ref_flux_magnitude)];
        fprintf(fid_ht_fl, '%s \r\n', data_line);

        if (mod(i,10) == 0)
            fprintf(fid_ht_fl, '%s \r\n', '****');
            fprintf(fid_ht_fl, '%s \r\n', '****');
        end
    end
    % Close file
    fclose(fid_ht_fl);
end
clear i  k
clear element_number
clear cmd_line  data_line
```

<u>00_Additive_mfg.inp</u>

```
*HEADING
Simulation of thermal distortion and stress in additive manufacturing
****
**
*NODE, NSET = Nset_All
*INCLUDE, INPUT = 01_Nodes.inp
**
****
```

```
****
**
*ELEMENT, ELSET = Elset_All, TYPE = C3D8RT
*INCLUDE, INPUT = 02_Elements.inp
**
****
****
**
*NSET, NSET = Nset_bottom_surf, GENERATE
    1,    451, 1
**
****
****
**
*INCLUDE, INPUT=03_Elsets_layers.inp
**
****
****
**
*INCLUDE, INPUT=04_Nsets_layers.inp
**
****
****
**
*SOLID SECTION, ELSET = Elset_All, MATERIAL = Ti-6Al-4V
**
****
****
**
*MATERIAL, NAME = Ti-6Al-4V
*DENSITY
4430
*ELASTIC
113.8E9, 0.342
*EXPANSION
9.7E-6
*CONDUCTIVITY, DEPENDENCIES=1
 0.2,  293, 0
19.4, 1878, 0
28.3, 1928, 0
**
 7.20,  299.85, 1
 8.15,  373.00, 1
 9.44,  473.00, 1
13.32,  773.00, 1
18.20, 1149.85, 1
19.79, 1273.00, 1
26.26, 1773.00, 1
28.27, 1928.00, 1
37.00, 2399.00, 1
42.00, 2699.85, 1
*SPECIFIC HEAT
 580,  293
 610,  478
 670,  698
 760,  923
 930, 1143
 936, 1273
```

```
1016, 1473
1095, 1673
1126, 1928
*LATENT HEAT
 500, 1878, 1928
**
****
** Include data for amplitude of heat flux to each element
*INCLUDE, INPUT=/scratch1/cvbuck/dmp2/00_ampl_inc_file.inp
**
** Include data for amplitude of zero flux to a layer
*AMPLITUDE, NAME=AMP_Zero
    0,    0,  0.01000,    0
****
**
*INITIAL CONDITION, TYPE=TEMPERATURE
Nset_All, 298
**
*INITIAL CONDITION, TYPE=FIELD
Nset_All,0
**
****
****
**
*STEP, NAME=Material_Removal_Step
*COUPLED TEMPERATURE-DISPLACEMENT
  0.00100000,  0.00100000
*MODEL CHANGE, REMOVE
Elset_All
*END STEP
**
****
****
**
*STEP, INC=35000, NAME = Adding Layer 1
*COUPLED TEMPERATURE-DISPLACEMENT
  0.01000000,  0.01000000
*MODEL CHANGE, ADD
Elset_Layer_1
*BOUNDARY
Nset_bottom_surf,   11,   11,       298
*BOUNDARY, TYPE=DISPLACEMENT
    1,    1,    3,    0
    2,    2,    3,    0
   20,    1,    1,    0
   20,    3,    3,    0
*OUTPUT, FIELD, NUMBER INTERVAL=1
*ELEMENT OUTPUT
EVOL
*END STEP
**
****
****
**
*STEP, INC=35000, NAME = Scanning Layer 1
*COUPLED TEMPERATURE-DISPLACEMENT
  0.01000000, 39.60000000
*INCLUDE, INPUT=/scratch1/cvbuck/dmp2/DFlux_Layer_1.inp
```

```
*OUTPUT, FIELD, NUMBER INTERVAL=100
*NODE OUTPUT
U,NT
*ELEMENT OUTPUT
S, PEEQ, THE, TEMP, EVOL
*OUTPUT, HISTORY, NUMBER INTERVAL=2
*NODE OUTPUT, NSET = Nset_All
U
*RESTART, WRITE, NUMBER INTERVAL=2, TIME MARKS=YES
*END STEP
**
****
****
**
*STEP, INC=35000, NAME = Set Flux Zero
*COUPLED TEMPERATURE-DISPLACEMENT
  0.01000000,  0.01000000
*DFLUX, OP=NEW, Amplitude=AMP_Zero
Elset_Layer_1, S2, 1
*FIELD, OP=MOD
Nset_Layer_1, 1
*OUTPUT, FIELD, NUMBER INTERVAL=1
*ELEMENT OUTPUT
FLUXS, FV
**
*END STEP
**
****
****
**
*STEP, INC=35000, NAME = Cooling Part
*COUPLED TEMPERATURE-DISPLACEMENT
  0.01000000, 39.60000000
*OUTPUT, FIELD, NUMBER INTERVAL=100
*NODE OUTPUT
U,NT
*ELEMENT OUTPUT
S, PEEQ, THE, TEMP, EVOL
*OUTPUT, HISTORY, NUMBER INTERVAL=1
*NODE OUTPUT, NSET = Nset_All
U
*FILE FORMAT, ASCII
*NODE FILE, NSET=Nset_Layer_1
NT
*RESTART, WRITE, NUMBER INTERVAL=2, TIME MARKS=YES
*END STEP
**
****
```

APPENDIX B. MATLAB SCRIPT FOR GENETIC ALGORITHM

Appendix B consists of the MATLAB scripts for the GA as originally written (i.e., not including the batch scripts for it to run on the Palmetto Cluster). There are 7 MATLAB scripts in total for the GA. These scripts include the generation of a random initial population, the tournament selection for selecting parents, the crossover and mutation for generating children, the fitness check of population members, and trimming the population back to the fittest individuals before the GA undergoes the next generation. Updated MATLAB scripts for the parallelization of the Abaqus runs on the Palmetto Cluster will be included in Appendix C: Batch Scripts for Palmetto Cluster GA Run.

GA_Maincode.m

```
clear all
clc
%% Genetic Algorithm - Main Code
% This Genetic Algorithm will be used to take any number of scanning
paths
% in DMLS and optimize the path to limit the deformations caused by
thermal
% gradients. The code links with Abaqus by writing the chosen path from
% each generation into the input file and running the simulation, and
then
% pulling out deformation results in CAE post-processing, and using
those
% values to evaluate the "fitness" of each path.

%%  initialize GA Parameters
Init_Pop=5;        % Size of initial population % originally 20
Num_kids=5;        % Number of kids generated in each run % originally
20
S=4;                % Number of designs competing in each tournament
Kpoints=2;          % Number of crossover points
Mut_rate=0.07;      % Mutation Rate
Ctype=1;            % Type of convergence
                    %Ctype=1 #generations Ctype=2 #calls Ctype=3 change
in avgfit Ctype=4 no change in top fitnesses
maxgen=200;         % Max num of generation for Ctype=1
maxcalls=2000;      % Max num of GA calls for Ctype=2
epsilon=0.00001;    % Convergence criteria for Ctype=3
Topcompare=10;      % Convergence criteria for Ctype=4
```

```
%%  initialize counters
convergence=0;
generation=1;

Pop=initialPop(Init_Pop);    % Set population // FUNCTION 1 -
initialPop.m
Fit=FitCheck(Pop);           % Assess fitness of initial population //
FUNCTION 2 - FitCheck.m
calls=Init_Pop;

if(Ctype==3)

    avgfit(generation)=mean(Fit);
end
if(Ctype==4)
    OldFit=Fit(1:Topcompare);
end

%% Enter GA loop
while(convergence==0)
    nkeep=Init_Pop; % set number of kept population from previous GA
run to current GA run's initial population
    for(i=1:Num_kids/2)
        P1=Tournament(Fit,S);   % Tournament selection to find P1 and
P2 from InitPop % FUNCTION 3 - Tournament.m
        P2=Tournament(Fit,S);
        [C1(i,:),C2(i,:)]=Crossover(Pop(P1,:),Pop(P2,:),Kpoints); %
Crossover P1, P2 to find C1, C2  // FUNCTION 4 - Crossover.m
        C1(i,:)=Mutation(C1(i,:),Mut_rate); % Mutate C1, C2 // FUNCTION
5 - Mutation.m
        C2(i,:)=Mutation(C2(i,:),Mut_rate);
    end
    Pop=[Pop;C1]; % Add C1, C2 to Population
    Pop=[Pop;C2];
    Fit=[Fit;FitCheck([C1;C2])]; % Check fitness // FUNCTION 6 -
FitCheck.m
    calls=calls+2*size(C1,1); % update GA calls
    [Pop,Fit]=Trim(Pop,Fit,nkeep); % Trim population back down to
number of designs // FUNCTION 7 - Trim.m
    generation=generation+1; % update generation tally

    %Check for Convergence
    switch Ctype
        case 1
            if(generation>maxgen)
                convergence=1;
            end
        case 2
            if(calls>maxcalls)
                convergence=1;
            end
        case 3
            avgfit(generation)=mean(Fit);
            if(abs(avgfit(generation)-avgfit(generation-
1))/avgfit(generation-1)<epsilon)
                convergence=1;
            end
        case 4
            if(Fit(1:Topcompare)==OldFit)
                convergence=1;
            end
            OldFit=Fit(1:Topcompare);
    end
Fit(1)
end

disp('done')
Fit(1)
```

initialPop.m

```
%% Sets initial population

function [Population] = initialPop(size)
%Makes 'size' number of designs each of which are a random sorting of
16-rows

Rows=16;    %number of rows for each design i.e. 16
Population=zeros(size,Rows);
for(i=1:size)
    Population(i,:)=randperm(Rows);
end


end
```

Tournament.m

```
%% Function to carry out Tournament Selection of population

function [Parent] = Tournament(Fit,S)
%Compares 's' random members of the population and chooses the best
fitness as the next parent

designs=numel(Fit);
selected=randi(designs,1,S);
[Sorted,ind]=sort(Fit(selected),'ascend');
Parent=selected(ind(1));

end
```

Crossover.m

```
%% K-Point Order Based Crossover - checks that no paths are repeated
after crossover ; switches repeating values
function [C1,C2] = Crossover(P1,P2,K)
% Uses k-point ordered crossover of parents 'P1' and 'P2'
% To output children designs 'C1' and 'C2'


C=[P1;P2]';
Genesize=size(P1,2);
k_points=sort(randperm(Genesize,K));
for(k=1:K)
    C(k_points(k):end,:)=C(k_points(k):end,[2,1]);
end

C_band=C(k_points(1):k_points(2)-1,:);
C_out=C([1:k_points(1)-1,k_points(2):end],:);
Oidx=find(ismember(C_out,C_band));
Bidx=find(ismember(C_band,C_out));
Rval=flip(C_band(Bidx));
C_out(Oidx)=Rval;
C_right=C(k_points(1):end,:);
C=[C_out(1:k_points(1)-1,:);C_band;C_out(k_points(1):end,:)];
C1=C(:,1)';
C2=C(:,2)';




end
```

Mutation.m

```
function [Mutated_Child] = Mutation(Child,Rate)
% each index has a percentage rate of 'Rate' (0 to 1) to be changed to
a random value
% then the index that originally contained the new value will be
replaced with the now missing term
% Ensures all values in design remain unique - non- repeating values

GeneSize=numel(Child);
Mutated_Child=Child;
for(i=1:GeneSize)
    if(rand<=Rate)
        swap=randi(GeneSize);
        Mutated_Child(i)=Child(swap);
        Mutated_Child(swap)=Child(i);
        Child=Mutated_Child;
    end
end


end
```

<u>FitCheck,</u>

```
% per Jennifer and Rama: This file will take the following steps:
% (1) take the start positions as the input from the ga
% (2) drop them into the Matlab file for creating the random scan
pattern
%     and writing the corresponding Aba inp file
% (3) Run Abaqus
% (4) Extract from the output the [z-displacements of the nodes] temp
of
%     the nodes
% (5) Some post processing step to average the nodal data over the
whole
%     layer

%% Real Fitness Function

function [Fitness] = FitCheck(Pop)

global rand_array  N_elem_z  N_layers_to_scan  N_elem_y

%evaluates the fitness of all of the designs (rows) of the input "Pop"

N=size(Pop,1);  %Number of designs
N_elem_z = 9; % variable will change depending on geom.
N_layers_to_scan=1; % Number of layers in z which will be added and
scanned
% "So when it comes to the interior block scan, I want to have 2^4
element
% rows in y direction. So that is 16 interior + 2 that will be in
% perimeter."
N_elem_y=2+(2^4);
% to keep track of jobs
% (2) pu the inputs into matlab and obtain the abaqus input file
Job_ctr=0;
Job_ctr_array=[];
Fitness=zeros(N,1);
for(i=1:N) % N is size of population
    % Keeping track of job runs
    Job_ctr=Job_ctr+1;
    Job_ctr_array=[Job_ctr_array; Job_ctr];
    % Set population variable for array of start positions
    D=Pop(i,:);
    % to keep from having to change out inner code that much
    rand_array = Pop(i,:)';
    % run random scan to generate input file
    run run_random_scan\A1_main_scan.m
    % Copy the file into a run directory
    % First create a run directory (this will keep track of runs done
by
    % the GA)
    new_dir_name=strcat('run_',num2str(Job_ctr));
    mkdir('C:\Users\cvbuc\OneDrive\Desktop\Optim Scan\GA
Code\Aba',new_dir_name);
```

```
    % get the files to copy
    files_from_path = 'C:\Users\cvbuc\OneDrive\Desktop\Optim Scan\GA
Code\run_random_scan';
    file1=strcat(files_from_path,'\00_Additive_mfg.inp');
    file2=strcat(files_from_path,'\01_Nodes.inp');
    file3=strcat(files_from_path,'\02_Elements.inp');
    file4=strcat(files_from_path,'\03_Elsets_layers.inp');
    file5=strcat(files_from_path,'\04_Nsets_layers.inp');
    file6=strcat(files_from_path,'\laser_scan_positions.txt');
    file7=strcat(files_from_path,'\heat_flux');

    % and where we are copying them
    files_to_path=strcat('C:\Users\cvbuc\OneDrive\Desktop\Optim Scan\GA
Code\Aba\',new_dir_name);

    % copy the files
    copyfile(file1,files_to_path);
    copyfile(file2,files_to_path);
    copyfile(file3,files_to_path);
    copyfile(file4,files_to_path);
    copyfile(file5,files_to_path);
    copyfile(file6,files_to_path);
    copyfile(file7,strcat(files_to_path,'\heat_flux'));

%% old dir code
% %     mkdir('C:\Users\cvbuc\Dropbox\Buck_Research\GA
Code\Final_for_Fadel\01_standalone_scan')
% %     addpath('.Final_for_Fadel\01_standalone_scan')
%% OLD Abaqus Run Code -- might revisit
    %** here is now how we run abaqus and pull outputs from each
    %  population run **
end
disp1=sprintf('All Input Files (total = %d) written for each member of
the population',N);
disp(disp1)
    for j=Job_ctr_array(1):Job_ctr_array(end)
    % Report time before Abaqus analysis starts
    t1=toc;
    % tic
    tAbaqus=0;
    run_path=strcat('C:\Users\cvbuc\OneDrive\Desktop\Optim Scan\GA
Code\Aba\run_',num2str(j));
    cd(run_path);
    % Run the input file 00_Additive_mfg.inp with Abaqus
    !abaqus analysis job=optim_scan_run input=00_Additive_mfg
    % Pause Matlab execution to give Abaqus enough time to create the
    % optim_scan_run.lck file
    pause(10)
    % If the optim_scan_run.lck file exists then halt Matlab execution
    while exist('optim_scan_run.lck','file')==2
        pause(0.1)
    end
    disp2=sprintf('Simulation %d Finished',j);
    disp(disp2)
    % Report time after Abaqus analysis terminates
      t2=toc;
      tAbaqus=tAbaqus+t2-t1;
%     tTOTAL=toc;
%     tAbaqus;
%     tMatlab=tTOTAL-tAbaqus;
%% from template
% % % % Postprocess Abaqus results file with Abaqus2Matlab
% % % % Assign all lines of the fil file in an one-row string (after
Abaqus
% % % % analysis terminates)
% % %
% % % disp('Reading simulation # %d.fil',j)
% % % Rec = Fil2str(['optim_scan_run.fil']);
% % %
% % %
% % % % Obtain the desired output data
% % % disp('Obtaining desired output data')
% % % out = RecXXX(Rec); % Put here the Rec function selected
%% From truss optimization example
    % Obtain the nodal displacements (CHECK DIRECTION)
    disp3=sprintf('reading simulation # %d .fil and obtaining desired
output data',j);
    disp(disp3)
    oldfolder = cd('C:\Users\cvbuc\OneDrive\Desktop\Optim Scan\GA
Code');
    newfolder = run_path;
    cd(newfolder);
    out2 = readFil('optim_scan_run.fil',101);
    NodalDisplacements=out2{1,1}(:,2:3);
    % Delete the files of last Abaqus run to avoid rewriting them
    delete('optim_scan_run.fil');
    delete('optim_scan_run.prt');
    delete('optim_scan_run.com');
    delete('optim_scan_run.sim');
    % Calculate the maximum nodal displacements
    maxNodDisplX1=max(abs(NodalDisplacements(:,1)));
    maxNodDisplY1=max(abs(NodalDisplacements(:,2)));
    maxNodDisplZ1=max(abs(NodalDisplacements(:,3)));

    Fitness(i) = maxNodDisplZ1;
    fprintf('Fitness of simulation #1 = %d', Fitness(i))
    cd(oldfolder);
    end
end

]
```

Trim.m

```
function [Population,Fitness] = Trim(Population,Fitness,nkeep)
% keeps the best 'nkeep' number of designs based on top fitness values
% Trims population back to desired number of population number
[Fitness,ind]=sort(Fitness,'ascend');
Population=Population(ind,:);
Fitness=Fitness(1:nkeep);
Population=Population(1:nkeep,:);

end
```

APPENDIX C. BATCH SCRIPTS FOR PALMETTO CLUSTER GA RUN

Appendix C includes the batch scripts written in Linux to help facilitate the i/o between MATLAB and Abaqus, as well parallelize the Abaqus runs, Updated or newly made MATLAB scripts from Grigori Yourganov to execute the parallelization of Abaqus runs are also included. The overall GA code remains the same in structure and goal, and the simlations for each member of each population is run the same as the code in Appendix A, which the only change being the scanning pattern in each simulation coming from the GA population members. Descriptions of each batch script and what they execute are included next to the file name if the file name is not descriptive enough.

main_batch_eps.sh: This file the main batch script file from which the GA and all functions within it are executed on the Cluster. This batch script includes convergences for either reaching max number of generations or if the average fitness is the same between two consecutive generations.

```
#!/bin/bash

num_kids=4
tourn_select=4
kpoints=2
mut_rate=0.01
init_pop=10
nkeep=$init_pop
maxgen=30
epsilon=0.01
fast_queue=0

code_path=$PWD
results_path="$PWD/fivelayertest/"
echo "Code path: $code_path"
echo "Results path: $results_path"

if [ ! -d $results_path ]; then
 mkdir $results_path
fi
cd $results_path
```

```
rm -rf *
cd $code_path
if [ -f lock_file.txt ]; then
 rm lock_file.txt
fi

echo "Initializing a population of ${init_pop}"
qsub -v code_path=$code_path,results_path=$results_path,init_pop=$init_pop
qsub_batch1.sh

echo "Waiting for the gen_sequences file to appear"
until [ -f $results_path/gen_sequences.txt ]; do
 sleep 1
done
while [ $(wc -l $results_path/gen_sequences.txt | awk '{print $1}') -lt $init_pop ]; do
 sleep 1
done

echo "Running initial run"
generation_number=1
keep_working=1
cd $results_path/Aba
run_folders=$(ls -d run_*)
for one_run in $run_folders; do
 cd $results_path/Aba
 cd ${one_run}
 run_path=$PWD
 cd $code_path
 if [ $fast_queue -eq 1 ]; then
  qsub -v code_path=$code_path,results_path=$results_path,run_path=$run_path
qsub_batch2.sh
 else
  qsub -v code_path=$code_path,results_path=$results_path,run_path=$run_path
qsub_batch2_slowq.sh
 fi
done

cd $results_path
echo "Waiting for the gen_fitnesses file to appear"
until [ -f gen_fitnesses.txt ]; do
 sleep 1
done
while [ $(wc -l gen_fitnesses.txt | awk '{print $1}') -lt $init_pop ]; do
 sleep 1
done
```

```
sort gen_sequences.txt > cur_sequences.txt
sort gen_fitnesses.txt > cur_fitnesses.txt
prev_average=$($code_path/calc_average.sh $results_path/cur_fitnesses.txt)

echo "Generation $generation_number"
echo "Sequences:"
cat gen_sequences.txt
echo "Fitnesses:"
cat gen_fitnesses.txt
echo "Average fitness = $prev_average"

#while [ $generation_number -lt $maxgen ]; do
while [ $keep_working -eq 1 ]; do
  cd $results_path

  generation_number=$(($generation_number+1))
  echo "processing generation $generation_number"
  rm gen_sequences.txt
  rm gen_fitnesses.txt
  rm -rf Aba/run*
  qsub -v
code_path=$code_path,results_path=$results_path,num_kids=$num_kids,tourn_select=$t
ourn_select,kpoints=$kpoints,mut_rate=$mut_rate $code_path/qsub_batch3.sh
  echo "Waiting for the gen_sequences file to appear"
  until [ -f gen_sequences.txt ]; do
   sleep 1
  done
  while [ $(wc -l gen_sequences.txt | awk '{print $1}') -lt $num_kids ]; do
   sleep 1
  done

  cd $results_path/Aba
  run_folders=$(ls -d run_*)
  for one_run in $run_folders; do
   cd $results_path/Aba/$one_run
   run_path=$PWD
   cd $code_path
   if [ $fast_queue -eq 1 ]; then
    qsub -v code_path=$code_path,results_path=$results_path,run_path=$run_path
qsub_batch2.sh
   else
    qsub -v code_path=$code_path,results_path=$results_path,run_path=$run_path
qsub_batch2_slowq.sh
   fi
  done
  echo "Waiting for the gen_fitnesses file to appear"
  until [ -f $results_path/gen_fitnesses.txt ]; do
   sleep 1
  done
  while [ $(wc -l $results_path/gen_fitnesses.txt | awk '{print $1}') -lt $num_kids ]; do
   sleep 1
  done

  echo "*******"
  echo "New kids created in Generation $generation_number"
  echo "Sequences:"
  cat $results_path/gen_sequences.txt
  echo "Fitnesses:"
  cat $results_path/gen_fitnesses.txt

  echo "Trimming in process..."
  echo "Trimming in process..." > $results_path/trim_lock.txt
  qsub -v code_path=$code_path,results_path=$results_path,nkeep=$nkeep
$code_path/qsub_batch4.sh
  while [ -f $results_path/trim_lock.txt ]; do
   sleep 1
  done
  echo "After combining and trimming:"
  echo "Sequences:"
  cat $results_path/cur_sequences.txt
  echo "Fitnesses:"
  cat $results_path/cur_fitnesses.txt

  curr_average=$($code_path/calc_average.sh $results_path/cur_fitnesses.txt)
  echo "Average fitness = $curr_average"
  diff=$(echo "sqrt(($curr_average - $prev_average)^2)" | bc)
  echo "Diff = $diff"
  keep_working=$(echo "$diff > $epsilon && $generation_number < $maxgen" | bc)
  prev_average=$curr_average
done

cd $results_path
echo "Finished!"
echo "Final sequences:"
cat cur_sequences.txt
echo "Final Fitnesses:"
cat cur_fitnesses.txt
cp cur_fitnesses.txt final_fitnesses.txt
cp cur_sequences.txt final_sequences.txt
```

qsub_batch1.sh: This batch script initializes the code and runs the 'getting_started.m matlab script. This initialization script creates the random sequences for the initial population.

```
#!/bin/bash
#
#PBS -N initialize
#PBS -l select=1:ncpus=1:mem=10gb
#PBS -l walltime=1:00:00
#PBS -o initialization.txt
#PBS -j oe

cd $code_path

matlab_input="getting_started('${code_path}','${results_path}',${init_pop})"
echo $matlab_input
module load matlab/2018b
module load abaqus/6.14
taskset -c 0-$(($OMP_NUM_THREADS-1)) matlab -nodisplay -nosplash -r
${matlab_input} -logfile $results_path/initialization.out
```

qsub_batch2_slowq.sh: This batch script creates the Abaqus input files for each member of the population, submits them in parallel, and then reads the output to calculate the fitness using the read_abaqus_output.m MATLAB script. The "slowq" part of the title refers to the use of the older nodes on the Cluster; while slowing in queue, these nodes have a higher walltime and memory which is needed for these runs.

```
#!/bin/bash
#
#PBS -N abaqus_matlab
#PBS -l select=1:ncpus=4:mem=20gb:interconnect=1g
#PBS -l walltime=150:00:00
#PBS -j oe
#PBS -o abaqus_matab.txt

#matlab_input="fit_check_onepop('${run_path}')"
run_number=$(echo $run_path | cut -d \_ -f 2)
output_log="$results_path/fitcheck_$run_number.out"
module load matlab/2018b
module load abaqus/6.14

cd $code_path
matlab_input="prepare_for_abaqus('${code_path}','${results_path}','${run_path}')"
echo $matlab_input
taskset -c 0-$(($OMP_NUM_THREADS-1)) matlab -nodisplay -nosplash -r
${matlab_input} -logfile ${output_log}
```

```
cd $run_path
/software/abaqus/6.14-1/code/bin/abaqus analysis job=optim_scan_run
input=00_Additive_mfg cpus=$OMP_NUM_THREADS background
sleep 10
while [ -f optim_scan_run.lck ]; do
 sleep 1
done

cd $code_path
matlab_input="read_abaqus_output('${code_path}','${results_path}','${run_path}')"
echo $matlab_input
taskset -c 0-$(($OMP_NUM_THREADS-1)) matlab -nodisplay -nosplash -r
${matlab_input} -logfile ${output_log}
```

<u>**qsub_batch3.sh:** This batch script calls spawn_generation.m, which completes the tournament selection, croasover, and mutation function in mtlab. The output is the children created from the selected parents in the overall population.</u>

```
#!/bin/bash
#
#PBS -N spawn_generation
#PBS -l select=1:ncpus=2:mem=10gb
#PBS -l walltime=5:00:00
#PBS -o spawn_output.txt
#PBS -j oe

cd $code_path

matlab_input="spawn_generation('${code_path}','${results_path}',${num_kids},${tourn
_select},${kpoints},${mut_rate})"
echo $matlab_input
module load matlab/2018b
module load abaqus/6.14
taskset -c 0-$(($OMP_NUM_THREADS-1)) matlab -nodisplay -nosplash -r
${matlab_input} -logfile $results_path/spawn_generation.out
```

<u>**qsub_batch4.sh:** This batch script analyzes the children and then trims the population back to the original size before the children were created, keeping only the members with the highest fitness (in this case, the members with the lowest temperature output from MATLAB).</u>

```
#!/bin/bash
#
#PBS -N process_generation
#PBS -l select=1:ncpus=2:mem=10gb
#PBS -l walltime=5:00:00
```

```
#PBS -o process_output.txt
#PBS -j oe

#echo "Trimming in progress!" > $results_path/trim_lock.txt

cd $code_path

matlab_input="process_generation('${code_path}','${results_path}',${nkeep})"
echo $matlab_input
module load matlab/2018b
module load abaqus/6.14
taskset -c 0-$(($OMP_NUM_THREADS-1)) matlab -nodisplay -nosplash -r
${matlab_input} -logfile $results_path/process_generation.out
```

getting_started.m: This MATLAB function sets up the initial population and creates the scanning patterns of the initial population.

```
function getting_started (code_path, results_path, Init_Pop)
% code_path = '/home/gyourga/source/cvbuck2';
% results_path = [code_path '/test'];
rng ('shuffle');
addpath (genpath (code_path));
if ~isfolder (results_path)
   mkdir (results_path);
end

Pop=initialPop(Init_Pop);   % Set population // FUNCTION 1 - initialPop.m
cd (results_path);
fp2 = fopen ([results_path '/all_sequences.txt'], 'w');
fprintf (fp2, '******\n');
mkdir ('Aba');
cd ('Aba');
for i = 1:Init_Pop
   run_name = ['run_' num2str(i)];
   if isfolder (run_name)
      rmdir (run_name);
   end
   mkdir (run_name);
   cd (run_name);
   fp = fopen ('sequence.txt', 'w');
   fprintf (fp, '%d ', Pop (i, :));
   fclose (fp);
   cd ..
   fp = fopen ([results_path '/gen_sequences.txt'], 'a');
   str = [num2str(i) ' ' sprintf('%d ', Pop (i, :))];
   fprintf (fp, '%s\n', str);
```

```
    fclose (fp);
    fprintf (fp2, '%s\n', str);
end
fclose (fp2);
```

prepare_for_abaqus.m: This MATLAB function creates the run folders for each Abaqus run of the members of the opulation.

```
function [Fitness] = prepare_for_abaqus(code_path, results_path, run_path)

global rand_array  N_elem_z  N_layers_to_scan  N_elem_y num_nodes_xy_plane
%#ok<NUSED>
%code_path = '/home/gyourga/source/cvbuck2';
%results_path = [code_path '/test'];
addpath (genpath (code_path));
rng ('shuffle');

N_elem_z = 9; % variable will change depending on geom.
N_layers_to_scan=5; % Number of layers in z which will be added and scanned
% "So when it comes to the interior block scan, I want to have 2^4 element
% rows in y direction. So that is 16 interior + 2 that will be in
% perimeter."
N_elem_y=2+(2^4);


cd (run_path);
fp = fopen ('sequence.txt');
sequence = fscanf (fp, '%d ');
fclose (fp);
rand_array = sequence;

cd (code_path);

while exist('lock_file.txt','file')==2
    pause(0.1)
end
disp (['Copying files for ' run_path]);

fp = fopen ('lock_file.txt', 'w');
fprintf (fp, 'Lemme finish!\n');
fclose (fp);

run run_random_scan/A1_main_scan.m
files_from_path = [code_path '/run_random_scan'];
file1=strcat(files_from_path,'/00_Additive_mfg.inp');
```

```
file2=strcat(files_from_path,'/01_Nodes.inp');
file3=strcat(files_from_path,'/02_Elements.inp');
file4=strcat(files_from_path,'/03_Elsets_layers.inp');
file5=strcat(files_from_path,'/04_Nsets_layers.inp');
file6=strcat(files_from_path,'/laser_scan_positions.txt');
file7=strcat(files_from_path,'/heat_flux');

% and where we are copying them
files_to_path=run_path;

% copy the files
copyfile(file1,files_to_path);
copyfile(file2,files_to_path);
copyfile(file3,files_to_path);
copyfile(file4,files_to_path);
copyfile(file5,files_to_path);
copyfile(file6,files_to_path);
copyfile(file7,strcat(files_to_path,'/heat_flux'));

% added by GY: copy the environment file that tell Abaqus not to prompt the user when
overwriting a file
copyfile ([code_path '/abaqus_v6.env'], files_to_path);

delete ('lock_file.txt');
```

process_generation.m: This function finds the fitnesses and sequences of the parents and children that have been calculated, and trims the generation back, keeping only the fittest individuals.

```
function process_generation (code_path, results_path, nkeep)
addpath (genpath (code_path));
rng ('shuffle');

cd (results_path);
%fp = fopen ('trim_lock.txt', 'w');
%fprintf (fp, 'Trimming in progress\n');
%fclose (fp);

fp_seq = fopen ('cur_sequences.txt');
while ~feof (fp_seq)
   line = fgetl (fp_seq);
   temp = sscanf (line, '%d');
   Pop (temp (1), :) = temp (2:length(temp));
end
fclose (fp_seq);
fp_fit = fopen ('cur_fitnesses.txt');
```

```
while ~feof (fp_fit)
   line = fgetl (fp_fit);
   temp = sscanf (line, '%f');
   Fit (temp (1)) = temp (2);
end
fclose (fp_fit);
Fit = Fit';

fp_seq = fopen ('gen_sequences.txt');
while ~feof (fp_seq)
   line = fgetl (fp_seq);
   temp = sscanf (line, '%d');
   new_sequence (temp (1), :) = temp (2:length(temp));
end
fclose (fp_seq);
fp_fit = fopen ('gen_fitnesses.txt');
while ~feof (fp_fit)
   line = fgetl (fp_fit);
   temp = sscanf (line, '%f');
   new_fitness (temp (1)) = temp (2);
end
fclose (fp_fit);
new_fitness = new_fitness';

Pop = [Pop; new_sequence];
Fit = [Fit; new_fitness];
[Pop,Fit]=Trim(Pop,Fit,nkeep); % Trim population back down to number of designs //
FUNCTION 7 - Trim.m

fp = fopen ('cur_fitnesses.txt', 'w');
for i = 1:size(Fit, 1)
   fprintf (fp, '%d %.8f\n', i, Fit(i));
end
fclose (fp);
fp = fopen ('cur_sequences.txt', 'w');
for i = 1:size(Pop, 1)
   str = [num2str(i) ' ' sprintf('%d ', Pop (i, :))];
   fprintf (fp, '%s\n', str);
end
fclose (fp);

delete ('trim_lock.txt');
```

read_abaqus_output.m: This function reads the output temperature data from MATLAB for all members of the population.

```
function [Fitness] = read_abaqus_output(code_path, results_path, run_path)

%code_path = '/home/gyourga/source/cvbuck2';
%results_path = [code_path '/test'];
addpath (genpath (code_path));
rng ('shuffle');

cd (run_path);

fprintf ('Reading the output file...\n');
out2 = Rec201 (fil2Str('optim_scan_run.fil'));
NodalTemps = out2 (:, 2); % changed by GY: Rec201 retuns a double array, not a cell
array
%THIS IS HARDCODED FOR NOW, CHANGE LATER
num_nodes=902;
% num_time=1980; %for .1m/s
num_time=3960; %for .05m/s
temps=reshape(NodalTemps,num_nodes,num_time);
[maxenv,~]=envelope(temps);
maxenv2=max(maxenv);
% Calculate the avgmaxNT11
avgmaxNT11=mean(maxenv2(1,:));

% BELOW IS FOR DISPLACEMENT
% out2 = readFil('optim_scan_run.fil',101);
% NodalDisplacements=out2{1,1}(:,2:3);
% Delete the files of last Abaqus run to avoid rewriting them
delete('optim_scan_run.fil');
delete('optim_scan_run.prt');
delete('optim_scan_run.com');
delete('optim_scan_run.sim');

Fitness=avgmaxNT11;

ii = find (run_path == '_');
jj = ii (length (ii));
seq_number = str2num (run_path (jj+1:length(run_path)));
fprintf('Sequence %d: fitness of simulation = %f\n', seq_number, Fitness);
cd(results_path);
fp = fopen ('gen_fitnesses.txt', 'a');
fprintf (fp, '%d %.8f\n', seq_number, Fitness);
fclose (fp);
```

Rec201.m: MATLAB function created by Abaqus2Matlab Toolboc for reading the temperature output from an Abaqus simulation [50].

```
function out = Rec201(Rec)
%
% ABAQUS temperature output to MATLAB
%
% Syntax
%     #Rec# = Fil2str('*.fil');
%     #out# = Rec201(#Rec#)
%
% Description
%     Read temperature output from the results (*.fil) file generated from
%     the ABAQUS finite element software. The asterisk (*) is replaced by
%     the name of the results file. The record key for temperature output
%     is 201. See section < < Results file output format > > in ABAQUS Analysis
%     User's manual for more details.
%     The following options with parameters have to be specified in the
%     ABAQUS input file for the results (*.fil) file to be created and to
%     contain temperature results:
%         ...
%         *FILE FORMAT, ASCII
%         *NODE FILE
%         NT
%         ...
%     NOTE: The results file (*.fil) must be placed in the same directory
%     with the MATLAB source files in order to be processed.
%
% Input parameters
%     #Rec# (string) is an one-row string containing the ASCII code of the
%         ABAQUS results (*.fil) file. It is generated by the function
%         Fil2str.m.
%
% Output parameters
%     #out# ([#n# x #m#]) is a double array containing the attributes of
%         the record key 201 as follows:
%         Column  1  ñ  Node number.
%         Column  2  ñ  Temperature.
%         Column  3  ñ  Etc (for heat shells)
%         where #n# is the number of nodes multiplied by the number of
%         increments and #m#-1 is the number of temperatures per node. If
%         the results file does not contain the desired output, #out# will
%         be an empty array
%
%
_____________________________________________________________________________
_
% Abaqus2Matlab - www.abaqus2matlab.com
% Copyright (c) 2016 by George Papazafeiropoulos
```

```
%
% If using this toolbox for research or industrial purposes, please cite:
% G. Papazafeiropoulos, M. Muniz-Calvente, E. Martinez-Paneda.
% Abaqus2Matlab: a suitable tool for finite element post-processing (submitted)
%
%

ind = strfind(Rec,'I 3201I'); % record key for node temperature output (201)
if isempty(ind)
    out=[];
    return;
end
nextpos=numel('I 3201')+1;
% Initialize record length matrix
NW=zeros(numel(ind),1);
for i=1:numel(ind)
    % find the record length (NW)
    Rec2=Rec(ind(i)-7:ind(i));
    indNW=strfind(Rec2,'*'); % record starting position
    % ensure that the record exists and that the record type key is at
    % location 2
    if isempty(indNW) || indNW>3
        ind(i)=NaN;
        continue;
    end
    % number of digits of record length
    ind1=indNW+2; % 1st digit of 2-digit integer of 1st data item
    ind2=indNW+2+1; % 2nd digit of 2-digit integer of 1st data item
    a1=str2num(Rec2(ind1:ind2));
    % Record length (NW)
    ind1=ind1+2; % +2 digits
    ind2=ind2+a1; % +2-digit integer
    NW(i)=str2num(Rec2(ind1:ind2));
end
% remove ind and NW values which do not correspond to output
NW(isnan(ind))=[];
ind(isnan(ind))=[];
% Initialize
NodeNum=zeros(numel(ind),1);
NodeOut=zeros(numel(ind),max(NW)-3);
for i=1:numel(ind)
    % number of digits of node number
    ind1=ind(i)+nextpos; % 1st digit of 2-digit integer of 3rd data item
    ind2=ind(i)+nextpos+1; % 2nd digit of 2-digit integer of 3rd data item
    a1=str2num(Rec(ind1:ind2));
    % Node number
    ind1=ind1+2; % +2 digits
    ind2=ind2+a1; % +2-digit integer
    NodeNum(i)=str2num(Rec(ind1:ind2));
    % node temperatures
    for j=1:NW(i)-3
        % temperature
        ind1=ind2+1+1; % +1 character+1
        ind2=ind2+1+22; % +1 character +22 floating point digits
        NodeOut(i,j)=str2num(Rec(ind1:ind2));
    end
end
% Assemply of matrices for output
out=[NodeNum NodeOut];

end
```

spawn_generation.m: This MATLAB function creates the new children for each generation.

```
function spawn_generation (code_path, results_path, Num_kids, S, Kpoints, Mut_rate)
addpath (genpath (code_path));
rng ('shuffle');

cd (results_path);
fp_seq = fopen ('cur_sequences.txt');
while ~feof (fp_seq)
    line = fgetl (fp_seq);
    temp = sscanf (line, '%d');
    Pop (temp (1), :) = temp (2:length(temp));
end
fclose (fp_seq);
fp_fit = fopen ('cur_fitnesses.txt');
while ~feof (fp_fit)
    line = fgetl (fp_fit);
    temp = sscanf (line, '%f');
    Fit (temp (1)) = temp (2);
end
fclose (fp_fit);
Fit = Fit';

fp_log = fopen ([results_path '/fitness_log.txt'], 'a');
for i = 1:length (Fit)
    sequence_str = sprintf ('%d ', Pop (i, :));
    fprintf (fp_log, '%d %s %.4f\n', i, sequence_str, Fit(i));
end
fclose (fp_log);
```

```
for(i=1:Num_kids/2)
   P1=Tournament(Fit,S);   % Tournament selection to find P1 and P2 from InitPop %
FUNCTION 3 - Tournament.m
   P2=Tournament(Fit,S);
   [C1(i,:),C2(i,:)]=Crossover(Pop(P1,:),Pop(P2,:),Kpoints); % Crossover P1, P2 to find
C1, C2  // FUNCTION 4 - Crossover.m
   C1(i,:)=Mutation(C1(i,:),Mut_rate); % Mutate C1, C2 // FUNCTION 5 - Mutation.m
   C2(i,:)=Mutation(C2(i,:),Mut_rate);
end
new_generation = [C1;C2];

cd (results_path);
fp2 = fopen ('all_sequences.txt', 'a');
fprintf (fp2, '******\n');
cd ('Aba');
for i = 1:Num_kids
   run_name = ['run_' num2str(i)];
   if isfolder (run_name)
      rmdir (run_name, 's');
   end
   mkdir (run_name);
   cd (run_name);
   fp = fopen ('sequence.txt', 'w');
   fprintf (fp, '%d ', new_generation (i, :));
   fclose (fp);
   cd ..
   fp = fopen ([results_path '/gen_sequences.txt'], 'a');
   str = [num2str(i) ' ' sprintf('%d ', new_generation (i, :))];
   fprintf (fp, '%s\n', str);
   fclose (fp);
   fprintf (fp2, '%s\n', str);
end
fclose (fp2);
```

APPENDIX D. FULL RESULTS FROM GENETIC ALGORITHM RUNS

Appendix D consists of the full results from the Genetic Algorithm runs. Each table has the scanning patterns and corresponding fitness for each generation.

SINGLE LAYER FULL OPTIMIZATION TEST ONE RESULTS:

Scanning Pattern	Fitness	Generation Number
10 15 1 5 7 11 8 3 6 14 16 4 12 2 9 13	523.6364	1
16 8 13 7 5 10 2 15 1 11 14 12 3 9 4 6	538.3928	1
7 4 10 13 8 15 9 1 5 14 6 3 12 2 16 11	525.8633	1
2 12 15 13 5 11 4 7 8 10 16 1 3 6 9 14	528.5111	1
8 7 10 11 6 12 15 9 3 5 4 2 13 1 14 16	543.4936	1
7 1 6 8 5 10 3 11 2 13 14 9 12 4 16 15	523.4062	1
12 13 9 16 8 2 5 6 7 15 10 4 1 14 3 11	522.2229	1
1 3 10 14 15 16 2 11 12 13 8 7 9 5 6 4	534.9879	1
2 7 10 6 14 11 16 13 3 12 9 15 1 4 5 8	538.7946	1
0 3 14 16 12 9 8 1 2 6 11 13 7 15 4 10 5	530.0398	1
12 13 9 16 8 2 5 6 7 15 10 4 1 14 3 11	522.2229	2
12 13 9 16 8 2 5 6 7 15 10 4 1 14 3 11	522.2229	2
12 13 9 16 8 2 5 6 7 15 10 4 1 14 3 11	522.2229	2
10 15 1 5 8 2 11 3 6 14 16 4 12 7 9 13	522.7798	2
7 1 6 8 5 10 3 11 2 13 14 9 12 4 16 15	523.4062	2
10 15 1 5 7 11 8 3 6 14 16 4 12 2 9 13	523.6364	2
12 13 9 16 7 11 5 6 2 15 10 4 1 14 3 8	523.8514	2
7 4 10 13 8 15 9 1 5 14 6 3 12 2 16 11	525.8633	2
2 12 15 13 5 11 4 7 8 10 16 1 3 6 9 14	528.5111	2
0 3 14 16 12 9 8 1 2 6 11 13 7 15 4 10 5	530.0398	2
12 13 9 16 8 2 5 6 7 15 10 4 1 14 3 11	520.5771	3
9 1 13 8 5 10 3 11 6 14 16 4 12 7 2 15	520.5771	3
12 13 9 16 8 2 5 6 7 15 10 4 1 14 3 11	522.2229	3
12 13 9 16 8 2 5 6 7 15 10 4 1 14 3 11	522.2229	3
12 13 9 16 8 2 5 6 7 15 10 4 1 14 3 11	522.2229	3
12 13 9 16 8 2 5 6 7 15 10 4 1 14 3 11	522.2229	3
10 15 1 5 8 2 11 3 6 14 16 4 12 7 9 13	522.7798	3
7 1 6 8 5 10 3 11 2 13 14 9 12 4 16 15	523.4062	3
10 15 1 5 7 11 8 3 6 14 16 4 12 2 9 13	523.6364	3
0 12 13 9 16 7 11 5 6 2 15 10 4 1 14 3 8	523.8514	3
9 13 1 8 5 10 3 11 6 14 16 4 12 7 2 15	520.0048	4

12 13 9 16 8 2 5 6 7 15 10 4 1 14 3 11	520.5771	4
9 1 13 8 5 10 3 11 6 14 16 4 12 7 2 15	520.5771	4
12 1 9 16 8 2 5 6 7 15 10 4 13 14 3 11	520.9359	4
12 13 9 16 8 2 5 6 7 15 10 4 1 14 3 11	522.2229	4
12 13 9 16 8 2 5 6 7 15 10 4 1 14 3 11	522.2229	4
12 13 9 16 8 2 5 6 7 15 10 4 1 14 3 11	522.2229	4
12 13 9 16 8 2 5 6 7 15 10 4 1 14 3 11	522.2229	4
12 13 9 16 8 2 5 6 7 15 10 4 1 14 3 11	522.2229	4
0 10 15 1 5 8 2 11 3 6 14 16 4 12 7 9 13	522.7798	4
9 13 1 8 5 10 3 11 6 14 16 4 12 7 2 15	520.0048	5
9 13 1 8 5 10 3 11 6 14 16 4 12 7 2 15	520.0048	5
9 1 13 8 5 10 3 11 6 14 16 4 12 7 2 15	520.1569	5
9 1 13 8 5 10 3 11 6 14 16 4 12 7 2 15	520.1569	5
12 13 9 16 8 2 5 6 7 15 10 4 1 14 3 11	520.5771	5
9 1 13 8 5 10 3 11 6 14 16 4 12 7 2 15	520.5771	5
12 1 9 16 8 2 5 6 7 15 10 4 13 14 3 11	520.9359	5
12 13 9 16 8 2 5 6 7 15 10 4 1 14 3 11	522.2229	5
12 13 9 16 8 2 5 6 7 15 10 4 1 14 3 11	522.2229	5
0 12 13 9 16 8 2 5 6 7 15 10 4 1 14 3 11	522.2229	5
9 13 1 8 5 10 3 11 6 14 16 4 12 7 2 15	520.0048	6
9 13 1 8 5 10 3 11 6 14 16 4 12 7 2 15	520.0048	6
9 13 1 8 5 10 3 11 6 14 16 4 12 7 2 15	520.0048	6
9 13 1 8 5 10 3 11 6 14 16 4 12 7 2 15	520.0048	6
9 1 13 8 5 10 3 11 6 14 16 4 12 7 2 15	520.1569	6
9 1 13 8 5 10 3 11 6 14 16 4 12 7 2 15	520.1569	6
9 1 13 8 5 10 3 11 6 14 16 4 12 7 2 15	520.1569	6
9 5 1 8 13 10 3 11 6 14 16 4 12 7 2 15	520.3534	6
12 13 9 16 8 2 5 6 7 15 10 4 1 14 3 11	520.5771	6
0 9 1 13 8 5 10 3 11 6 14 16 4 12 7 2 15	520.5771	6
9 13 1 8 5 10 3 11 6 14 16 4 12 7 2 15	520.0048	7
9 13 1 8 5 10 3 11 6 14 16 4 12 7 2 15	520.0048	7
9 13 1 8 5 10 3 11 6 14 16 4 12 7 2 15	520.0048	7
9 13 1 8 5 10 3 11 6 14 16 4 12 7 2 15	520.0048	7
9 13 1 8 5 10 3 11 6 14 16 4 12 7 2 15	520.0048	7
9 13 1 8 5 10 3 11 6 14 16 4 12 7 2 15	520.0048	7
9 13 1 8 5 10 3 11 6 14 16 4 12 7 2 15	520.0048	7
9 13 1 8 5 10 3 11 6 14 16 4 12 7 2 15	520.0048	7
9 1 13 8 5 10 3 11 6 14 16 4 12 7 2 15	520.1569	7
0 9 1 13 8 5 10 3 11 6 14 16 4 12 7 2 15	520.1569	7

SINGLE LAYER FULL OPTIMIZATION TEST TWO RESULTS:

Scanning Pattern	Fitness	Generation Number
15 14 12 5 16 6 4 13 3 1 8 2 11 7 9 10	549.6049	1
16 14 10 4 9 12 5 3 7 6 13 8 11 15 1 2	536.1955	1
11 15 12 10 2 7 6 1 16 8 3 13 4 9 14 5	524.7907	1
15 8 12 13 14 9 7 5 4 2 16 3 10 1 11 6	535.0237	1
9 15 6 16 11 5 1 13 4 3 10 8 14 2 12 7	544.5099	1
15 12 7 16 6 8 14 5 9 11 4 10 2 13 3 1	540.1388	1
6 10 14 5 7 15 16 2 4 12 8 13 1 9 11 3	537.7729	1
1 14 15 13 8 4 10 12 6 9 16 5 3 2 11 7	533.6638	1
13 4 10 3 5 15 6 14 9 1 11 16 2 7 12 8	539.3556	1
0 2 14 13 5 4 10 15 3 1 9 6 7 11 8 12 16	539.5525	1
11 15 12 10 2 7 6 1 16 8 3 13 4 9 14 5	524.7907	2
1 14 15 13 8 4 10 12 6 9 16 3 5 2 11 7	533.6483	2
1 14 15 13 8 4 10 12 6 9 16 5 3 2 11 7	533.6638	2
15 8 12 13 14 9 7 3 4 2 16 5 10 1 11 6	534.7108	2
15 8 12 13 14 9 7 5 4 2 16 3 10 1 11 6	535.0237	2
16 14 10 4 9 12 5 3 7 6 13 8 11 15 1 2	536.1955	2
16 14 10 4 9 12 5 3 7 6 13 8 11 15 1 2	536.1955	2
16 14 10 4 9 12 5 3 7 6 13 8 11 15 1 2	536.1955	2
6 10 14 5 7 15 16 2 4 12 8 13 1 9 11 3	537.7729	2
0 13 4 10 3 5 15 6 14 9 1 11 16 2 7 12 8	539.3556	2
11 15 12 10 2 7 6 1 16 8 3 13 4 9 14 5	524.7907	3
11 15 12 10 4 7 6 1 13 9 16 5 3 2 14 8	525.2977	3
1 14 15 13 8 4 10 12 6 9 16 3 5 2 11 7	533.6483	3
1 14 15 13 8 4 10 12 6 9 16 3 5 2 11 7	533.6483	3
1 14 15 13 8 4 10 12 6 9 16 5 3 2 11 7	533.6638	3
1 14 15 13 8 4 10 12 6 9 16 5 3 2 11 7	533.6638	3
15 8 12 13 14 9 7 3 4 2 16 5 10 1 11 6	534.7108	3
15 8 12 13 14 9 7 5 4 2 16 3 10 1 11 6	535.0237	3
16 14 10 4 9 12 5 3 7 6 13 8 11 15 1 2	536.1955	3
0 16 14 10 4 9 12 5 3 7 6 13 8 11 15 1 2	536.1955	3
11 15 12 10 2 7 6 1 16 8 3 13 4 9 14 5	524.7907	4
11 15 12 10 4 7 6 1 13 9 16 5 3 2 14 8	525.2977	4
11 15 12 10 4 7 6 1 13 9 16 5 3 2 14 8	525.2977	4
11 15 12 10 2 7 6 1 16 8 3 4 13 9 14 5	525.464	4
11 14 15 13 2 7 6 1 16 8 3 10 4 9 12 5	527.8739	4
1 14 15 13 8 4 10 12 6 9 16 3 5 2 11 7	533.6483	4

1 14 15 13 8 4 10 12 6 9 16 3 5 2 11 7	533.6483	4
1 14 15 13 8 4 10 12 6 9 16 5 3 2 11 7	533.6638	4
1 14 15 13 8 4 10 12 6 9 16 5 3 2 11 7	533.6638	4
0 15 8 12 13 14 9 7 3 4 2 16 5 10 1 11 6	534.7108	4
8 14 12 10 4 7 6 1 13 9 16 3 5 2 11 15	524.4788	5
11 15 12 10 2 7 6 1 16 8 3 13 4 9 14 5	524.7907	5
11 15 12 10 4 7 6 1 13 9 16 5 3 2 14 8	525.2977	5
11 15 12 10 4 7 6 1 13 9 16 5 3 2 14 8	525.2977	5
11 15 12 10 2 7 6 1 16 8 3 4 13 9 14 5	525.464	5
11 14 15 10 2 7 6 1 16 8 3 13 4 9 12 5	525.9824	5
11 15 12 13 2 7 6 1 16 8 3 10 4 9 14 5	526.7671	5
11 14 15 13 2 7 6 1 16 8 3 10 4 9 12 5	527.8739	5
11 1 15 13 8 4 10 12 6 9 16 5 3 2 14 7	532.5524	5
0 1 14 15 13 8 4 10 12 6 9 16 3 5 2 11 7	533.6483	5
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	6
8 14 12 10 4 7 6 1 13 9 16 3 5 2 11 15	524.4788	6
8 14 12 10 4 7 6 1 13 9 16 3 5 2 11 15	524.4788	6
8 14 12 10 4 7 6 1 13 9 16 3 5 2 11 15	524.4788	6
11 15 12 10 2 7 6 1 16 8 3 13 4 9 14 5	524.7907	6
11 15 12 10 4 7 6 1 13 9 16 5 3 2 14 8	525.2977	6
11 15 12 10 4 7 6 1 13 9 16 5 3 2 14 8	525.2977	6
11 15 12 10 2 7 6 1 16 8 3 4 13 9 14 5	525.464	6
11 14 15 10 2 7 6 1 16 8 3 13 4 9 12 5	525.9824	6
0 11 15 12 13 2 7 6 1 16 8 3 10 4 9 14 5	526.7671	6
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	7
8 14 12 10 4 7 6 1 13 9 16 3 5 2 11 15	524.4788	7
8 14 12 10 4 7 6 1 13 9 16 3 5 2 11 15	524.4788	7
8 14 12 10 4 7 6 1 13 9 16 3 5 2 11 15	524.4788	7
8 14 12 10 4 7 6 1 13 9 16 3 5 2 11 15	524.4788	7
8 14 12 10 4 7 6 1 13 9 16 3 5 2 11 15	524.4788	7
8 14 12 10 4 7 6 1 13 9 16 3 5 2 11 15	524.4788	7
8 14 12 10 4 7 6 3 13 9 16 1 5 2 11 15	524.5997	7
11 15 12 10 2 7 6 1 16 8 3 13 4 9 14 5	524.7907	7
0 11 15 12 10 4 7 6 1 13 9 16 5 3 2 14 8	525.2977	7
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	8
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	8
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	8
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	8
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	8
8 14 12 10 4 7 6 1 13 9 16 3 5 2 11 15	524.4788	8

8 14 12 10 4 7 6 1 13 9 16 3 5 2 11 15	524.4788	8
8 14 12 10 4 7 6 1 13 9 16 3 5 2 11 15	524.4788	8
8 14 12 10 4 7 6 1 13 9 16 3 5 2 11 15	524.4788	8
0 8 14 12 10 4 7 6 1 13 9 16 3 5 2 11 15	524.4788	8
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	9
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	9
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	9
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	9
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	9
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	9
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	9
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	9
11 8 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.9205	9
0 8 14 12 10 4 7 6 1 13 9 16 3 5 2 11 15	524.4788	9
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	10
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	10
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	10
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	10
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	10
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	10
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	10
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	10
8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	10
0 8 11 12 10 4 7 6 1 15 9 16 5 3 2 14 13	520.8018	10

SINGLE LAYER FULL OPTIMIZATION TEST THREE RESULTS:

Scanning Pattern	Fitness	Generation Number
5 13 3 10 14 4 16 7 9 8 6 1 2 15 12 11	542.6426	1
14 3 6 7 1 9 12 4 5 16 10 11 15 2 8 13	522.0725	1
15 10 1 5 9 6 8 13 16 12 4 14 3 7 2 11	524.6888	1
5 12 16 13 2 10 1 15 6 8 14 3 7 9 11 4	541.7777	1
16 7 3 14 4 9 15 11 13 1 2 6 8 12 10 5	542.4941	1
12 1 9 13 3 10 4 14 6 7 2 5 16 11 15 8	539.9081	1
2 5 6 15 11 3 14 1 13 12 7 10 9 8 4 16	530.5689	1
3 13 14 10 7 2 5 9 15 11 16 1 4 6 12 8	521.9986	1
15 9 2 1 8 14 4 13 12 16 7 6 10 5 3 11	521.1286	1
0 11 16 12 10 6 13 2 1 3 5 8 4 15 7 9 14	539.2106	1
10 3 2 1 8 14 4 13 12 16 7 11 15 5 9 6	516.9315	2

15 9 2 1 11 3 14 13 12 16 7 6 10 5 4 8	520.8314	2
15 13 6 7 1 9 12 4 5 16 10 14 8 2 3 11	520.8314	2
15 9 2 1 8 14 4 13 12 16 7 6 10 5 3 11	521.1286	2
3 13 14 10 7 2 5 9 15 11 16 1 4 6 12 8	521.9986	2
14 3 6 7 1 9 12 4 5 16 10 11 15 2 8 13	522.0725	2
15 10 1 5 9 6 8 13 16 12 4 14 3 7 2 11	524.6888	2
2 5 6 15 8 14 4 1 13 12 7 10 9 3 11 16	529.3367	2
2 5 6 15 11 3 14 1 13 12 7 10 9 8 4 16	530.5689	2
0 11 16 12 10 6 13 2 1 3 5 8 4 15 7 9 14	539.2106	2
10 3 2 1 8 14 4 13 12 16 7 11 15 5 9 6	516.9315	3
10 3 2 1 8 14 4 13 12 16 7 11 15 5 9 6	516.9315	3
15 9 2 1 11 3 14 13 12 16 7 6 10 5 4 8	520.8314	3
15 13 6 7 1 9 12 4 5 16 10 14 8 2 3 11	520.8314	3
15 9 2 1 8 14 4 13 12 16 7 6 10 5 3 11	521.1286	3
15 9 2 1 8 14 4 13 12 16 7 6 10 5 3 11	521.1286	3
3 13 14 10 7 2 5 9 15 11 16 1 4 6 12 8	521.9986	3
14 3 6 7 1 9 12 4 5 16 10 11 15 2 8 13	522.0725	3
15 9 2 1 11 3 14 13 12 16 7 6 10 4 8 5	522.1378	3
0 15 10 1 5 9 6 8 13 16 12 4 14 3 7 11 2	524.6798	3
10 3 2 1 8 14 4 13 12 16 7 11 15 5 9 6	516.9315	4
10 3 2 1 8 14 4 13 12 16 7 11 15 5 9 6	516.9315	4
10 3 2 1 8 14 4 13 12 16 7 11 15 5 9 6	516.9315	4
15 13 6 7 1 9 12 4 5 16 10 14 8 2 11 3	517.0456	4
3 10 2 1 8 14 4 13 12 16 7 6 15 5 9 11	517.0456	4
15 9 2 1 11 3 14 13 12 16 7 6 10 5 4 8	520.8314	4
15 13 6 7 1 9 12 4 5 16 10 14 8 2 3 11	520.8314	4
9 15 2 1 8 14 4 13 12 16 7 11 10 5 3 6	520.9726	4
15 9 2 1 8 14 4 13 12 16 7 6 10 5 3 11	521.1286	4
0 15 9 2 1 8 14 4 13 12 16 7 6 10 5 3 11	521.1286	4
10 3 2 1 8 14 4 13 12 16 7 11 15 5 9 6	516.9315	5
10 3 2 1 8 14 4 13 12 16 7 11 15 5 9 6	516.9315	5
10 3 2 1 8 14 4 13 12 16 7 11 15 5 9 6	516.9315	5
10 3 2 1 8 14 4 13 12 16 7 11 15 5 9 6	516.9315	5
10 3 2 1 8 14 4 13 12 16 7 11 15 5 9 6	516.9315	5
15 5 6 1 8 14 4 3 12 16 10 9 7 2 11 13	517.0123	5
15 13 6 7 1 9 12 4 5 16 10 14 8 2 11 3	517.0456	5
3 10 2 1 8 14 4 13 12 16 7 6 15 5 9 11	517.0456	5
15 9 2 1 11 3 14 13 12 16 7 6 10 5 4 8	520.8314	5
0 15 13 6 7 1 9 12 4 5 16 10 14 8 2 3 11	520.8314	5
10 3 2 1 8 14 4 13 12 16 7 11 15 5 9 6	516.9315	6

10 3 2 1 8 14 4 13 12 16 7 11 15 5 9 6	516.9315	6
10 3 2 1 8 14 4 13 12 16 7 11 15 5 9 6	516.9315	6
10 3 2 1 8 14 4 13 12 16 7 11 15 5 9 6	516.9315	6
10 3 2 1 8 14 4 13 12 16 7 11 15 5 9 6	516.9315	6
10 3 2 1 8 14 4 13 12 16 7 11 15 5 9 6	516.9315	6
10 3 2 1 8 14 4 13 12 16 7 11 15 5 9 6	516.9315	6
10 3 2 1 8 14 4 13 12 16 7 11 15 5 9 6	516.9315	6
10 3 2 1 8 14 4 13 12 16 7 11 15 5 9 6	516.9315	6
0 15 5 6 1 8 14 4 3 12 16 10 9 7 2 11 13	517.0123	6

FIVE LAYER FULL OPTIMIZATION TEST ONE RESULTS:

Scanning Pattern	Fitness	Generation Number
15 6 10 2 8 7 11 4 3 16 12 13 14 1 9 5	686.1648	1
13 5 4 2 8 9 15 11 16 12 10 1 7 6 3 14	691.3158	1
5 3 15 16 7 14 4 2 1 10 13 6 8 9 11 12	692.1203	1
13 11 9 4 2 7 12 16 15 3 8 6 1 10 5 14	694.5306	1
11 12 9 4 13 15 6 8 3 14 7 16 5 1 2 10	694.9699	1
15 11 5 6 10 16 2 13 9 4 12 1 7 14 3 8	697.2562	1
8 14 16 4 6 15 3 13 7 12 5 11 2 9 1 10	700.6226	1
6 5 1 9 15 4 13 12 2 16 11 7 10 8 14 3	700.7995	1
2 9 16 5 6 13 4 3 10 1 8 15 7 13 11 12	707.0362	1
7 1 3 11 6 8 13 10 5 14 15 9 16 2 12 4	710.0605	1
15 6 10 2 8 7 11 4 3 16 12 13 1 9 5 14	685.6925	2
15 6 10 2 8 7 11 4 3 16 12 13 14 1 9 5	686.1648	2
15 6 10 2 8 7 11 4 3 16 12 1 9 5 13 14	689.2971	2
13 5 4 2 8 9 15 11 16 12 10 3 14 1 7 6	690.7996	2
13 5 4 2 8 9 15 11 16 12 10 1 7 6 3 14	691.3158	2
5 3 15 16 7 14 4 2 1 10 13 6 8 9 11 12	692.1203	2
4 11 9 13 2 7 12 16 15 3 8 6 14 1 10 5	693.4291	2
13 11 9 4 2 7 12 16 15 3 8 6 1 10 5 14	694.5306	2
11 12 9 4 13 15 6 8 3 14 7 16 5 1 2 10	694.9699	2
15 11 5 6 10 16 2 13 9 4 12 1 7 14 3 8	697.2562	2
15 6 10 2 8 7 11 4 3 16 12 13 1 9 5 14	685.6925	3
15 6 10 2 8 7 11 4 3 16 12 13 1 9 5 14	685.6925	3
15 6 10 2 8 7 11 4 3 16 12 13 1 9 5 14	685.6925	3
15 6 10 2 8 7 11 4 3 16 12 13 14 1 9 5	686.1648	3

15 6 10 2 8 7 11 4 3 16 12 1 9 5 13 14	689.2971	3
13 5 4 2 8 9 15 11 16 12 10 3 14 1 7 6	690.7996	3
13 5 4 2 8 9 15 11 16 12 10 1 7 6 3 14	691.3158	3
5 3 15 16 7 14 4 2 1 10 13 6 8 9 11 12	692.1203	3
4 11 9 13 2 7 12 16 15 3 8 6 14 1 10 5	693.4291	3
13 5 15 2 8 9 11 4 16 12 10 1 7 6 3 14	693.4862	3
15 6 10 2 8 7 11 4 3 16 12 13 1 9 5 14	685.6925	4
15 6 10 2 8 7 11 4 3 16 12 13 1 9 5 14	685.6925	4
15 6 10 2 8 7 11 4 3 16 12 13 1 9 5 14	685.6925	4
15 6 10 2 8 7 11 4 3 16 12 13 1 9 5 14	685.6925	4
15 6 10 2 8 7 11 4 3 16 12 13 1 9 5 14	685.6925	4
15 6 10 2 8 7 11 4 3 16 12 13 1 9 5 14	685.6925	4
15 6 10 2 8 7 11 4 3 16 12 13 1 9 5 14	685.6925	4
15 6 10 2 8 7 11 4 3 16 12 13 14 1 9 5	686.1648	4
15 6 10 2 8 7 11 4 3 16 12 1 9 5 13 14	689.2971	4
13 5 4 2 8 9 15 11 16 12 10 3 14 1 7 6	690.7996	4
15 6 10 2 8 7 11 4 3 16 12 13 1 9 5 14	685.6925	5
15 6 10 2 8 7 11 4 3 16 12 13 1 9 5 14	685.6925	5
15 6 10 2 8 7 11 4 3 16 12 13 1 9 5 14	685.6925	5
15 6 10 2 8 7 11 4 3 16 12 13 1 9 5 14	685.6925	5
15 6 10 2 8 7 11 4 3 16 12 13 1 9 5 14	685.6925	5
15 6 10 2 8 7 11 4 3 16 12 13 1 9 5 14	685.6925	5
15 6 10 2 8 7 11 4 3 16 12 13 1 9 5 14	685.6925	5
15 6 10 2 8 7 11 4 3 16 12 13 1 9 5 14	685.6925	5
15 6 10 2 8 7 11 4 3 16 12 13 1 9 5 14	685.6925	5
15 6 10 2 8 7 11 4 3 16 12 13 14 1 9 5	686.1648	5

REFERENCES

[1] M. W. Khaing, J. Y. H. Fuh, and L. Lu, “Direct metal laser sintering for rapid tooling: Processing and characterisation of EOS parts,” in *Journal of Materials Processing Technology*, 2001.

[2] Anon, “Laser sintering ushers in new route to PM parts,” *Met. Powder Rep.*, 1997.

[3] C. H. Fu and Y. B. Guo, “Three-Dimensional Temperature Gradient Mechanism in Selective Laser Melting of Ti-6Al-4V,” *J. Manuf. Sci. Eng.*, 2014.

[4] T. E. of E. Britannica, “Sintering,” *Encyclopaedia Britannica*. 2016.

[5] A. Simchi, F. Petzoldt, and H. Pohl, “On the development of direct metal laser sintering for rapid tooling,” *J. Mater. Process. Technol.*, 2003.

[6] D. E. Gambardella, “US20160288207A1,” 2015.

[7] A. E. Patterson, S. L. Messimer, and P. A. Farrington, “Overhanging Features and the SLM/DMLS Residual Stresses Problem: Review and Future Research Need,” 2017.

[8] M. F. Zaeh and G. Branner, “Investigations on residual stresses and deformations in selective laser melting,” *Prod. Eng.*, 2010.

[9] H. Pohl, A. Simchi, M. Issa, and H. C. Dias, “Thermal stresses in direct metal laser sintering,” *Proc. SFF Symp.*, 2001.

[10] L. Parry, I. A. Ashcroft, and R. D. Wildman, “Understanding the effect of laser

scan strategy on residual stress in selective laser melting through thermo-mechanical simulation," *Addit. Manuf.*, 2016.

[11] N. Keller and V. Ploshikhin, "NEW METHOD FOR FAST PREDICTIONS OF RESIDUAL STRESS AND DISTORTION OF AM PARTS," in *Solid Freeform Fabrication 2014*, 2014.

[12] T. Mukherjee, W. Zhang, and T. DebRoy, "An improved prediction of residual stresses and distortion in additive manufacturing," *Comput. Mater. Sci.*, 2017.

[13] P. Mercelis and J. Kruth, "Residual stresses in selective laser sintering and selective laser melting," *Rapid Prototyp. J.*, 2006.

[14] I. Yadroitsava, S. Grewar, D. Hattingh, and I. Yadroitsev, "Residual Stress in SLM Ti6Al4V Alloy Specimens," *Mater. Sci. Forum*, vol. 828–829, pp. 305–310, 2015.

[15] B. Vrancken, "Study of Residual Stresses in Selective Laser Melting," 2016.

[16] I. A. Roberts, C. J. Wang, R. Esterlein, M. Stanford, and D. J. Mynors, "A three-dimensional finite element analysis of the temperature field during laser melting of metal powders in additive layer manufacturing," *Int. J. Mach. Tools Manuf.*, 2009.

[17] K. Dai and L. Shaw, "Distortion minimization of laser-processed components through control of laser scanning patterns," *Rapid Prototyp. J.*, vol. 8, no. 5, pp. 270–276, 2002.

[18] J. Robinson, I. Ashton, P. Fox, E. Jones, and C. Sutcliffe, "Determination of the effect of scan strategy on residual stress in laser powder bed fusion additive

manufacturing," *Addit. Manuf.*, 2018.

[19] C. Li, C. H. Fu, Y. B. Guo, and F. Z. Fang, "Fast Prediction and Validation of Part Distortion in Selective Laser Melting," in *Procedia Manufacturing*, 2015.

[20] K. Zeng, D. Pal, and B. Stucker, "A review of thermal analysis methods in laser sintering and selective laser melting," in *23rd Annual International Solid Freeform Fabrication Symposium - An Additive Manufacturing Conference, SFF 2012*, 2012.

[21] T. Osman and A. Boucheffa, "Analytical solution for the 3D steady state conduction in a solid subjected to a moving rectangular heat source and surface cooling," *Comptes Rendus - Mec.*, 2009.

[22] X. Zhao, A. Iyer, P. Promoppatum, and S. C. Yao, "Numerical modeling of the thermal behavior and residual stress in the direct metal laser sintering process of titanium alloy products," *Addit. Manuf.*, 2017.

[23] Z. Luo and Y. F. Zhao, "Finite Element Thermal Analysis of Melt Pool in Selective Laser Melting Process," in *Proceedings of the ASME 2018 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference IDETC/CIE 2018*, 2017.

[24] L. Li, C. Lough, A. Replogle, D. Bristow, R. Landers, and E. Kinzel, "THERMAL MODELING OF 304L STAINLESS STEEL SELECTIVE LASER MELTING," in *Solid Freeform Fabrication 2017: Proceedings of the 28th Annual International Solid Freeform Fabrication Symposium – An Additive Manufacturing Conference*,

2017.

[25] K. Zeng, D. Pal, H. J. Gong, N. Patil, and B. Stucker, “Comparison of 3DSIM thermal modelling of selective laser melting using new dynamic meshing method to ANSYS,” *Mater. Sci. Technol.*, 2015.

[26] M. M. Francois *et al.*, “Modeling of additive manufacturing processes for metals: Challenges and opportunities,” *Current Opinion in Solid State and Materials Science*, vol. 21, no. 4. 2017.

[27] K. Dai and L. Shaw, “Thermal and stress modeling of multi-material laser processing,” *Acta Mater.*, 2001.

[28] C. H. Fu and Y. B. Guo, “3-Dimensional Finite Element Modeling of Selective Laser Melting Ti-6Al-4V Alloy,” *Solid Free. Fabr. Symp.*, pp. 1129–1144, 2014.

[29] C. R. P. D. Fisher and K. P. D. Nahshon, “Simulation of Weld Mechanical Behavior to Include Welding-Induced Stress and Distortion: Coupling of SYSWELD and Abaqus Codes,” West Bethesda, Maryland, 2015.

[30] X. C. Wang, T. Laoui, J. Bonse, J. P. Kruth, B. Lauwers, and L. Froyen, “Direct selective laser sintering of hard metal powders: Experimental study and simulation,” *Int. J. Adv. Manuf. Technol.*, 2002.

[31] J. C. Steuben, A. J. Birnbaum, J. G. Michopoulos, and A. P. Iliopoulos, “Enriched analytical solutions for additive manufacturing modeling and simulation,” *Addit. Manuf.*, 2019.

[32] T. Nolan, Y. Lian, and M. Sussman, “Development of Simulation Tools for Selective Laser Melting Additive Manufacturing,” in *Solid Freeform Fabrication 2017: Proceedings of the 28th Annual International Solid Freeform Fabrication Symposium*, 2017.

[33] T. CM, “Direct laser sintering of stainless steel: thermal experiments and numerical modelling,” *PhD Thesis; Univ. Leeds, UK*, 2004.

[34] L. Li, C. Lough, A. Replogle, D. Bristow, R. Landers, and E. Kinzel, “Thermal Modeling of 304L Stainless Steel Selective Laser Melting,” *Solid Free. Fabr.*, 2017.

[35] W. King, A. T. Anderson, R. M. Ferencz, N. E. Hodge, C. Kamath, and S. A. Khairallah, “Overview of modelling and simulation of metal powder bed fusion process at Lawrence Livermore National Laboratory,” *Mater. Sci. Technol.*, vol. 31, no. 8, pp. 957–968, 2015.

[36] M. Saraswat and A. K. Sharma, “Genetic Algorithm for optimization using MATLAB,” *Int. J. Adv. Res. Comput. Sci.*, vol. 4, no. 3, 2013.

[37] M. Matyjaszczyk, “TSP using Genetic Algorithms in MATLAB.”

[38] A. J. Umbarkar and P. D. Sheth, “Crossover Operators in Genetic Algorithms: A Review,” *ICTACT J. Soft Comput.*, vol. 6, no. 1, 2015.

[39] H. Braun, “On solving travelling salesman problems by genetic algorithms,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial*

Intelligence and Lecture Notes in Bioinformatics), 1991.

[40] A. Majdi, A. Rani, R. Fua-Nizan, and M. Yazid Din, "Manufacturing Methods for Medical Protheses - A Review," *Int. Med. Device Technol. Conf.*, 2017.

[41] MatWeb, "Titanium Ti-6Al-4V (Grade 5), Annealed." .

[42] J. J. (Concurrent T. C. Valenica and P. N. (National P. L. Quested, "Thermophysical Properties," *ASM Handb.*, vol. 15, pp. 468–481, 2008.

[43] K. Zeng, D. Pal, H. J. Gong, N. Patil, and B. Stucker, "Comparison of 3DSIM thermal modelling of selective laser melting using new dynamic meshing method to ANSYS," *Mater. Sci. Technol.*, vol. 31, no. 8, pp. 945–956, 2015.

[44] Abaqus, "Abaqus Version 6.6 Documentation." .

[45] R. L. Ott and M. Longnecker, *An Introduction to Statistical Methods and Data Analysis*, Sixth. Brooks/Cole Cengage Learning, 2010.

[46] "*DFLUX." .

[47] Simulia, "34.4.4 Thermal Loads." [Online]. Available: http://dsk.ippt.pan.pl/docs/abaqus/v6.13/books/usb/default.htm?startat=pt07ch34s04aus123.html.

[48] Y. Yaman and S. Kakaç, *Heat Conduction*, Fourth. Taylor & Francis, 2008.

[49] H. Peng *et al.*, "Optimization of Build Orientation for Minimum Thermal Distortion in DMLS Metallic Additive Manufacturing," in *Solid Freeform*

Fabrication 2017: Proceedings of the 28th Annual International Solid Freeform Fabrication Symposium, 2017.

[50] G. Papazafeiropoulous, M. Muniz-Calvente, and E. Martinez-Paneda, "Abaqus2Matlab: a suitable tool for finite element post-processing (submitted)." 2016.

[51] J. Finkelstein, "OrderedCrossoverFunction.java." jmona, 2009.

[52] A. Hassanat, K. Almohammadi, E. Alkafaween, E. Abunawas, A. Hammouri, and V. . S. Prastah, "Choosing Mutation and Crossover Ratios for Genetic Algorithsm-A Review with a New Dynamic Approach," *Information*, 2019.

[53] N. M. Razali and J. Geraghty, "Genetic algorithm performance with different selection strategiesin solving TSP," in *Proceedings of the World Congress on Engineering 2011, WCE 2011*, 2011.