Infill Optimization for Additive Manufacturing

Rio Hall-Zazueta, Ellie Prince, Andrew Zerbe
CS361: Engineering Design Optimization (Stanford University)

Abstract

Generative design tools provide users with the ability to generate optimal mechanical parts given loading conditions, but existing software is closed-source, complicated to use, and can ignore design intent. Here we show initial development of a more robust and usable method using truss-based modeling. We developed and compared two potential optimization algorithms–a genetic algorithm and a node replacement algorithm–that utilize this method to generate mass-optimal infill structures for FDM 3D printing. Compared to grid infill, both algorithms reduced strain energy by over 70% at the same mass. The genetic algorithm also generated an approximate Pareto frontier, which a designer could use to choose an appropriate balance of low weight and high stiffness. Optimization of truss-based infill is shown to be a powerful tool to improve the performance of 3D-printed parts without additionally complicating the design process.

Problem

Generative design software is a capable tool for creating 3D models, but is not often used outside of specific industrial applications due to the fact it is closed-source, computationally intensive (and thus time-intensive), and often difficult to mold to one's design intent (e.g. ensuring that holes are supported or that the shape of the design is maintained by setting excluded and preserved regions) (Nordin 2018).

3D printed infill is particularly suited to generative design. By default, Fused Deposition Modeling (FDM) 3D printers create sparse structures inside the solid shell of a printed part to maintain its strength while reducing cost and print time. The designer chooses the pattern and uniform density of this infill, serving as a way to adjust the part's strength-to-mass ratio. However, many of these infill patterns are simplistic; they do not take the part's expected loading conditions into account (Fig. 1). More advanced users may create variable infill (often using manual workarounds such as support blockers) to use different patterns (e.g. linear, grid, honeycomb, etc.) and densities in different parts of the print, but this process is labor-intensive and frequently requires the designer to use different techniques for each part.

This work proposes generative design techniques to create mass-optimal infill structures given a part and its expected loading. Rather than parameterizing and optimizing existing infill patterns, our method generates a set of load-bearing infill elements that comprise an optimal pattern. This focus on infill ensures that the overall object's outer shape is maintainded during the optimization, which preserves design intent wihtout requiring additional specification of the geometry. Because the part's outer shape is constant, this also shifts our focus from designing a pattern that is exactly optimal for the load case, to improving the performance of existing infill strategies, ideally while reducing computation time. Given our limited time, we focused on creating 2D infill structures as a proof of concept; however, the algorithms presented can and should be generalized to 3D for real-world usage.

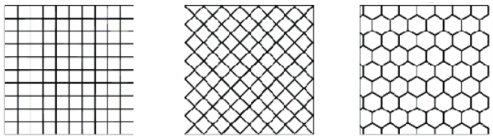

Figure 1: Infill patterns including linear (left), grid, (center),and hexagonal/honeycomb (right) (Dey, 2019).

Literature Review

Many works have taken different approaches to investigating and optimizing infill in additive manufacturing.

One common area of investigation is topology optimization, an algorithmic process that finds efficient designs based on a set of constraints by removing material from the design and keeping track of the number of connected components/boundaries. In their 2019 paper, García-Domínguez, Claver, and Sebastián run topology optimization using Rhinoceros "Grasshopper" software, demonstrating how existing physics-based simulation libraries can prove useful in this application (García-Domínguez, Claver, and Sebastián 2019). In a 2022 paper, Liu et al. examine topology optimization for prototyping a multimaterial-like compliant finger object by varying printing infill density, creating a complex objective function based on mechanical properties and then optimizing (Liu, Chen, and Yang 2022). They reference that the traditional objective function minimizes "strain energy" (SE) for rigid structures and minimizes "mutual potential energy" (MPE) for compliant mechanisms. The authors

propose that a design can be approximated by varying the values of infill densities in different portions of a 3D-printed component–in short varying choice of regions of default infill with different parameters, but still the same patterns.

Other literature focuses more on identifying design variables and optimal design than algorithms. In their 2019 article, Shmitt, Mehta, and Kim investigate additive manufacturing infill optimization for automotive components by running various trials and finding optima via experimental results (Schmitt, Mehta, and Kim 2020). They find that solid, double dense, and triangular infill, all with eight contour layers, are optimal for component regions experiencing high stress, moderate stress and low stress, respectively.

These works' identifications of design variables, uses of physics-based simulation libraries, and optimization strategies show promise in infill optimization but also leave open the opportunity to examine optimization of infill pattern itself rather than simply hyperparameters of default patterns.

Approach

Our approach can be split into two parts: evaluation and generation of candidates to be evaluated, and optimization of candidates.

Evaluation and Generation of Candidates

Our goal was to create mass-optimal designs given loading conditions. This meant evaluating mass and some proxy for strength; in this work, we used strain energy (SE) because of its established use in previous literature (Liu, Chen, and Yang 2022). Because we are trying to optimize both mass and strain energy, this is a multi-objective optimization problem, so we evaluate our optimized designs in the criterion space to determine if they are Pareto-optimal (Kochenderfer and Wheeler 2019).

Mass can be simply evaluated by determining the volume of any pattern and multiplying by material density. Typically, FEA tools would be used to evaluate stress within a model in response to loading and return SE. However, these tools are computationally expensive, as they create an exhaustive mesh of the solid object. We instead chose to represent candidate designs as 2-dimensional truss structures, which can easily be evaluated with the Direct Stiffness method. By constraining our designs to truss shapes, we were likely to generate less mass-optimal designs than a strategy such as topology optimization; however, trusses are still well suited to act as support structures and evaluate orders of magnitude faster. Additionally, rather than creating meshes for each design, which is time-consuming, each design can be evaluated as a truss and then the final design meshed for slicing and printing only at the end.

The printed shape optimized was a 2D square box with side lengths of 50 millimeters (i.e. the cross-section of a 3D-printed 50-millimeter cube). "Nodes" represented points on the design at which "elements," or linking pieces of 3D-printed material, intersected. To create trusses that are always stable without elements that intersect each other, we first generated a 2D array of nodes, and then created a Delaunay triangulation to generate the elements that connect them (sci 2023) (Fig. 2).

To generate 3D models for 3D printing from our 2D truss representations, we developed a script utilizing CADQuery, an open-source software package for CAD based on Python code (cad 2015). Our CADQuery script was input a list of element start and end points, and generated a union of cylinders corresponding to each item on the list. This was then exported from CADQuery in STL format.

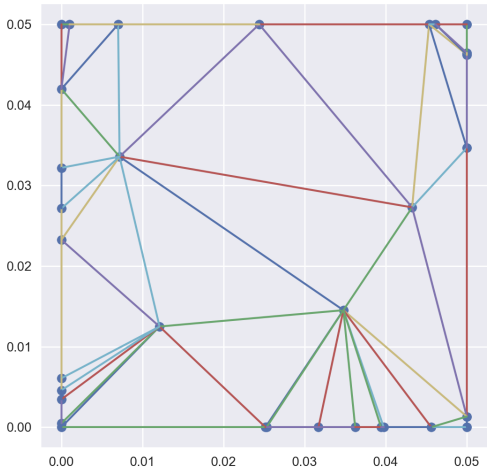

Figure 2: Example of a randomly generated truss structure with nodes (circular points) and elements (lines connecting nodes).

Optimization Algorithms

Custom Iterative Replacement Algorithm

A custom "replacement algorithm" was created as an alternative to classic optimization strategies later used. For a chosen number of iterations, the algorithm does the following:

- Locate the truss node with minimum utility (minimum force sustained).
- Move this "minimal contributor" node to a new random candidate location.
- Re-solve the truss with this new node, and measure a sum of mass and strain energy (equation below).
- If this metric is lower than before the node was moved, accept the new node location. If no improvement, retry other random candidate locations for n more attempts, returning the original node if no improvement even after n attempts.

The final truss is returned in the form of the remaining nodes and linking elements.

Non-Domination Genetic Algorithm

A second algorithm leverages genetic algorithms (GA). These algorithms mimic the natural process of evolution with a 5-step process (Kochenderfer and Wheeler 2019):

1. Generation of an initial population of candidate designs
2. Selection of sets of two parents based on objective function performance

3. Crossover of each set of parents, resulting in (on average) better-performing child candidates than the previous generation
4. Mutation of child candidates, in order to encounter new designs not present in the initial population
5. Repeat steps 2-4 for a specified number of iterations.

It is crucial to the success of a GA strategy that its candidate representation and crossover methods maintain the structure of the parent candidates; in other words, crossing over two successful parents should tend to preserve the essence of what made each parent perform well.

We have chosen to implement the GA strategy as follows. Starting with a uniform grid of nodes and adjoining elements, each point is warped randomly within a small area centered on its original location. Because nodes between candidates correspond in terms of rough position, this is a generalizable representation of each candidate. Some nodes are deleted based on random probability (typically set to 20%).

After this initial population generation, a selection phase begins. Strain energy and mass are evaluated for each candidate in the current population using internal stresses calculated by the slientruss3d Python package (Cheng 2021). The "level" of each candidate is found: Candidates on the naive Pareto frontier are set to Level 1, candidates on the naive Pareto frontier (excluding Level 1 candidates) are set to Level 2, etc. Parents from candidates in Level 1, or from the lowest-level candidates in the first half of the population (if there are not enough Level 1 candidates), are then selected (Fig. 3). This ensures a well-distributed approximation of the Pareto frontier.

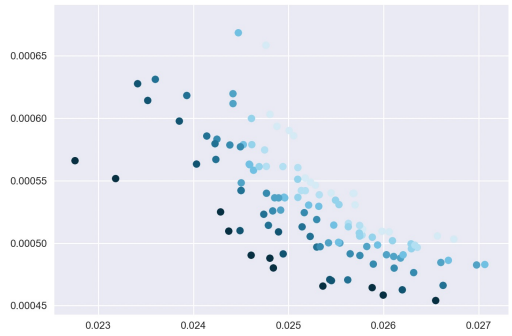

Figure 3: Example of non-domination ranking, where closer to Pareto-optimal points (represented with darker shades) are chosen first as parents.

During the crossover phase, single-point crossover occurs. Here, one region of the truss has the structure of the first truss, and the other region has the structure of the second. Uniform crossover, in contrast, is undesirable because truss point locations only make sense in the context of their neighbors, not in isolation; uniform crossover would not preserve the successful attributes of parent candidates.

Lastly, during the mutation phase, each point is translated in both dimensions according to random numbers from a normal distribution, subject to being inside the convex hull of the input shape. This is similar to the initial warping step, but the translation scale is much smaller. Additionally, points are not deleted; because they cannot be reborn, deleting points over time would tend towards removing every point from all the candidates in the population.

Weighted Sum Genetic Algorithm

To validate the effectiveness of our non-domination selection for generating an approximate Pareto frontier, we compared it with an otherwise equivalent genetic algorithm that used a weighted sum of our two objectives (mass and SE) to select parents. Because this algorithm determines one result, rather than a population of Pareto-optimal results, we ran it several times over the range of weight vectors from $(0, 1)$ to $(1, 0)$. This theoretically swept any convex regions of the Pareto curve. As shown in the plotted evaluations of the two methods, they generated a similar result, validating each other. The non-domination selection method is preferable, because it results in a more varied distribution along the approxiamte Pareto frontier (Fig. 4).

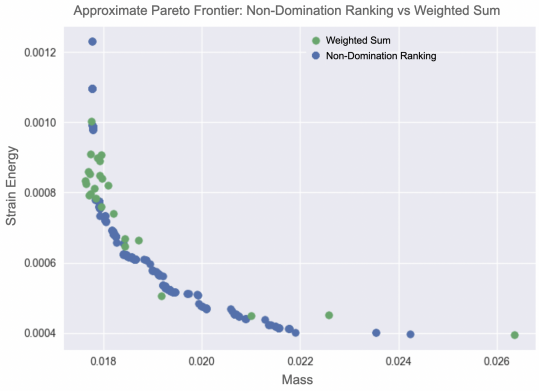

Figure 4: Approximate Pareto frontier as generated by genetic algorithm with non-domination and weighted sum selection methods.

Results and Discussion

To test and compare the performance of the algorithms, we chose a simple case where a distributed, downwards (compressive) load was applied to the top of the box (our input model). The approximate Pareto frontier generated by the non-domination algorithm is plotted along with representative examples of generated designs at different points along the frontier (Fig. 5).

Both algorithms were also plotted for comparison (Fig. 6). The replacement algorithm finds some points superior to the genetic algorithm's pseudo-frontier. The replacement algorithm, however, also shows much more variable performance: While it champions the most optimal point, it still produces more points dominated by the genetic algorithm

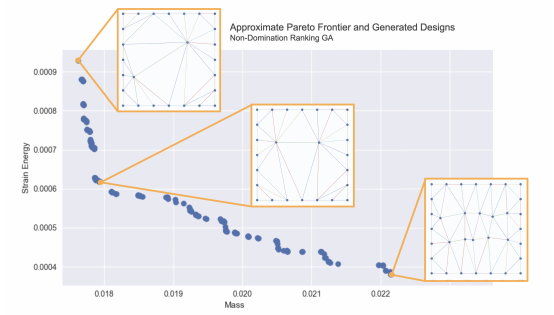

Figure 5: Pareto frontier generated by the non-domination algorithm, with several example designs highlighted and plotted.

than not. The genetic algorithm, on the other hand, benefits from being a population method and shows a frontier-like spread of points, demonstrating a clear tradeoff between mass and strain energy from a single run of the algorithm. In contrast, it takes many evaluations of replacement to ensure good points are generated. The genetic algorithm also covers more of the tails of the curve than the replacement algorithm.

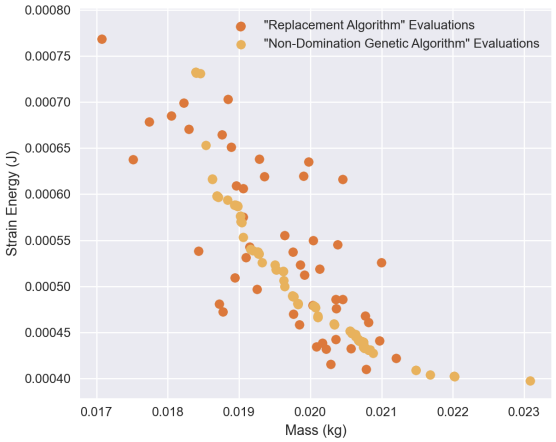

Figure 6: Evaluations of both optimization algorithms, with objectives mass and strain energy on axes.

Both of the algorithms far outperform a standard grid infill pattern one might find on a common 3D-printing software platform (Fig. 7). This pattern was modeled in the same node-element method as the algorithms written.

It should be noted as well that both algorithms take as input multiple hyperparameters that characterize the searches: number of iterations, weights, etc. While the hyperparameters for the results above were chosen after brief experimentation to put the algorithms on roughly equal footing, tuning these further could alter both algorithms' performance.

For physical visualization of the optimal infill patterns, three of the designs generated via the non-domination genetic algorithm were 3D printed (Fig. 8). These designs visually demonstrate how infill density can be varied by the designer, while each part is still-mass optimized.

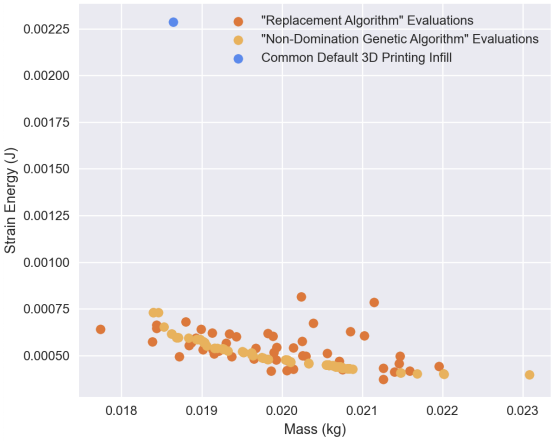

Figure 7: Evaluations of both optimization algorithms in comparison to the same evaluation of a common default 3D-printing infill.

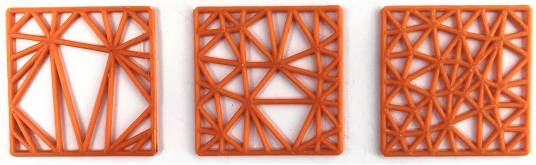

Figure 8: Physical 3D-printed prototypes of the infill patterns generated by the optimization algorithms. The leftmost design is on the low-mass side of the Pareto curve; the two designs to its right progressively increase mass and decrease SE.

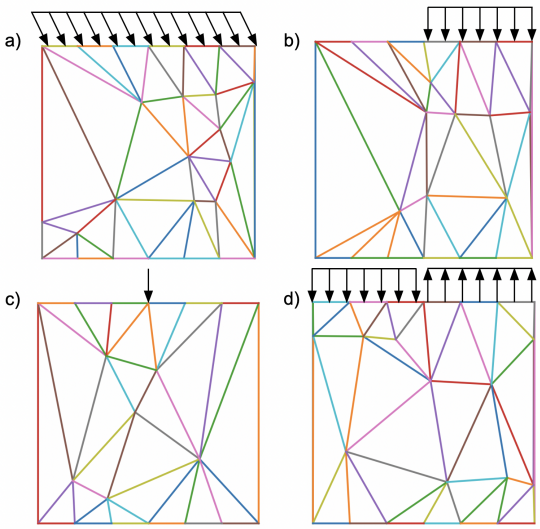

Figure 9: Optimal infill designs for a) angled distributed load, b) one-sided load, c) point load, and d) twist load.

To further test the effectiveness of the genetic algorithm, we generated sets of designs for four new load cases: angled distributed load, one-side distributed load, point load,

and twist load (Fig. 9). Our intent was to determine if the optimization process yielded part designs that were substantively different depending on loading; if not, that would indicate that infill patterns do not necessarily need to be load-specific. The optimization algorithm did generate significantly distinct results for each load case. These results intuitively match their load cases. The tilted load design has many members at the exact angle of the tilted force; the design with loading on the right side has almost all nodes on the right side as well; the point loading design has a triangular distribution of nodes starting at the force application point and widening out at the base of the part; and the twist loading design is roughly 180° rotationally symmetric (like its loading pattern).

Conclusions and Future Work

In this paper, we have implemented an evolutionary method and a custom method for optimizing 3D-printed infill structures given specified loading cases. We then evaluated the algorithms on a simple 2D loading case. The results show that truss-based optimization of custom infill patterns offers designs that can massively outperform standard grid infill patterns in terms of mass and strain energy (SE)–by a factor of 3 or more. We also found that our method's outputs substantially vary with different load cases, showing that the optimality of a structure is highly dependent on loading and that one-size-fits-all infill patterns will fall short in this respect. Finally, we demonstrated that our evolutionary algorithm can find a well-populated approximation of the Pareto curve, allowing designers to select their desired trade-off between mass and strength.

Countless avenues remain for future work on this subject. First and foremost, this 2D investigation should be expanded into a software package that generates fully 3D trusses for arbitrary 3D geometry. Optimizing the code for fast runtime will be crucial when evaluating more complicated 3D design problems.

The algorithms could also be made more complex. Slight modifications could allow the algorithms to remove nodes on bottom and top surfaces, which would allow for fewer linking elements and therefore lighter mass. We could allow the algorithms to vary the diameter of individual elements. We would expect these changes, in addition to more refined hyperparameter tuning (eg number of iterations, crossover method, and amount of node displacement in mutation), to help resolve discrepancies between the best performing replacement algorithm points and genetic algorithm generated pseudo-frontier. Expert selection of points from the approximate Pareto frontier could also be used to identify suggested regions for desired design applications.

Constraints could be added to enhance realism. Printability constraints could be considered and added; for example, 3D printers cannot feasibly print diagonal truss elements with low overhang angles, so a minimum angle of elevation for elements could be set. A constraint on compressive elements to avoid buckling could also be added.

Group Contributions

The three-person team is comprised of two students registered for 4 units (Rio, Andrew) and one student registered for 3 units (Ellie). The base project of the problem setup and custom optimization was created and implemented by the entire group, and additional time was spent by the two students registered for 4 units on implementing the genetic algorithm and variations. Individual contributions included the following:

Rio:

- Ideated node/element representation of infill in 2D.
- Wrote code for infill design generation.
- Contributed to implementation of truss analysis on candidate designs.
- Wrote code for non-domination ranking Pareto frontier in genetic algorithm.
- Generated and analyzed standard infill patterns for comparison.
- Generated plots for optimized designs on approximate Pareto frontier.
- Edited report.

Ellie:

- Set up optimization representation (constraints, design variables, objective function, etc.).
- Wrote code for replacement-strategy optimization algorithm.
- Created comparison plots for optimization algorithms.
- Drafted report.

Andrew:

- Ideated general project topic and setup.
- Contributed to implementation of truss analysis on candidate designs.
- Wrote code for genetic algorithm and weighted-sum Pareto frontier.
- Wrote code to generate printable models from trusses, and 3D printed physical versions of optimized trusses.
- Created modifications to genetic algorithm to allow boundary point removal.
- Tested optimization code on different loading conditions.
- Edited report.

References

[cad 2015] 2015. Cadquery.

[Cheng 2021] Cheng, S.-C. 2021. slientruss3d : Python for stable truss analysis and deep learning research.

[García-Domínguez, Claver, and Sebastián 2019] García-Domínguez, A.; Claver, J.; and Sebastián, M. A. 2019. Infill optimization for pieces obtained by 3d printing. *Procedia Manufacturing* 41:193–199. 8th Manufacturing Engineering Society International Conference, MESIC 2019, 19-21 June 2019, Madrid, Spain.

[Kochenderfer and Wheeler 2019] Kochenderfer, M. J., and Wheeler, T. A. 2019. *Algorithms for optimization*. Mit Press.

[Liu, Chen, and Yang 2022] Liu, C.-H.; Chen, Y.; and Yang, S.-Y. 2022. Topology optimization and prototype of a multimaterial-like compliant finger by varying the infill density in 3d printing. *Soft Robotics* 9(5):837–849. PMID: 34619072.

[Nordin 2018] Nordin, A. 2018. Challenges in the industrial implementation of generative design systems: An exploratory study. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 32:16–31.

[Schmitt, Mehta, and Kim 2020] Schmitt, M.; Mehta, R. M.; and Kim, I. Y. 2020. Additive manufacturing infill optimization for automotive 3d-printed abs components. *Rapid Prototyping Journal* 26:89–99.

[sci 2023] 2023. scipy.spatial.delaunay.