In Learning we Truss: Structural Design Optimization Using Deep Learning

Vicente Ariztia
Graduate School of Business
Stanford
variztia@stanford.edu

Amber Yang
Computer Science
Stanford
yanga@stanford.edu

Abstract

We focused on a special problem in structural design optimization: that of choosing an efficient allocation of material volume along a given set of elements connecting the nodes in a truss. This is known as sizing optimization and has been traditionally solved with non-linear mathematical programming, approximations, and insights from the Finite Element Method. This is an important research topic and has received a lot of attention in the last decades. Recently, evolutionary algorithms and neural networks have been applied to improve the applications in large and complex trusses.

Here, we explore a new approach using a single Deep Neural Network. First, we train the network to learn the non-linear relationship between the different cross-section areas of the elements in the truss, and the corresponding performance, which incorporates the total volume and the structure's stiffness. Then, we use the same network to find a new truss with a high performance. We show that the method works, significantly improving the structure's performance and can be further generalized to include non-linear optimization among others.

1. Introduction

Trusses have been widely used in civil and mechanical constructions for at least 200 years. We can model a truss as a set of nodes connected by a set of elements. The elements are bars of a certain length ('L'), a certain material, and (in our case) a constant cross-section area ('A'). The nodes connect one or more elements together and therefore place restrictions on the relative displacements of the elements' ends.

1.1. Structural analysis of trusses

When doing linear analysis, for any single element, the relationship between the applied force ('F') and the corresponding deformation ('δ') is $\frac{F}{\delta} = \frac{A \cdot E}{L}$, where 'E' is a constant associated to each material. In this case, both the force and the deformation occur along the axis of the element and this ratio can be understood as the stiffness of the element. This means that the higher the area (and the lower the length), the stiffer the element and vice versa.

To calculate the forces and displacements in all the elements and nodes of a truss, we apply cinematic transformations to element-specific matrices and vectors, which specify applied forces, displacements, and stiffness. This process ends with a matrix linear system $F = K \cdot u$ for the whole structure, where 'F', 'K' and 'u' are applied forces, stiffness matrix and displacements in all the degrees of freedom (vertical and horizontal displacements) of the structure. The solution of this system gives us resulting displacements, internal forces of the elements and reaction forces in restricted degrees of freedom.

1.2. Optimization of trusses

For a given set of nodes and connecting elements, we can ask ourselves which distribution of cross-section areas the elements should have. Should we have uniform cross-section areas or some thicker elements? Which elements should be thicker than the others? To answer these questions, we need an optimization criterion (objective function and restrictions) and some algorithm.

Traditionally, structural optimization has typically used as criterion the minimization of total weight (or volume of material) subject to certain restrictions on displacements and/or internal forces. Regarding algorithms, an early efficient method was developed in the 1970's and consist of non-linear mathematical programming using the Finite Element Method (numerical solutions of elasticity problems) as a parallel tool [1]. However, for certain problem sizes and complexities, this method does not deliver great results. More recently, non-gradient methods, like evolutionary algorithms, have been developed and have very good results despite their higher computational costs [2]. Genetic programming has achieved good results in sizing optimization (our focus) and in topology optimization (which we briefly discuss at the end). Deep Learning has also been used, mostly recently, for structural truss optimization and it also showed promising results. The approach taken by Nguyen, et. al. [3] was to build two neural networks, one to learn the Finite Element Method approximations of displacements and for a given structure,

and another one to find the optimal (minimum volume) structure. They used computer generated examples of trusses as the dataset.

1.3. The problem

For a given truss, if we assume linear behavior of the force-displacement relationship of every element, as well as relatively small deformations, there is a corresponding linear system $F = K \cdot u$ that connects forces and displacements. However, the displacements and forces depend on the cross-section areas of every element in the truss and that relationship is non-linear. Exhibit A shows the analysis and optimization problem for a simple truss, which can even be done analytically.

We focused on the question regarding how to efficiently allocate construction material to maximize the performance ('P') of a generic truss, just like the example of the simple truss. 'P' was defined as the ratio between the structure's overall stiffness ('S') and its total volume ('V'). 'S' was defined as the ratio between the applied vertical force ('F') and the corresponding vertical displacement in the same node ('v').

Given a set of nodes and elements, for every distribution of volume (i.e. a vector containing the cross-section areas of every single element), there is a corresponding scalar 'P'. This relationship is non-linear, and our problem consists of finding a distribution that achieves a high level of 'P'.

2. Our algorithm

In defining an objective function that incorporates both the total volume and the corresponding stiffness, we made the optimization criterion simpler than the traditional approach, which seeks to minimize the total volume while setting restrictions on displacements and internal forces. But our focus was on the algorithm, which we explain here.

First, we generated a large number of random examples of a generic truss, each consisting of 625 numbers that correspond to the cross-section area of each of the 625 elements. These 625-dimension vectors were our $x^{(i)}$ examples in the dataset. We used the Finite Element Method to compute displacements and internal forces for every example, and consequently computing the performance ('P') which is the scalar. The output $y^{(i)}$ is a linear transformation of 'P'.

Second, we trained a Deep Neural Network to learn the relationship between the distribution of cross-section areas and the performance of the generic truss. This is what we call the first stage of our algorithm.

Finally, we used the same network to modify an initially uniform truss (all cross-sections equal) and increase its performance.

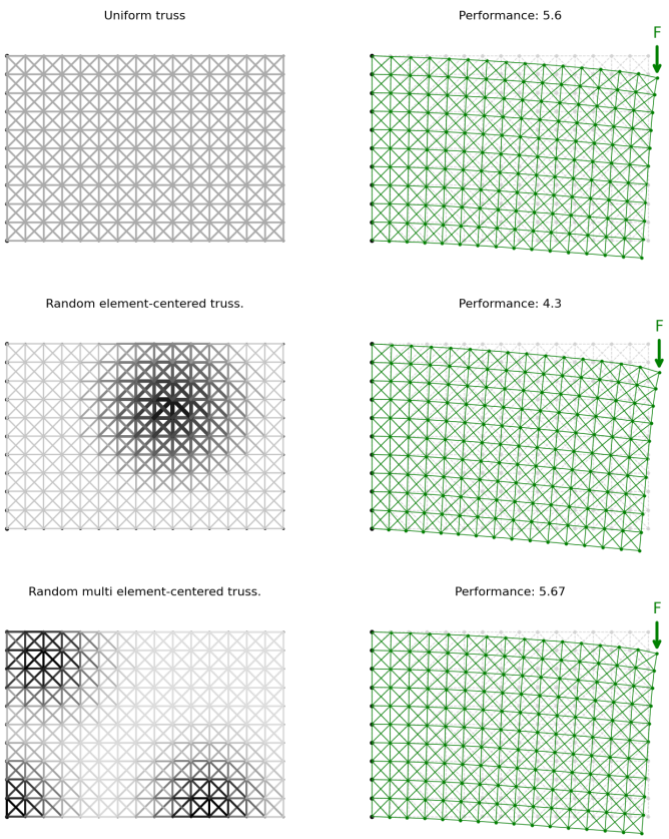

Figure 1: A uniform truss (all cross-section areas equal), and two examples of randomly generated trusses used for the training of our algorithm. The right-hand side of each row show the deformation of the structure and its performance. Darker and thicker elements mean higher cross-section area.

2.1. Generation of the dataset

For generating the different training examples $x^{(i)}$, the general strategy was to introduce variability in terms of cross-section areas as well as the corresponding performance.

We used a geometrically inspired strategy to generate trusses with various levels of performance. This means that we randomly reinforced certain areas of the truss by giving a higher-than-average (or lower-than-average) cross-section area to elements in a specific vicinity. The trusses were generated with a random number of theses "hubs" ranging from 1 to 10, each of which had a central element, a certain dispersion (i.e. how far away from the central element the cross-section area gets affected), and a modulator (i.e. how different is the cross-section area of elements in the hub compared to elements far away.)

A total of 47,500 examples were generated using this technique. 98% of which were used to train the Deep Neural Network in the first stage and the remaining 2% were

evenly split to form dev. and test sets.

Our random generation algorithm achieved a high level of variability in terms of cross-section area for every single element, as well as corresponding performance.

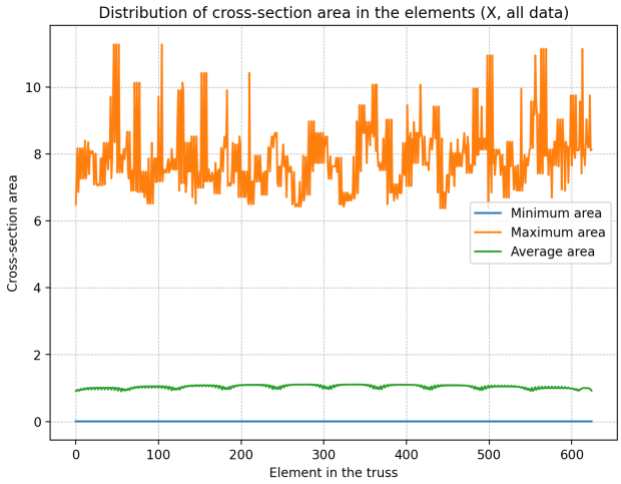

Figure 2: The distribution of the input data shows the high level of dispersion of cross-section area for every element, as well as a similar distribution for every element in the generic truss.

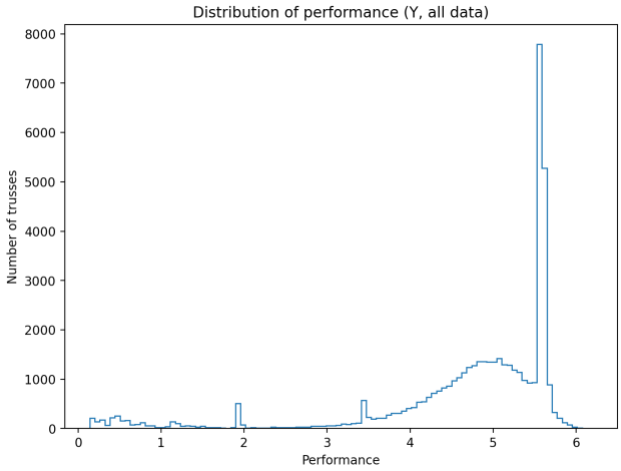

Figure 3: Distribution of performance in the dataset.

The performance of the examples ranged from 0.14 to 6.19. For reference, a uniform cross-section area truss of these characteristics (height, length, number of nodes, number of elements and load) has a performance of 5.60, which explains the high frequency around this value. Our goal is to find a truss that significantly improves this number.

2.2. Network architecture and optimization algorithm

We used a single neural network for training the dataset and for generating the optimal example. We used a deep network of purely fully connected layers to (i) first, learn the multivariable and non-linear relationship between the 625 cross-section areas and the corresponding performance, and (ii) second, find the optimal 625 distribution of cross-section areas that deliver a high level of performance.

The architecture of the network is special because we used the same network for two different tasks: first, learn the parameters of all the network except for the so called "Key layer". These parameters are trained using the training set of 46,550 (98% of 47,500) examples of trusses and their corresponding performance. In this first stage, the parameters of the Key layer are frozen and set to be: (i) an identity matrix for the weights, and (ii) a vector of zeros for the biases. In other words, during this stage, the Key layer is fully transparent: it is simply letting the different values of $x^{(i)}$ to pass forward and then the gradients to backpropagate.

Given that the values of performance in the dataset range from 0.14 to 6.19, we normalized them into a new space where the range goes from 0 to 1 because we used a cross-entropy loss function. However, since we are aiming at outperforming our dataset, we use a "target 'y' value" as the high end of the new space which was set to 9.0 based on iterative experience. This means that obtaining a 0 in the output layer is equivalent to the worst training example and obtaining 1 is equivalent to a high-performance truss. We decided to focus on the evolution of the Mean Absolute Error between $\hat{y}^{(i)}$ and $y^{(i)}$ because the output has a continuous range between 0 and 1. The MAE in the dev. set

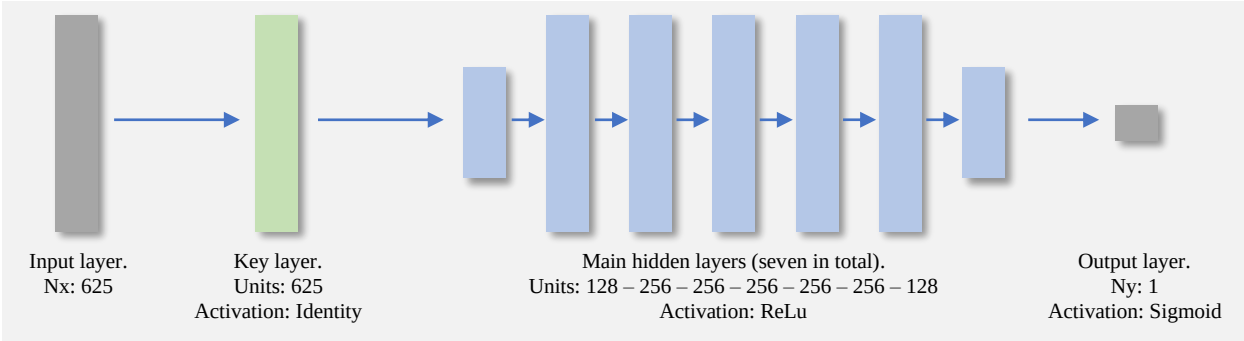

Figure 4: Network architecture.

was 0.025, slightly above the one in the training set, so we achieved a low variance in the first stage.

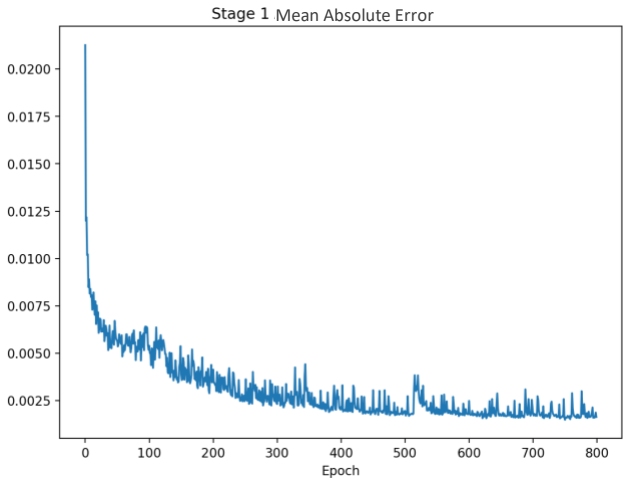

Figure 5: Optimization metric during training of first stage.

We used the Adam optimizer with a learning rate of 0.001, β_1 of 0.9 and β_2 of 0.999. These hyperparameters worked better every single time we tried with different ones. Others, such as the number of epochs and network size, were tuned by experience.

Once the network is trained, i.e. having finished the first stage, we freeze all the layers except for the Key layer, whose biases are now initialized with ones and the weights are set to be the identity matrix again. As it will become obvious, the weights will not matter. Now, we use the same Adam optimizer and the same loss function to train the network with one single "training example". The $x^{(1)}$ is a vector of zeros and the $y^{(1)}$ is a number 1. Here we are forcing the algorithm to produce biases in the Key layer that act as the different units / coordinates of the input layer, since all the values in $x^{(1)}$ are zero. And we do so with an output value of 1 since we are aiming at the target performance. Now, we also add L_2 regularization to the Key layer biases so that there is additional pressure for the biases to be small and hence to improve performance (since the biases of this layer are the cross-section areas of the optimized truss).

We used Python as the programming language and the TensorFlow library with Keras as the user interface framework.

2.3. Results

Our algorithm successfully produced highly optimized trusses, achieving the target performance. Exhibit B contains a brief discussion regarding the technical aspects of the optimized design, including the probable reasons why certain areas were reinforced and others weakened.

A simple way to interpret our result is that, with the same material volume, a uniform truss will exhibit a deformation

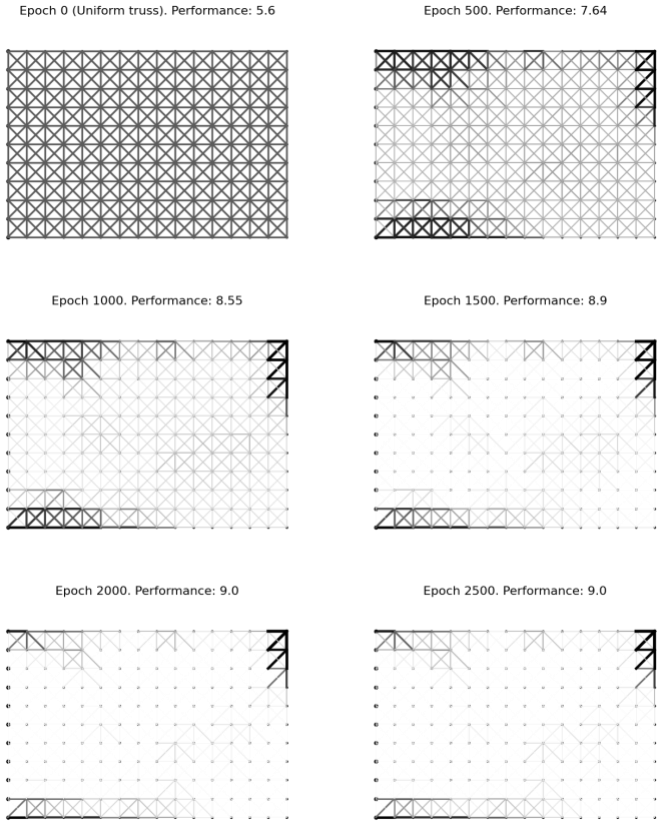

Figure 6: Evolution of the truss during the second stage. It starts as a uniform truss with performance 5.6 and ends as a highly optimized truss with the target performance of 9.0.

64% higher than our optimized truss for any given applied force. Figure 6 shows the evolution of the optimized truss from the initialization (uniform) to the final step of the second stage.

2.4. Hyperparameter discussion

As mentioned above, the Adam hyperparameters were slightly modified to analyze how sensitive the model was, but we decided to maintain the default values because no improvements were seen.

Other hyperparameters were defined using a search algorithm that ran the two stages of the model 150 times with different configurations. These hyperparameters were:

a. The number of layers and hidden units.
b. The L_2 regularization parameter for the biases of the Key layer, which was set to be $5 \cdot 10^{-4}$ for the best results.
c. The initialization values of the biases of the Key layer, which were set to be all ones for the best results.
d. Finally, the minimum cross-section area. Our algorithm naturally reduces the cross-section area of certain elements. However, we cannot allow the area to reach zero, since that would produce a singular stiffness

matrix and therefore, we would not be able to analyze the truss as a structure. Hence, during the optimization process, we set as restriction that the biases in the Key layer could not be less than a certain threshold. This threshold was tuned and ended up being 0.12.

3. Further research

In the available literature and in the present work, we see promising results for further research in the area, namely, the application of Deep Neural Networks to structural design optimization. As we learnt through the development of the present work, the most challenging part can be the generation of the data, the selection of the performance criteria, and the optimization algorithm with the hyperparameters associated to the overall architecture.

On a positive account, our architecture and optimization algorithm could be immediately applied to non-linear structural optimization with no incremental cost in the deployment, which could be especially promising. This is so because the much higher costs associated to calculating trusses with non-linear behavior will happen at the data generation stage, where calculating the performance for every truss will take longer. But once we have generated enough $x^{(i)}$, $y^{(i)}$ pairs and trained the model, it should take very little computational effort to calculate the best design considering non-linear behavior.

One immediate step for further development is the improvement of the optimization criteria: we seek to maximize a single construct for the truss that incorporates the total volume and the displacement in a certain point—the performance. We need to normalize this value and aim at a fixed target to use a cross-entropy loss function. The obvious drawback of this approach is that it might not be as strong as we would like in going even further in terms of maximum performance. We do not know if our algorithm is generating the best possible truss, even with the criteria that we are using, and we need to manually try to increase the target performance to do so.

Another point of further development is the generalization of the truss geometries. Even though our algorithm is general in the sense that we could do the training and second stage optimization with any geometry, we could extend the power of it by training weights in the first stage that can generalize the optimization of the second stage. We could train the algorithm with many different geometries, not only rectangular, as well as different restricted nodes (connections with the environment). This way, we could be able to generate an optimal truss of any geometry by using the same pre-trained model. This could also help us to extend the algorithm from the current sizing optimization (defining the cross-section areas) to also include topology optimization (which elements to include or where to add more).

Finally, our analysis was performed for a static single load, and it could be extended for multiple loads and dynamic analysis too.

4. References

[1] L. A. Schmit, H. Miura, 1976, “A New Structural Analysis/Synthesis Capability-ACCESS 1” https://arc.aiaa.org/doi/10.2514/3.61405

[2] Hirad Assimi, Ali Jamali, Nader Nariman-zadeh, 2017, “Sizing and topology optimization of truss structures using genetic programming” https://www.sciencedirect.com/science/article/pii/S2210650217300251

[3] Long C. Nguyen, H. Nguyen-Xuan, 2020, “Deep learning for computational structural optimization” https://www.sciencedirect.com/science/article/pii/S0019057820301415

Exhibit A: Structural optimization of a simple truss

We illustrate the problem with a simple case: a two-element truss with a single unrestricted node where a vertical force is applied.

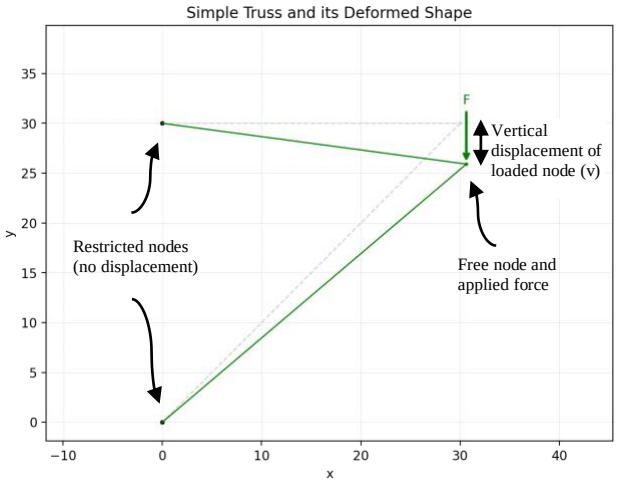

Figure 7: A simple truss in its deformed shape due to a load.

Given that there is a single unrestricted node, there are only two free degrees of freedom: the horizontal and vertical displacements of such node, which we call 'u' and 'v' respectively. Free degrees of freedom are those that can have displacements because are not restricted.

With this geometry, we have the following 2x2 stiffness matrix:

$$K = \frac{E}{L} \cdot \begin{bmatrix} A_1 + \frac{A_2}{2\sqrt{2}} & \frac{A_2}{2\sqrt{2}} \\ \frac{A_2}{2\sqrt{2}} & \frac{A_2}{2\sqrt{2}} \end{bmatrix}$$

We then solve the linear system that connects displacements to applied forces and obtain the scalars 'u' and 'v'.

$$\begin{bmatrix} u \\ v \end{bmatrix} = K^{-1} \cdot \begin{bmatrix} 0 \\ -F \end{bmatrix} = \frac{L \cdot F}{E} \cdot \begin{bmatrix} \frac{1}{A_1} \\ \frac{-\left(2\sqrt{2}A_1 + A_2\right)}{A_1 \cdot A_2} \end{bmatrix}$$

The stiffness of the overall system (S) is the relationship between the magnitude of the applied force (F) and the vertical displacement of the unrestricted node ('v'):

$$S = \left|\frac{F}{v}\right| = \frac{E \cdot A_1 \cdot A_2}{L \cdot \left(2\sqrt{2}A_1 + A_2\right)}$$

And finally, the performance of the system (P) is the relationship between the stiffness and the total volume of material used in the bars. The idea behind this definition of performance is that we could easily achieve a high stiffness by using thick bars in all elements, but this would, of course be very inefficient and expensive. The total volume of material is the sum over all elements of the product between their length and their cross-section area. In this case:

$$P(A_1, A_2) = \frac{S}{V} = \frac{E}{L^2} \cdot \frac{A_1 \cdot A_2}{\left(2\sqrt{2}A_1 + A_2\right) \cdot \left(A_1 + \sqrt{2}A_2\right)}$$

If we define ξ as the ratio between A_2 and A_1, we can rewrite and analytically obtain the optimal performance (P).

$$P(\xi) = \frac{E}{L^2} \cdot \frac{\xi}{\left(\sqrt{2}\xi^2 + 5\xi + 2\sqrt{2}\right)}$$

The solution in this case is $\xi = \sqrt{2}$, which means A_2 should be larger than A_1 by a factor of $\sqrt{2}$. We have shown analytically that this distribution will maximize the performance of the system.

Exhibit B: Structural commentary of the results

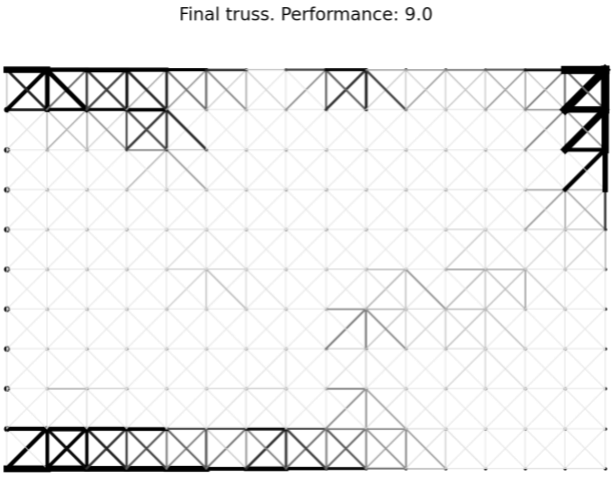

Figure 8: Distribution of cross-section areas in the optimized truss.

As it can be seen in the highest performance truss shown, the distribution of volume makes sense from a structural perspective. The algorithm chooses to reinforce:

a. The higher left part of the structure, which is subject to a high level of tension due to the bending moment produced by the load.
b. The lower left part of the structure, which is subject to a high level of compression because of the same reason as (a).
c. The higher right part of the structure, which is receiving the load and therefore it concentrates a high level of stress, not able to transmit such stress to other elements yet.
d. The center right part, where, absent strong reinforcements in the top and bottom, more volume is necessary to transmit the shear produced by the load.

On the other hand, the algorithm chooses to weaken other parts:

e. The lower right part of the structure, since it is not receiving nor transmitting a significant amount of stress.
f. The center left part of the structure, because it does not get compression nor tension, and the bulk of the shear is being transmitted by the reinforced top and bottom parts.
g. All the elements in the extreme left, since, given that they connect two restricted nodes each, cannot suffer any deformations and therefore cannot transmit load. In other words, it would be a waste of material to place elements there.