MCEN 5125: Optimal Design

Optimal Truss Topology

Jacob Haimes

1 Introduction

For a vast majority of history, the question "what is the best geometry with which to build a structure or object?" has been answered with human intuition. Over approximately the last 150 years [1] an alternative method has been developed called topology optimization. This process allows us to know the optimal layout of material for a specific loading case and boundary conditions. Although many structures and objects have multiple relevant loading conditions, topology optimization can be used to inform our designs and make them far more efficient than our human intuition can.

One method of topology optimization is linear programming. This method optimizes a linear function subject to any number of both equality and inequality constraints. The general form for a linear program (LP) is described in detail in §3.2. In this report, we will explore a simplified version of a truss topology optimization problem, and determine a solution using an LP. The problem is defined in §2, while the formulation of our linear program is discussed in §3. Our solutions generated using said LP are then presented in §4, with closing remarks following in §5.

2 Problem Definition

For this problem, our overarching goal is to design an optimal truss (for a given loading case) within the design constraints provided using linear programming. Specifically, the nodes of this truss must exist only at integer coordinates on an 11×20 unit grid. Each of these coordinates can be thought of as an index in a two-dimensional matrix. Using the same conventions as MATLAB, the node in the upper right corner of the grid is at position $(1, 1)$, and the node in the lower right corner is at $(11, 20)$. This results in a total of $n = 220$ nodes. We choose to designate each node $N^\natural_{r,c}$ by its two dimensional index in the aforementioned matrix. Linear indexing can then be used to create a one dimensional vector $N_{(220\times1)}$ that is analogous to $N^\natural_{(11\times20)}$

Functions for transforming between a two dimensional array $P^\natural_{(U\times V)}$ and a one dimensional array $P_{(U*V\times1)}$ can be seen in Eq. (1) and Eq. (2). The MATLAB functions `ind2sub` and `sub2ind` will also accomplish this task.

$$\begin{aligned} &H : P^\natural_{u,v} \to P_w \\ &\text{where} \;\; H(u,v) = w = U(v-1) + u \end{aligned} \tag{1}$$

$$\begin{aligned} &H^{-1} : P_w \to P^\natural_{u,v} \\ &\text{where} \;\; H^{-1}(w) = (u,v) = \left((w-1 \pmod U) + 1, \quad \left\lfloor \frac{w-1}{U} \right\rfloor + 1 \right) \end{aligned} \tag{2}$$

Our truss is made up of beams that connect two nodes in our design grid. The beam connecting nodes N_α and N_β is designated beam $B^\natural_{\alpha,\beta}$. A $B^\natural$ matrix that contains all of the possible beams in this problem would then have dimensions 220×220, but only elements above the diagonal would be necessary to describe all beams, as $B^\natural_{\alpha,\beta} = B^\natural_{\beta,\alpha}$, and beams must connect two different nodes. An example of such a $B^\natural$ matrix is seen in Eq. (3).

$$B^\natural = \begin{bmatrix} 0 & B^\natural_{1,2} & B^\natural_{1,3} & \cdots & B^\natural_{1,220} \\ & 0 & B^\natural_{2,3} & \cdots & B^\natural_{2,220} \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & B^\natural_{119,220} \\ 0 & & & & 0 \end{bmatrix} \tag{3}$$

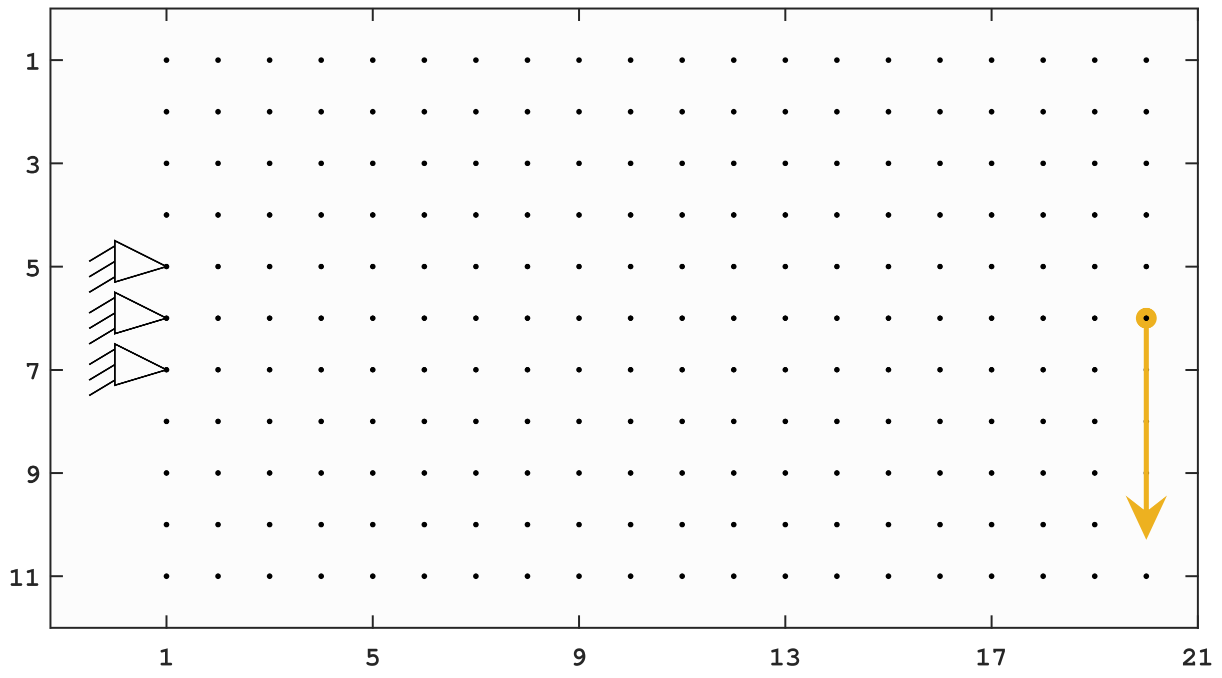


Figure 1: A visualization of the design space that we are working in, including the nodes that are supported, and the force that the truss will be subjected to.

The total number of possible beams is then

$$m = \sum_{i=1}^{n-1} i = \frac{219(220)}{2} = 24\,090.$$

We will also specify that all beams will have uniform cross sectional area $A_{\text{cross}} = 1$, and will be composed of a material that has a yield strength $\sigma_y = 8$ in both tension an compression.

Within the grid there are three nodes that are anchored at positions $(5, 1)$, $(6, 1)$, and $(7, 1)$. The truss will be subjected to a force of $F = 4$ directed vertically downward at node $N^{\natural}_{6,20}$. A visualization of this design grid can be seen in Fig. (1). To qualify as a solution, the truss must be attached to $N^{\natural}_{6,20}$, and be able to maintain static equilibrium without breaking when the force is applied. Any change in strain within the beams is considered negligible, and thus will be ignored.

The two ways in which our truss will be optimal can be thought of as *minimum weight* and *minimum weight considering feasibility/cost.* For clarity, we designate these as problems $\textcircled{1}$ and $\textcircled{2}$, respectively. The difference between these two trusses will be that we penalize longer beams in $\textcircled{2}$. The optimization equations that we will be using are developed in §3.4.3.

3 Formulation

3.1 Efficient Beam Representation

At this point, we have defined almost all relevant variables as some constant value. The only remaining choices that we have are which of the $m = 24\,090$ beams we will use in our final design. Because the elements of our $B^{\natural}$ matrix that are on or below the diagonal are always zero, we can use a modified form of linear indexing to represent $B^{\natural}$ (or any property that individual beams have) as a much smaller one dimensional vector without losing any information. Eq. (4) and Eq. (5) map between the upper triangular two dimensional array $P^{\natural}_{(U\times U)}$ and the one dimensional array $P_{(\ell\times 1)}$, where $\ell = \frac{1}{2}U(U-1)$. Note that in Eq. (5) the expression used to find u utilizes the value obtained for v.

$$H_{\text{ut}} : P^{\natural}_{u,v} \to P_w$$

$$\text{where } \; H_{\text{ut}}(u,v) = w = \frac{(v-1)(v-2)}{2} + u \tag{4}$$

$$H^{-1}_{\text{ut}} : P_w \to P^{\natural}_{u,v}$$

$$\text{where } \; H^{-1}_{\text{ut}}(w) = (u,v) = \left(w - \tfrac{1}{2}(v-1)(v-2), \quad \left\lceil \tfrac{1}{2}\left(1+\sqrt{8w+1}\right)\right\rceil \right) \tag{5}$$

With this mapping established, a beam connecting nodes N_α and N_β can be expressed in two equivalent ways: $B^{\natural}_{\alpha,\beta}$ and B_j, where $j = H_{\text{ut}}(\alpha,\beta)$.

3.2 General Form for Linear Programming

Because we wish to solve our problem through linear programming, we first define what the general form for an LP:

$$\textbf{minimize} \quad \hat{J}^T\hat{z} \tag{6}$$

$$\textbf{subject to} \quad \hat{A}\hat{z} \le \hat{b} \tag{7}$$

$$\hat{A}_{\text{eq}}\hat{z} = \hat{b}_{\text{eq}} \tag{8}$$

where $\hat{z} \in \mathbb{R}^\eta$ is a column vector of the η variables, $\hat{J} \in \mathbb{R}^\eta$ contains the coefficients of the cost function that we are trying to minimize, $\hat{A} \in \mathbb{R}^{\mu\times\eta}$ is the μ constraint inequalities with $\hat{b} \in \mathbb{R}^\mu$ representing the corresponding inequality constants, and $\hat{A}_{\text{eq}} \in \mathbb{R}^{\rho\times\eta}$ is the ρ constraint equations with $\hat{b}_{\text{eq}} \in \mathbb{R}^\rho$ representing the corresponding inequality constants. Often, Eq. (6), Eq. (7), and Eq. (8) are called the minimization function, inequality constraints, and equality constraints, respectively.

When we look at this problem, we quickly recognize that it is *sufficiently large* such that it would be practically impossible to solve by hand. There are, however, many well documented ways that LP problems can be solved using the assistance of a computer, once they are written in this general form. Consequently, our goal will be to identify a pattern that characterizes the general form LP, use that to build the general form for our specific large problem, and then use MATLAB's `linprog` function to find its solution.

3.3 Building Tables of Known Constants

Prior to writing our problem in general form, we will create look-up tables for values that will be used when formulating our problem. The vector $l_{(m\times 1)}$ contains the length of each beam, while the vectors $C_{(m\times 1)}$ and $S_{(m\times 1)}$ contain the cosine and sine of the angle between the horizontal at node N_α and the beam $B^{\natural}_{\alpha,\beta}\, \forall \{\alpha,\beta \mid \alpha < \beta\}$ (upper right triangle of the $B^{\natural}$ matrix).

3.3.1 Beam Length

If our goal is to calculate the length of beam B_j, we must first determine the indices of the two nodes N_α, and N_β that the beam spans between. This is achieved through the use of Eq. (5).

$$(\alpha,\beta) = H^{-1}_{\text{ut}}(j)$$

We then determine the locations of both nodes N_α and N_β in our grid using Eq. (2). Because we will be using these row-column positions at least twice, we make sure to save the vectors $r(N)$ and $c(N)$. The distance formula can then be used to find l_j, which is given in Eq. (9).

$$\left(r_\alpha, c_\alpha\right) = H^{\text{-}1}(N_\alpha)$$
$$\left(r_\beta, c_\beta\right) = H^{\text{-}1}(N_\beta)$$

$$l_j = \sqrt{\left(c_\beta - c_\alpha\right)^2 + \left(r_\beta - r_\alpha\right)^2} \tag{9}$$

3.3.2 Angle Between Nodes

The most intuitive way to store cosine and sine information for our problem would be to create both $C^\natural_{(n\times n)}$ and $S^\natural_{(n\times n)}$, where the row index corresponds to the starting node N_α, and the column index corresponds to terminal node N_β, similar to how we constructed the two-dimensional beam matrix $B^\natural$. Prior to populating the table, we actually already know a couple things about the entries due to the geometry of the design space and the ordering that we chose for our nodes. First, we know that all elements on the diagonal of both $C^\natural$ and $S^\natural$ will be irrelevant, as these entries correspond to cases where $\alpha = \beta$, but a beam $B^\natural_{\alpha,\beta}$ must connect two *unique* nodes. As a result, we will set the diagonals to zero. Additionally, we know that elements in the upper right triangle of $C^\natural$ will be in the range $[0, 1]$, and elements in the same region of $S^\natural$ will be in the range $[-1, 1]$. Although C and S do not contain entries corresponding to the angle between the horizontal at node N_β to the beam $B^\natural_{\alpha,\beta}$, these values will be equivalent to the negative of the corresponding element in the upper right portion of the matrix. Mathematically this is written

$$\aleph^\natural_{\alpha,\beta} = \text{-}\aleph^\natural_{\beta,\alpha}, \tag{10}$$

where $\aleph^\natural$ is either $C^\natural$ or $S^\natural$. This realization allows us to store our cosine and sine information in a $m \times 1$ vector, instead of a two dimensional matrix, if we follow the rule stated in Eq. (11).

$$\aleph^\natural_{\alpha,\beta} = \begin{cases} 0, & \alpha = \beta \\ \aleph_{H_{\text{ut}}(\alpha,\beta)}, & \alpha < \beta \\ \text{-}\aleph_{H_{\text{ut}}(\alpha,\beta)}, & \alpha > \beta \end{cases} \tag{11}$$

Both $C_{(m\times 1)}$ and $S_{(m\times 1)}$ are then populated using Eq. (13) and Eq. (14), respectively.

$$\theta_{\alpha,\beta} = \tan\left(\frac{r_\beta - r_\alpha}{c_\beta - c_\alpha}\right) \tag{12}$$
$$C_j = \arccos\left(\theta_{\alpha,\beta}\right) \tag{13}$$
$$S_j = \arcsin\left(\theta_{\alpha,\beta}\right) \tag{14}$$

3.4 A Smaller Problem in General Form

Due to the size of this problem, we will use a smaller version initially. A diagram of this smaller design grid can be seen in Fig. (3). In this problem, we have $n = 6$ nodes and $m = 15$ possible beams. Additionally, the anchored nodes are $N^\natural_{1,1} = N_1$ and $N^\natural_{2,1} = N_2$, and the node subjected to the force is node $N^\natural_{2,3} = N_6$. We have "constructed" our l, C, and S look up tables, which are seen in Eq. (15).

$$l = \begin{bmatrix} l_1 \\ l_2 \\ \vdots \\ l_{15} \end{bmatrix}, \qquad C = \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_{15} \end{bmatrix}, \qquad S = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{15} \end{bmatrix} \tag{15}$$

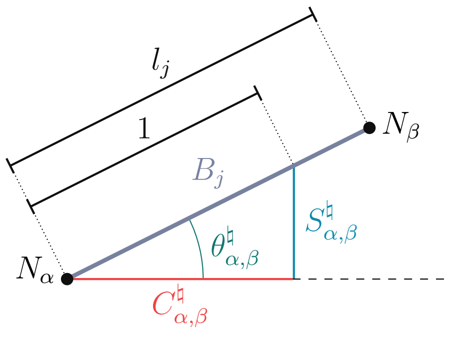

l_j
1
N_β
B_j
$S^\natural_{\alpha,\beta}$
$\theta^\natural_{\alpha,\beta}$
N_α
$C^\natural_{\alpha,\beta}$

Figure 2: Diagram portraying $\theta^\natural_{\alpha,\beta}$, which is described as the angle between the horizontal at node N_α and beam B_j, which connects N_α to N_β. The cosine and sine of $\theta_{\alpha,\beta}$ are also visualized with $C^\natural_{\alpha,\beta}$ and $S^\natural_{\alpha,\beta}$, respectively.

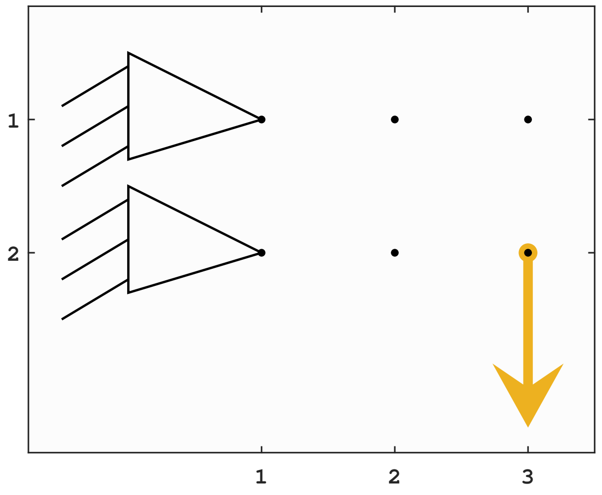


Figure 3: A visualization of the smaller design space, including the nodes that are supported, and the force that the truss will be subjected to.

3.4.1 "Hold the Load" Constraint

To guarantee that this truss is capable of holding the load applied, we want to mandate that the truss be in static equilibrium. To satisfy this, we know that the sum of forces at each non-anchored node must be equal to zero. Because we will be dealing with equalities, we know that we are currently working on building Eq. (8).

Although we could immediately ignore the anchored nodes, making our problem slightly smaller, we choose to include them for now as they may help us identify a pattern. Later, we can choose to either remove the rows that correspond to our anchored nodes, or we can add columns for our reaction forces. We begin by defining a vector $f_{(m\times 1)}$, which will contain the force information for each beam. Recall that $f_{(m\times 1)}$ has an equivalent two dimensional matrix $f^\natural_{(n\times n)}$.

Now we write out the force equilibrium equation at each node in order to identify a pattern in Eq. (17).

To reduce clutter, we also define $\Phi_{\alpha,\beta}$ in Eq. (16).

$$\Phi_{\alpha,\beta} = \begin{bmatrix} C^{\natural}_{\alpha,\beta} \\ S^{\natural}_{\alpha,\beta} \end{bmatrix} \tag{16}$$

$$\begin{array}{llllllll}
 & f^{\natural}_{1,2}\Phi_{1,2} & + f^{\natural}_{1,3}\Phi_{1,3} & + f^{\natural}_{1,4}\Phi_{1,4} & + f^{\natural}_{1,5}\Phi_{1,5} & + f^{\natural}_{1,6}\Phi_{1,6} & (\ +R_1\ ) & = \mathbb{0}_{(2\times1)} \\
f^{\natural}_{2,1}\Phi_{2,1} & & + f^{\natural}_{2,3}\Phi_{2,3} & + f^{\natural}_{2,4}\Phi_{2,4} & + f^{\natural}_{2,5}\Phi_{2,5} & + f^{\natural}_{2,6}\Phi_{2,6} & (\ +R_2\ ) & = \mathbb{0}_{(2\times1)} \\
f^{\natural}_{3,1}\Phi_{3,1} & + f^{\natural}_{3,2}\Phi_{3,2} & & + f^{\natural}_{3,4}\Phi_{3,4} & + f^{\natural}_{3,5}\Phi_{3,5} & + f^{\natural}_{3,6}\Phi_{3,6} & & = \mathbb{0}_{(2\times1)} \\
f^{\natural}_{4,1}\Phi_{4,1} & + f^{\natural}_{4,2}\Phi_{4,2} & + f^{\natural}_{4,3}\Phi_{4,3} & & + f^{\natural}_{4,5}\Phi_{4,5} & + f^{\natural}_{4,6}\Phi_{4,6} & & = \mathbb{0}_{(2\times1)} \\
f^{\natural}_{5,1}\Phi_{5,1} & + f^{\natural}_{5,2}\Phi_{5,2} & + f^{\natural}_{5,3}\Phi_{5,3} & + f^{\natural}_{5,4}\Phi_{5,4} & & + f^{\natural}_{5,6}\Phi_{5,6} & & = \mathbb{0}_{(2\times1)} \\
f^{\natural}_{6,1}\Phi_{6,1} & + f^{\natural}_{6,2}\Phi_{6,2} & + f^{\natural}_{6,3}\Phi_{6,3} & + f^{\natural}_{6,4}\Phi_{6,4} & + f^{\natural}_{6,5}\Phi_{6,5} & & (\ +F\ ) & = \mathbb{0}_{(2\times1)}
\end{array} \tag{17}$$

If reaction forces R_1 and R_2 are ignored, this looks very similar to our cosine and sine matrices, $C^{\natural}$ and $S^{\natural}$. We also realize that any process we create to determine the force in one dimension will be easily applied to the other. As a result, we will work only in the x dimension for now. The horizontal components of Eq. (17) are now rewritten, meaning that we use C instead of Φ. This time, indices of the one dimensional f are used instead of the indices of $f^{\natural}$, and the reaction forces R_1 and R_2 are ignored, as they won't contribute to pattern identification.

$$\begin{array}{lllllll}
 & f_1C^{\natural}_{1,2} & + f_2C^{\natural}_{1,3} & + f_4C^{\natural}_{1,4} & + f_7C^{\natural}_{1,5} & + f_{11}C^{\natural}_{1,6} & = 0 \\
f_1C^{\natural}_{2,1} & & + f_3C^{\natural}_{2,3} & + f_5C^{\natural}_{2,4} & + f_8C^{\natural}_{2,5} & + f_{12}C^{\natural}_{2,6} & = 0 \\
f_2C^{\natural}_{3,1} & + f_3C^{\natural}_{3,2} & & + f_6C^{\natural}_{3,4} & + f_9C^{\natural}_{3,5} & + f_{13}C^{\natural}_{3,6} & = 0 \\
f_4C^{\natural}_{4,1} & + f_5C^{\natural}_{4,2} & + f_6C^{\natural}_{4,3} & & + f_{10}C^{\natural}_{4,5} & + f_{14}C^{\natural}_{4,6} & = 0 \\
f_7C^{\natural}_{5,1} & + f_8C^{\natural}_{5,2} & + f_9C^{\natural}_{5,3} & + f_{10}C^{\natural}_{5,4} & & + f_{15}C^{\natural}_{5,6} & = 0 \\
f_{11}C^{\natural}_{6,1} & + f_{12}C^{\natural}_{6,2} & + f_{13}C^{\natural}_{6,3} & + f_{14}C^{\natural}_{6,4} & + f_{15}C^{\natural}_{6,5} & & = \text{-}F_x
\end{array} \tag{18}$$

We know that f is of the form

$$f = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{15} \end{bmatrix}, \tag{19}$$

meaning that we can replicate the left hand side of Eq. (18) by choosing a vector a_i^T strategically, and using the operation $a_i^T x$ for each row i.

$$\begin{array}{l}
a_1^T = \begin{bmatrix} C^{\natural}_{1,2} & C^{\natural}_{1,3} & 0 & C^{\natural}_{1,4} & 0 & 0 & C^{\natural}_{1,5} & 0 & 0 & 0 & C^{\natural}_{1,6} & 0 & 0 & 0 & 0 \end{bmatrix} \\
a_2^T = \begin{bmatrix} C^{\natural}_{2,1} & 0 & C^{\natural}_{2,3} & 0 & C^{\natural}_{2,4} & 0 & 0 & C^{\natural}_{2,5} & 0 & 0 & 0 & C^{\natural}_{2,6} & 0 & 0 & 0 \end{bmatrix} \\
a_3^T = \begin{bmatrix} 0 & C^{\natural}_{3,1} & C^{\natural}_{3,2} & 0 & 0 & C^{\natural}_{3,4} & 0 & 0 & C^{\natural}_{3,5} & 0 & 0 & 0 & C^{\natural}_{3,6} & 0 & 0 \end{bmatrix} \\
a_4^T = \begin{bmatrix} 0 & 0 & 0 & C^{\natural}_{4,1} & C^{\natural}_{4,2} & C^{\natural}_{4,3} & 0 & 0 & 0 & C^{\natural}_{4,5} & 0 & 0 & 0 & C^{\natural}_{4,6} & 0 \end{bmatrix} \\
a_5^T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & C^{\natural}_{5,1} & C^{\natural}_{5,2} & C^{\natural}_{5,3} & C^{\natural}_{5,4} & 0 & 0 & 0 & 0 & C^{\natural}_{5,6} \end{bmatrix} \\
a_6^T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & C^{\natural}_{6,1} & C^{\natural}_{6,2} & C^{\natural}_{6,3} & C^{\natural}_{6,4} & C^{\natural}_{6,5} \end{bmatrix}
\end{array} \tag{20}$$

We begin to see that there are two patterns that can be combined to create the A_{eq} matrix that we will need for our LP problem. We first look at the pattern of diagonals that are increasing in length. This pattern reminds us of the `diag` function, which takes the elements of a vector of length k and inserts them as the diagonal in a square $k \times k$ matrix. Using this function, the upper right portion of the a_i^T stack, which we denote $A_{C\text{up}}$, is seen in Eq. (21).

$$A_{C\text{up}(6\times15)} = \begin{bmatrix} \texttt{diag}(C_1) & \texttt{diag}(C_{2:3}) & \texttt{diag}(C_{4:6}) & \texttt{diag}(C_{7:10}) & \texttt{diag}(C_{11:15}) \\ \mathbb{0}_{5\times1} & \mathbb{0}_{4\times2} & \mathbb{0}_{3\times3} & \mathbb{0}_{2\times4} & \mathbb{0}_{1\times5} \end{bmatrix} \tag{21}$$

The lower left portion is denoted $A_{C\text{lo}}$. Notice that in Eq. (20), the $C^\natural$ indices in this are explicitly *not* in the corresponding one dimensional C vector, as this row index α is greater than the column index β. We take advantage of Eq. (10) to construct Eq. (22).

$$A_{C\text{lo}(6\times15)} = \begin{bmatrix} & & & \mathbb{0}_{(1\times15)} & & \\ \text{-}C_1 & & & & \mathbb{0}_{(1\times14)} & \\ & \text{-}C_{2:3} & & & \mathbb{0}_{(1\times12)} & \\ \mathbb{0}_{(4\times1)} & & \text{-}C_{4:6} & & & \mathbb{0}_{(1\times9)} \\ & \mathbb{0}_{(3\times2)} & & \text{-}C_{7:10} & & \mathbb{0}_{(1\times5)} \\ & & \mathbb{0}_{(2\times3)} & \mathbb{0}_{(1\times4)} & & \text{-}C_{11:15} \end{bmatrix} \tag{22}$$

The coefficients in our equality constraints, A_{eq}, are then expressed as the sum of these two matrices, while the constants are simply the right hand side of Eq. (18). The variables present in the constraint equations are in z. We do not yet designate $\hat{A}_{\text{eq}}$, $\hat{b}_{\text{eq}}$, or $\hat{z}$ though, as we may need to pad our arrays in order to maintain compatibility with our cost function.

Before we write our equality constraints in general form, we recall that we have ignored our reaction forces R_1 and R_2. We recognize that we can add four more columns, two of which will correspond to the horizontal components of the reaction forces, while the others correspond to the vertical components. This also requires that we add four more rows to our variable vector. These rows and columns can be seen in Eq. (23). The remaining piece of given information, F, is included in the constants vector b_{eq}.

$$z_{R(4\times1)} = \begin{bmatrix} R_{1,x} \\ R_{2,x} \\ R_{1,y} \\ R_{2,y} \end{bmatrix}, \qquad A_{R(12\times4)} = \begin{bmatrix} 1 & 0 & \mathbb{0}_{(6\times2)} & \\ 0 & 1 & & \\ & & 1 & 0 \\ \mathbb{0}_{(10\times2)} & & 0 & 1 \\ & & \mathbb{0}_{(4\times2)} & \end{bmatrix} \tag{23}$$

$$(1)R_{1,x} + (0)R_{2,x} + (2)R_{1,y} + (2)R_{2,y} = 0 \tag{24}$$

Finally, we concatenate our matrices appropriately to arrive at Eq. (25).

$$z_{\text{eq}(19\times1)} = \begin{bmatrix} f \\ z_R \end{bmatrix}, \qquad A_{\text{eq}(12\times19)} = \begin{bmatrix} A_{C\text{up}} + A_{C\text{lo}} & A_R \\ A_{S\text{up}} + A_{S\text{lo}} & \end{bmatrix}, \qquad b_{\text{eq}(12\times1)} = \begin{bmatrix} \mathbb{0}_{(5\times1)} \\ \text{-}F_x \\ \mathbb{0}_{(5\times1)} \\ \text{-}F_y \end{bmatrix} \tag{25}$$

3.4.2 "Don't Break" Constraint

Before constructing these constraints, we hark back to our days in a thoroughly riveting lecture regarding material properties that we all assuredly had. Among many other things, we learned that the stress in an object is given by

$$\sigma = \frac{\vec{F}}{A_{\text{cross}}}, \tag{26}$$

and that the maximum stress that a material can take before permanently changing its shape is yield strength σ_y. The "don't break" constraint in this problem means that the stress $|\sigma_j| \leq 8$ in any beam B_j. For this problem, we also assume that strain is negligible, that the beam material is *even*[1], and that this material exhibits no *elastic deformation*[2].

Because the cross sectional area of our beams was provided as $A_{\text{cross}} = 1$, we see that $\sigma = f$. When f is then substituted into our inequality we get

$$|f| \leq 8 * \mathbb{1}_{(15\times1)} \implies \begin{cases} f \leq 8 * \mathbb{1}_{(15\times1)} \\ \text{-}f \leq 8 * \mathbb{1}_{(15\times1)} \end{cases} \tag{27}$$

Eq. (27) then leads us to the A and b in Eq. (28).

$$A = \begin{bmatrix} \boldsymbol{I}_{15} \\ \text{-}\boldsymbol{I}_{15} \end{bmatrix}, \qquad b = 8 * \mathbb{1}_{(30\times1)} \tag{28}$$

3.4.3 Minimization Function

The next piece in our problem formulation puzzle is determining the components of the cost function, J^T and z. We cannot lose sight of our goals during this process, of which we have two. The first is to obtain a *minimum weight* solution to this problem. Because all of our beams have the same A_{cross} and are made of the same material, ① is analogous to *minimum length.* Our distances are predefined, suggesting that we can't minimize the length of the beams. Although we can't change individual beam lengths, we can minimize the one-norm of the internal beam forces, meaning that our first minimization function will be

$$J_1^T z_1 = \|f\|_1. \tag{29}$$

The idea here is that as more beams are used in the design, this value will increase. At the same time, long unsupported beams will be difficult to manufacture, ship, and work with, so it would also make sense to create a minimization function that addresses this concern. Our solution to ② will be to minimize the length weighted sum of the internal beam forces, leading to our second minimization function,

$$J_2^T z_2 = \|l \odot f\|_1 = l \odot \|f\|_1. \tag{30}$$

In Eq. (30), we use the $\odot$ operator to signify element-wise multiplication. With these functions defined, we now attempt to build the general form of both problems without considering their constraints. Let's examine the LP for ①.

$$\begin{aligned} \|f\|_1 &= |f_1| + |f_2| + \cdots + |f_{15}| \\ &= \mathbf{max}\Big\{f_1, \text{-}f_1\Big\} + \mathbf{max}\Big\{f_2, \text{-}f_2\Big\} + \cdots + \mathbf{max}\Big\{f_{15}, \text{-}f_{15}\Big\} \end{aligned} \tag{31}$$

[1] A material that is *even* has a compressive yield strength that is equivalent to its tensile yield strength

[2] *Elastic deformation* occurs when an object changes shape due to a load, and it would return to its original shape if that load were removed.

This pattern can be succinctly expressed with an *auxiliary variable*:

$$t_j = \mathbf{max}\left\{ f_j, \text{-}f_j \right\}. \tag{32}$$

Furthermore, the entire minimization function can be written

$$\sum_{j=1}^{15} t_j. \tag{33}$$

The minimization function can now be expressed in general form, where

$$J_1 = \begin{bmatrix} \mathbb{0}_{(15\times1)} \\ \mathbb{1}_{(15\times1)} \end{bmatrix}, \quad z_1 = \begin{bmatrix} f_{(15\times1)} \\ t_{(15\times1)} \end{bmatrix}, \quad A_1 = \begin{bmatrix} \boldsymbol{I}_{15} & \text{-}\boldsymbol{I}_{15} \\ \text{-}\boldsymbol{I}_{15} & \text{-}\boldsymbol{I}_{15} \end{bmatrix}, \quad \text{and} \quad b_1 = \mathbb{0}_{(30\times1)}. \tag{34}$$

LP ② uses the same *auxiliary variable*, but we modify J slightly so that our cost function looks like Eq. (35).

$$l\|f\|_1 = l_1|f_1| + l_2|f_2| + \cdots + l_{15}|f_{15}| \tag{35}$$

We then can write this LP in general form (ignoring other constraints), where

$$J_2 = \begin{bmatrix} \mathbb{0}_{(15\times1)} \\ l_{(15\times1)} \end{bmatrix}, \quad z_2 = \begin{bmatrix} f_{(15\times1)} \\ t_{(15\times1)} \end{bmatrix}, \quad A_2 = \begin{bmatrix} \boldsymbol{I}_{15} & \text{-}\boldsymbol{I}_{15} \\ \text{-}\boldsymbol{I}_{15} & \text{-}\boldsymbol{I}_{15} \end{bmatrix}, \quad \text{and} \quad b_2 = \mathbb{0}_{(30\times1)}. \tag{36}$$

3.4.4 Unifying the Sub-Problems

We now have a general form representation of both constraints and the minimization function, which can be seen in Eqs. (25), (28), (34), and (36). Our next step, then, is to synthesize these separate equations into our final general form LP problem.

$$\begin{aligned}
\hat{z} &= \begin{bmatrix} f_{(15\times1)} \\ t_{(15\times1)} \\ z_{R(4\times1)} \end{bmatrix}, \qquad \hat{J}_1 = \begin{bmatrix} \mathbb{0}_{(15\times1)} \\ \mathbb{1}_{(15\times1)} \\ \mathbb{0}_{(4\times1)} \end{bmatrix}, \qquad \hat{J}_2 = \begin{bmatrix} \mathbb{0}_{(15\times1)} \\ l_{(15\times1)} \\ \mathbb{0}_{(4\times1)} \end{bmatrix}, \\
\hat{A} &= \begin{bmatrix} \boldsymbol{I}_{15} & \text{-}\boldsymbol{I}_{15} & \\ \text{-}\boldsymbol{I}_{15} & \text{-}\boldsymbol{I}_{15} & \mathbb{0}_{(60\times4)} \\ \boldsymbol{I}_{15} & \mathbb{0}_{(30\times15)} & \\ \text{-}\boldsymbol{I}_{15} & & \end{bmatrix}, \qquad \hat{b} = \begin{bmatrix} \mathbb{0}_{(30\times1)} \\ 8 * \mathbb{1}_{(30\times1)} \end{bmatrix}, \\
\hat{A}_{\text{eq}} &= \begin{bmatrix} A_{C\text{up}} + A_{C\text{lo}} & \mathbb{0}_{(12\times15)} & A_R \\ A_{S\text{up}} + A_{S\text{lo}} & & \end{bmatrix}, \qquad \hat{b}_{\text{eq}} = \begin{bmatrix} \mathbb{0}_{(5\times1)} \\ \text{-}F_x \\ \mathbb{0}_{(5\times1)} \\ \text{-}F_y \end{bmatrix}
\end{aligned} \tag{37}$$

3.5 Expanding to an Arbitrarily Sized Design Grid

Now that we have formulated this smaller problem, it is much easier to wrap our heads around the larger problem that we have set out to solve. To make our framework usable for any set of loading and anchoring conditions, we define a general form for an arbitrarily sized grid of nodes in which there are no applied forces, and no nodes are anchored. Although this is useless as a standalone finding, we will also define what must be added to this form to create any possible loading and anchoring conditions. Here, n represents the total number of nodes in the grid and m represents the total number of possible beams that can be created in the grid. We first define a few other variables that will help us build the final matrices required for our problem formulation. Note that in Eq. (39) the bottom left corner of the i^{th} diagonal is on row i.

$$\left.\begin{aligned} \xi_i &= \binom{i}{2} + 1 \\ \zeta_i &= \xi - 1 + i \end{aligned}\right\} \text{for row index } i \in [2:n] \tag{38}$$

$$A_{\aleph\text{up}(n\times m)} = \begin{bmatrix} \texttt{diag}(\aleph_1) & & & & & & \\ & \texttt{diag}(\aleph_{2:3}) & & & & & \\ & & \texttt{diag}(\aleph_{4:6}) & & & & \\ & & & \ddots & & & \\ & & & & \texttt{diag}(\aleph_{\xi_i:\zeta_i}) & & \\ & & & & & \ddots & \\ \mathbb{0} & & & & & & \texttt{diag}(\aleph_{m-n+1:m}) \\ & & & \mathbb{0}_{(1\times m)} & & & \end{bmatrix} \tag{39}$$

$$A_{\aleph\text{lo}(n\times m)} = \begin{bmatrix} & & & \mathbb{0}_{(1\times m)} & & & \\ \text{-}\aleph_1 & & & & & & \mathbb{0} \\ & \text{-}\aleph_{2:3} & & & & & \\ & & \text{-}\aleph_{4:6} & & & & \\ & & & \ddots & & & \\ & & & & \text{-}\aleph_{\xi_{i-1}:\zeta_{i-1}} & & \\ & & & & & \ddots & \\ \mathbb{0} & & & & & & \text{-}\aleph_{m-n+1:m} \end{bmatrix} \tag{40}$$

$$z_{\text{gen}} = \begin{bmatrix} f_{(m\times 1)} \\ t_{(m\times 1)} \end{bmatrix}, \qquad J_{1,\text{gen}} = \begin{bmatrix} \mathbb{0}_{(m\times 1)} \\ \mathbb{1}_{(m\times 1)} \end{bmatrix}, \qquad J_{2,\text{gen}} = \begin{bmatrix} \mathbb{0}_{(m\times 1)} \\ l_{(m\times 1)} \end{bmatrix},$$

$$A_{\text{gen}} = \begin{bmatrix} \boldsymbol{I}_m & \text{-}\boldsymbol{I}_m \\ \text{-}\boldsymbol{I}_m & \text{-}\boldsymbol{I}_m \\ \boldsymbol{I}_m & \\ \text{-}\boldsymbol{I}_m & \mathbb{0}_{(2m\times m)} \end{bmatrix}, \qquad b_{\text{gen}} = \begin{bmatrix} \mathbb{0}_{(2m\times 1)} \\ 8 * \mathbb{1}_{(2m\times 1)} \end{bmatrix},$$

$$A_{\text{eq,gen}} = \begin{bmatrix} A_{C\text{up}} + A_{C\text{lo}} & \mathbb{0}_{(2n\times m)} \\ A_{S\text{up}} + A_{S\text{lo}} & \end{bmatrix}, \quad b_{\text{eq,gen}} = \begin{bmatrix} \mathbb{0}_{(2n\times 1)} \end{bmatrix} \tag{41}$$

With the framework established, we now add the loading conditions and anchor node reaction forces. Note that the variable δ represents the number of anchor nodes in the problem.

For each anchor node N_i :

$$z_{\text{gen}} = \begin{bmatrix} z_{\text{gen}} \\ R_{\varphi,x} \\ R_{\varphi,y} \end{bmatrix} \tag{42}$$

$$J_{1,\text{gen}} = \begin{bmatrix} J_{1,\text{gen}} \\ \mathbb{0}_{(2,1)} \end{bmatrix} \tag{43}$$

$$J_{2,\text{gen}} = \begin{bmatrix} J_{2,\text{gen}} \\ \mathbb{0}_{(2,1)} \end{bmatrix} \tag{44}$$

$$A_{\text{gen}} = \begin{bmatrix} A_{\text{gen}} & \mathbb{0}_{(4m\times 2\delta)} \end{bmatrix} \tag{45}$$

$$A_{R_i} = \begin{bmatrix} \mathbb{0}_{1\times i-1} & \\ & \mathbb{0}_{1\times n+i-1} \\ 1 & \\ \mathbb{0}_{1\times 2n-i} & \\ & 1 \\ & \mathbb{0}_{1\times n-i} \end{bmatrix} \tag{46}$$

Once the above process has been completed for all anchor nodes, we create the matrix A_R by concatenating all A_{R_i} horizontally. A_R is then horizontally concatenated with $A_{\text{eq,gen}}$. Finally, for each node that is subjected to a force F_i, we set the value of b_{eq} at index i to equal -$F_{i,x}$, and the value at index $n+i$ to equal -$F_{i,y}$.

4 Solution

The process described in §3 is then implemented in MATLAB, which can be seen in Appendix A. It is worth noting that our final $\hat{A}$, the largest matrix that we use in this problem, will have dimensions of $4m \times (2m + 2\delta) = 96\,360 \times 48\,186$, resulting in more than 4.6×10^9 elements. If we construct these large matrices normally, the process will take a rather long time, and we are likely going to be limited by our computer's memory. Because the large matrices in this problem, including $\hat{A}$, are *sparse*[3], they are more efficiently stored as a *sparse matrix*[4]. Although we have attempted to reduce the size of this problem, the `linprog` function takes approximately 89.8 s to find the optimal solution for $\textcircled{1}$, and approximately 297.4 s to find the optimal solution for $\textcircled{2}$. The code used to set up and execute these LPs can be viewed in Appendix A.

The `linprog` function has now returned our optimal $\hat{z}^*$ vectors, but these data require a small amount of post-processing before we have the solution to our problems. First, $\hat{z}^*$ contains more information than we need, as the reaction forces and auxiliary variables are not part of our cost function. As a result, we define a variable that contains only the force data, $f^* = \hat{z}^*(1 : m)$. There is a high likelihood that $\hat{f}^*$ will contain some entries whose force magnitude is very small. To handle this, we will set elements in f^* that are less than some small value to zero. Here, we have chosen for the threshold to be a force magnitude of .001.

Plots of the design optimal trusses for both optimization problems can be seen in Fig. (4), while a tabulated version of our solution to $\textcircled{1}$ can be seen in Table I. The solution for $\textcircled{2}$ is not tabulated, as

[3]A matrix whose elements are mostly zero.

[4]MATLAB can store any matrix using either the traditional or *sparse* datatype. In *sparse matrices* all entries are zero, except those noted otherwise. Each nonzero element, then, has three pieces of data attached to it: the row index, the column index, and the element value. A product of this representation is that matrices with many more zeros than nonzeros can be stored more efficiently in this form.

$\boldsymbol{B_j}$	$\boldsymbol{N_\alpha}$	(r_α, c_α)	$\boldsymbol{N_\beta}$	(r_β, c_β)	$\boldsymbol{f_j}$
22 371	5	(5, 1)	213	(4, 20)	7.913
22 373	7	(7, 1)	213	(4, 20)	-8.000
22 583	5	(5, 1)	214	(5, 20)	7.956
22 585	7	(7, 1)	214	(5, 20)	-8.000
22 796	5	(5, 1)	215	(6, 20)	8.000
22 798	7	(7, 1)	215	(6, 20)	-8.000
23 004	213	(4, 20)	215	(6, 20)	0.832
23 005	214	(5, 20)	215	(6, 20)	0.838
23 010	5	(5, 1)	216	(7, 20)	8.000
23 012	7	(7, 1)	216	(7, 20)	-7.956
23 220	215	(6, 20)	216	(7, 20)	-0.838
23 225	5	(5, 1)	217	(8, 20)	6.274
23 227	7	(7, 1)	217	(8, 20)	-6.205
23 435	215	(6, 20)	217	(8, 20)	-0.652

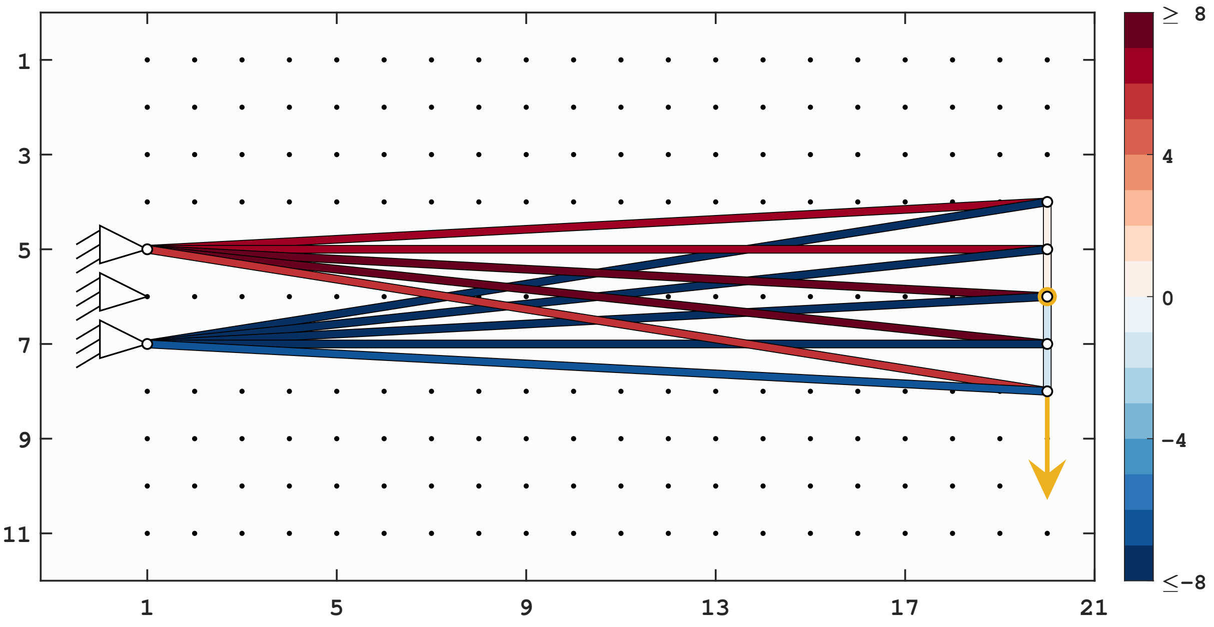
TABLE I: Solution to LP ①. We note that many of the beams have an internal force whose magnitude is equivalent to the maximum allowable force: 8. A visualization of this solution can be seen in Fig. (4).

there are too many beams to gain a meaningful understanding of the truss without visualization.

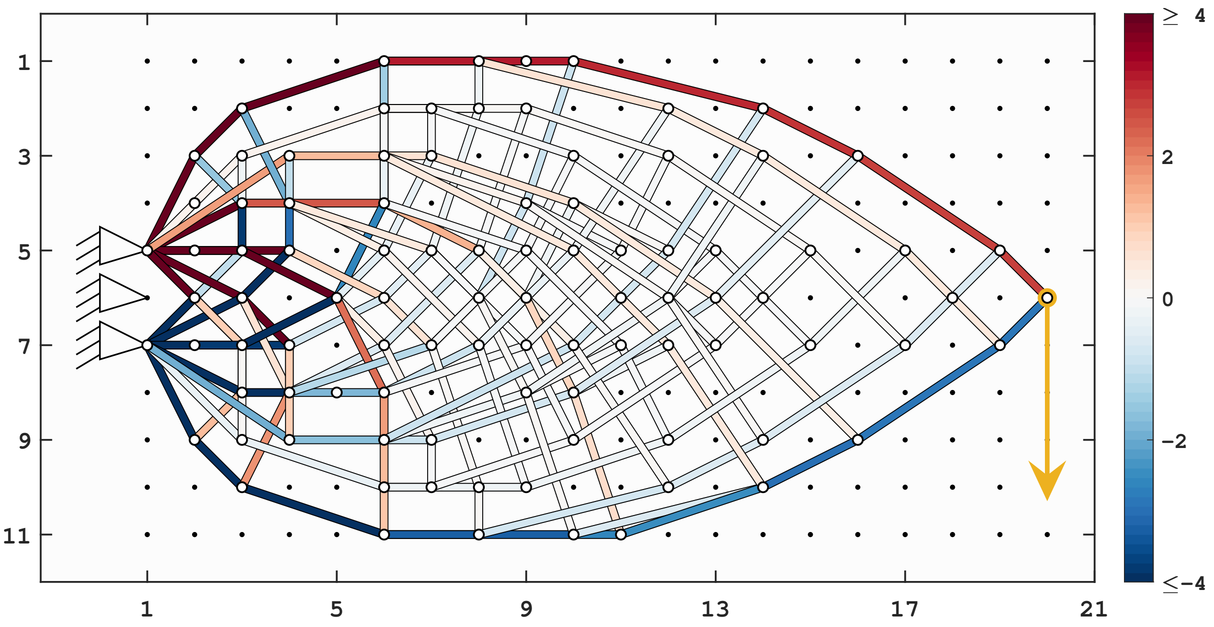
For LP ①, we have used 14 of our 24 090 potential beams, resulting in an optimal cost function value of 79.46 and total beam length of 196.79. In this design, no beams are connected to the anchor node $N^{\natural}_{6,1}$, while the other two anchor nodes are both connected to the same 5 nodes in the column where the force is applied. Although it is difficult to tell from Fig. (4), Table I shows us that the vertical beams are not symmetric about row 6.

In contrast to the relatively simple design created by LP ①, LP ② creates a design that would be rather difficult to intuit. This truss uses 183 beams, and produces an optimal cost function value of 554.46 and has a total beam length of 423.34. Similarly to our other solution, this truss is not connected to anchor node $N^{\natural}_{6,1}$ at all.

When comparing these two solutions, we must remind ourselves that both are optimal in precisely the way our cost function specified. As could be expected, our solution to ① has a much lower total beam length than our solution to ②. Two more interesting comparisons are maximum and average beam lengths. The solution to ① has a maximum beam length of 19.24 and a mean beam length of 14.06, while the solution to ② contains no beam longer than 4.47, and has an average beam length of 2.31. If long beams are hard to come by, the latter solution would have a significant advantage.



(a) Visualization of LP ① solution



(b) Visualization of LP ② solution

Figure 4: Visualizations of the solutions to our LP problems. Beam color is a function of the internal force that the beam is experiencing, shown on the colorbar. Tension is considered to be a force in the positive direction, meaning that beams in tension are a shade of red, while beams in compression are a shade of blue. A plot that is colored with the categorical classification of each beam (tension or compression) can be seen in Appendix B.

Attribute	①	②
Total beam length	196.79	423.34
Cost function value	79.46	554.46
Sum of force magnitudes	79.46	265.61
Number beams used	14	183
Maximum beam length	19.24	4.47
Average beam length	14.06	2.31
Solver time	89.8	297.4

Table II: Comparison of various attributes for the solutions to ① and ②. We note that although ① has a total beam length that is about half of ②, the average beam length used in ② is smaller than one sixth the average beam length of ①.

5 Conclusion

We recall that the goal of this report was to find the optimal truss topology for a set design space. To be considered a viable solution, the truss must support a set applied load without breaking. In more quantitative terms, this requirement can be expressed with two fundamental constraints: the truss must maintain static equilibrium when the force is applied, and the stress in each beam must not exceed the yield stress of the beam material. Furthermore, there were two distinct optimum that we wished to identify: the topologies associated with ① and ②. Finally, we were to arrive at our solution using linear programming.

Using many tools, such as clever grid numbering, matrix linearization, and using a smaller version of the problem we were able to construct a general form algorithm for a problems similar to the one provided. This process was then implemented in Matlab, and solutions for both objectives were found. Plots of our two optimal solutions can be seen in Fig. (4). For both cost functions, there is relative symmetry about the sixth row in the design space. Table II shows a comparison between the two optimal trusses.

One potentially interesting direction to go with this project would be to attempt to construct our own cost function that results in a compromise between solutions ① and ②. Another interesting aspect to explore would be the optimization of the algorithms for speed. I would be interested to explore potential methods that could reduce computation time of a large LP like these.

References

[1] L. János and H. Ismail, “Milestones in the 150-Year History of Topology Optimization: A Review,” Computer Assisted Methods in Engineering and Science, vol. 27, pp. 97–132, Sep. 2020.

A MATLAB Code

A.1

```matlab
%% Problem Specifications

% Set design grid dimensions as (row #, col #)
grid_dim = [11,20];

% Define anchor nodes (row #, col #)
anch = [ 5 1; 6 1; 7 1];

% Define forces
%   Floc = ( row #, col #)
%   Fcom = ( X force, Y force )
Floc = [6,20];
Fcom = [0,-4];

% Define maximum force inside beams (for both tension and compression)
Fmax = 8;

%% Save all known values
%   Both individual numerical values and lookup tables

% Save number of anchor and force nodes
delta = size(anch,1);
Fnum = size(Floc,1);

% Calculate total number of nodes (n) and possible beams (m)
n = grid_dim(1) * grid_dim(2);
m = nchoosek(n,2);

% Calculate beam numberings for the beams connecting anchor nodes
anch_AlBe = nchoosek(sub2ind(grid_dim,anch(:,1),anch(:,2)),2);
anch_B = H_ut(anch_AlBe(:,1),anch_AlBe(:,2));

% Create N
%   Consists of Nr and Nc, holds the (row,column) positions of each node
[Nc,Nr] = meshgrid(1:grid_dim(2),1:grid_dim(1));
N = [ reshape(Nr,n,1), reshape(Nc,n,1) ];

% Determine the nodes (Nalpha,Nbeta) that correspond to each beam Bj
AlBe = H_ut(1:m);

% Create beam length lookup table
L = sqrt(...
            ( N(AlBe(:,2),2) - N(AlBe(:,1),2) ).^2 ... % (cBe - cAl)^2
          + ( N(AlBe(:,2),1) - N(AlBe(:,1),1) ).^2 ... % (rBe - rAl)^2
        );

% Create beam angle lookup table
theta = atan2(...
                ( N(AlBe(:,2),1) - N(AlBe(:,1),1) ), ... % (rBe - rAl)
                ( N(AlBe(:,2),2) - N(AlBe(:,1),2) )  ... % (cBe - cAl)
         );

% Create cosine and sine lookup tables
C = cos(theta);
S = sin(theta);

% Create xi and zeta lookup tables
xi = [ 0; arrayfun(@nchoosek,2:n,2*ones(1,n-1))' + 1 ];
zeta = xi + (1:n)' - 1;
```

```matlab
%% Define constraints and cost function
% Create upper right portion of cosine equality component matrix
ACup = zeros(n,m);
ACup(1,1) = C(1);
for i = 2:n-1
    ACup(1:i,xi(i):zeta(i)) = diag(C(xi(i):zeta(i)));
end
ACup(:,anch_B) = zeros(n,delta);

% Create upper right portion of sine equality component matrix
ASup = zeros(n,m);
ASup(1,1) = S(1);
for i = 2:n-1
    ASup(1:i,xi(i):zeta(i)) = diag(S(xi(i):zeta(i)));
end
ASup(:,anch_B) = zeros(n,delta);

% Create lower left portion of cosine equality component matrix
AClo = zeros(n,m);
AClo(2,1) = -C(1);
for i = 2:n-1
    AClo(i+1,xi(i):zeta(i)) = -C(xi(i):zeta(i));
end
AClo(:,anch_B) = zeros(n,delta);

% Create lower left portion of sine equality component matrix
ASlo = zeros(n,m);
ASlo(2,1) = -S(1);
for i = 2:n-1
    ASlo(i+1,xi(i):zeta(i)) = -S(xi(i):zeta(i));
end
ASlo(:,anch_B) = zeros(n,delta);

% Create reaction force portion of constraints
AR = spalloc(2*n,2*delta,2*delta);
for k = 1:delta
    AR(sub2ind(grid_dim,anch(k,1),anch(k,2)),2*k - 1) = 1;
    AR(n+sub2ind(grid_dim,anch(k,1),anch(k,2)),2*k) = 1;
end

% Create equality constraint matrix and constants vector
%   Add upper and lower components together for both cosine and sine.
%   Concatenate the resulting matrices vertically.
Aeq = sparse([[ ACup + AClo; ASup + ASlo ] zeros(2*n,m), AR]);
beq = spalloc(2*n,1,Fnum);
for k = 1:Fnum
    beq(sub2ind(grid_dim,Floc(k,1),Floc(k,2),1)) = Fcom(k,1);
    beq(n+sub2ind(grid_dim,Floc(k,1),Floc(k,2),1)) = Fcom(k,2);
end

% Create inequality constraints matrix and constants vector
A = [[ speye(m) -speye(m);...
      -speye(m) -speye(m);...
        [ speye(m); -speye(m) ]  spalloc(2*m,m,1) ]...
    spalloc(4*m,2*delta,1) ];
b = [ zeros(2*m,1); Fmax*ones(2*m,1) ];

% Create cost function coefficient vectors
J1 = [ zeros(m,1); ones(m,1); zeros(2*delta,1) ];
J2 = [ zeros(m,1); L; zeros(2*delta,1) ];
```

```
%% Solve LPs and visualize
% Solve minimum weight LP
[z1,J1val] = linprog(J1,A,b,Aeq,beq);

% Save the beam forces
f1 = z1(1:m);
f1_avmag = mean(abs(f1(f1~=0)));

% Create struct with all necessary plotting information
P1.n = n;
P1.file = 'optTruss1.eps';
P1.zLog = logical(z1(1:m));
P1.L = L(P1.zLog);
P1.Lsum = sum(P1.L);
P1.B = z1(P1.zLog);
P1.Buse = AlBe(P1.zLog,:);
P1.Bnum = size(P1.Buse,1);
P1.Aluse = N(P1.Buse(:,1),:);
P1.Beuse = N(P1.Buse(:,2),:);
P1.cuse = [ P1.Aluse(:,2)'; P1.Beuse(:,2)' ];
P1.ruse = [ P1.Aluse(:,1)'; P1.Beuse(:,1)' ];

% Set colormap
bin = 16;
P1.range = [-8,8];
P1.map = brewermap(bin,'*RdBu');
P1.scale = P1.B;
P1.scale( P1.B > P1.range(2) ) = P1.range(2);
P1.scale( P1.B < P1.range(1) ) = P1.range(1);
P1.colorInd = floor((bin-1)*(.5 + P1.scale ./ (P1.range(2)-P1.range(1))))+1;
P1.colorVal = P1.map(P1.colorInd,:);
plotspace(P1);

% Solve minimum weight considering feasibility/cost LP
[z2,J2val] = linprog(J2,A,b,Aeq,beq);

% Save the beam forces
f2 = z2(1:m);
f2_avmag = mean(abs(f2(f2~=0)));

% Create struct with all necessary plotting information
P2.n = n;
P2.file = 'optTruss2.eps';
P2.filter = abs(z2) > 1e-3;
P2.zLog = P2.filter(1:m);
P2.L = L(P2.zLog);
P2.Lsum = sum(P2.L);
P2.B = z2(P2.zLog);
P2.Buse = AlBe(P2.zLog,:);
P2.Bnum = size(P2.Buse,1);
P2.Aluse = N(P2.Buse(:,1),:);
P2.Beuse = N(P2.Buse(:,2),:);
P2.cuse = [ P2.Aluse(:,2)'; P2.Beuse(:,2)' ];
P2.ruse = [ P2.Aluse(:,1)'; P2.Beuse(:,1)' ];

% Set colormap
bin=60;
P2.range = [-4,4];
P2.map = brewermap(bin,'*RdBu');
P2.scale = P2.B;
P2.scale( P2.B > P2.range(2) ) = P2.range(2);
P2.scale( P2.B < P2.range(1) ) = P2.range(1);
P2.colorInd = floor((bin-1)*(.5 + P2.scale ./ (P2.range(2)-P2.range(1))))+1;
P2.colorVal = P2.map(P2.colorInd,:);
plotspace(P2);
```

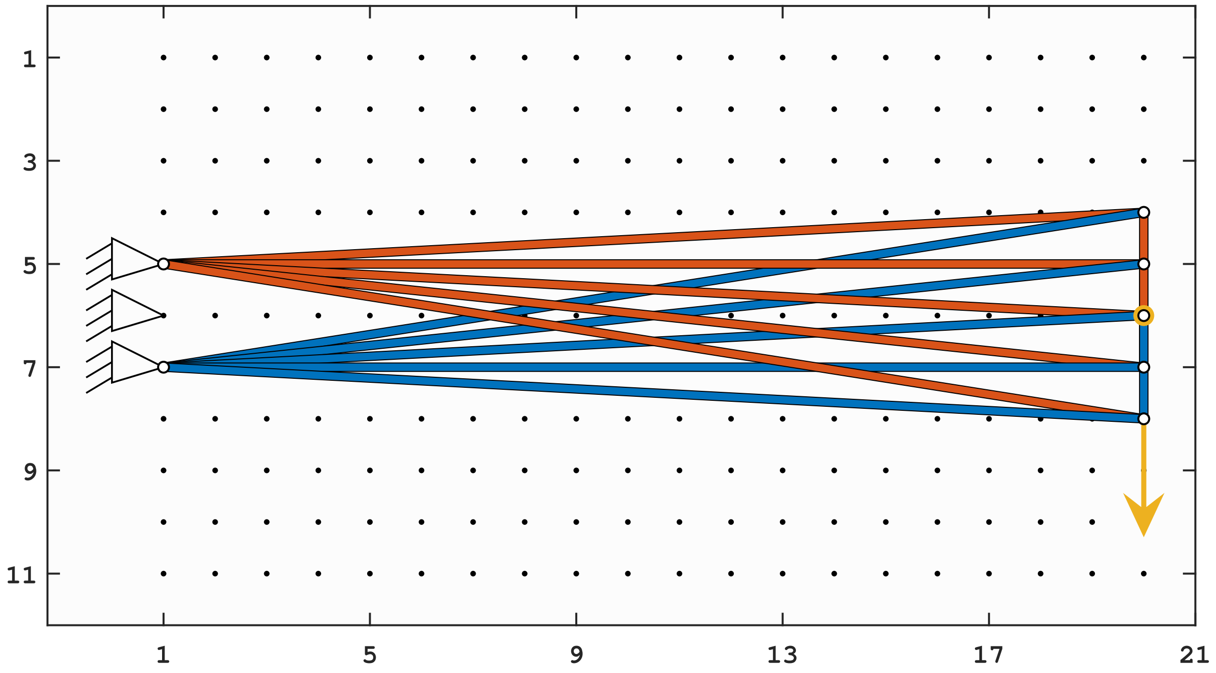
A.2 Plotting Function

```
function handles = plotspace(P)
%PLOTSPACE - Plot design space for optimal truss topology problem

f_loc = [20,6];
sleft_loc = [ 1 5; 1 6; 1 7 ];
nsleft = size(sleft_loc,1);
supX = repmat([ 1 0 0 1 ],nsleft,1);
supY = repmat([ 0 .3 -.5 0 ],nsleft,1) + sleft_loc(:,2);
groundX = fliplr([ -.5  0 ] + zeros(3*nsleft,1));
groundY = [ -.3  0 ] + (repmat([-.1,.2,.5]',nsleft,1) + repelem(sleft_loc(:,2),3));
forceX = [-.03,.03,.03,.35,0,-.35,-.03] + f_loc(1);
forceY = [0,0,3.75,3.5,4.25,3.5,3.75] + f_loc(2);

fig = figure('Name','Optimal Truss Problem');
    set(fig,'Position',[2,200,1075,510])%910
    ax = axes('Parent',fig,'xlim',[-1.25,21],'ylim',[0 12],'yDir','reverse',...
        'DataAspectRatio',[1 1 1],'Color',[.99 .99 .99],'FontName','Monospaced',...
        'FontWeight','Bold','FontSize',13.5,'LineWidth',1);
    box(ax,'on');                   hold(ax,'on');
    [X,Y] = meshgrid(1:20,1:11);
    X = reshape(X,220,1);           Y = reshape(Y,220,1);
    s = scatter(ax,X,Y,8,'k','filled');
        s.DataTipTemplate.DataTipRows(1,:).Label = '\bf i';
        s.DataTipTemplate.DataTipRows(1,:).Value = 'YData';
        s.DataTipTemplate.DataTipRows(2,:).Label = '\bf j';
        s.DataTipTemplate.DataTipRows(2,:).Value = 'XData';
        s.DataTipTemplate.DataTipRows(end+1) = dataTipTextRow('\bf N',1:P.n);
    xticks(ax,1:4:21);
    yticks(ax,1:2:11);
    patch('Parent',ax,'xData',forceX,'yData',forceY,'EdgeColor','#EDB120',...
        'FaceColor','#EDB120','LineWidth',1,'PickableParts','None');
    s2 = scatter(ax,20,6,8,'k','filled');
        s2.DataTipTemplate.DataTipRows(1,:).Label = '\bf i';
        s2.DataTipTemplate.DataTipRows(1,:).Value = 'YData';
        s2.DataTipTemplate.DataTipRows(2,:).Label = '\bf j';
        s2.DataTipTemplate.DataTipRows(2,:).Value = 'XData';
        s2.DataTipTemplate.DataTipRows(end+1) = dataTipTextRow('\bf N',215);
    patch('Parent',ax,'xData',supX','yData',supY','LineWidth',.9,...
            'EdgeColor','black','FaceColor','None','PickableParts','None');
    plot(ax,groundX',groundY',...
            'k','LineWidth',.9,'HandleVisibility','Off','PickableParts','None');
    if exist('P','var')
        % Plot beams
        %   Beams color is tied to the force in the beam, beams in tension
        %   are red, while beams in compression are blue. The darker the
        %   color, the larger the magnitude of the force in the beam.
        for ii = 1:P.Bnum
            plot(ax,P.cuse(:,ii),P.ruse(:,ii),'k',...
                'LineWidth',4.75,'HandleVisibility','Off','PickableParts','None')
            plot(ax,P.cuse(:,ii),P.ruse(:,ii),'Color',P.colorVal(ii,:),...
                'LineWidth',3.75,'HandleVisibility','Off','PickableParts','None');
        end

        scatter(ax,20,6,100,'MarkerEdgeColor','#EDB120',...
            'MarkerFaceColor','#EDB120','PickableParts','None');

        % Re-plot nodes used in the truss
        %   We want to see these nodes, but the other nodes are not
        %   necessary, so it is ok if unused nodes are covered by the beams
        unique_nodes = unique([P.Aluse;P.Beuse],'rows');
        s3 = scatter(ax,unique_nodes(:,2),unique_nodes(:,1),30,'filled',...
            'LineWidth',1,'MarkerEdgeColor','k','MarkerFaceColor','w');
            s3.DataTipTemplate.DataTipRows(1,:).Label = '\bf i';
            s3.DataTipTemplate.DataTipRows(1,:).Value = 'YData';
            s3.DataTipTemplate.DataTipRows(2,:).Label = '\bf j';
            s3.DataTipTemplate.DataTipRows(2,:).Value = 'XData';
            s3.DataTipTemplate.DataTipRows(end+1) = dataTipTextRow('\bf N',...
                sub2ind([11,20],unique_nodes(:,1),unique_nodes(:,2)));
    end

    colormap(ax,P.map);
    colorbar(ax,'Ticks',[P.range(1) .5*P.range(1) 0 .5*P.range(2) P.range(2)],...
        'TickLabels',{sprintf('\\leq%i',P.range(1)),string(.5*P.range(1)),'0',...
        string(.5*P.range(2)),sprintf('\\geq %i',P.range(2))});
    ax.CLim = P.range;
    hold(ax,'off');

exportgraphics(fig,P.file);
handles = [ fig, ax ];
end
```

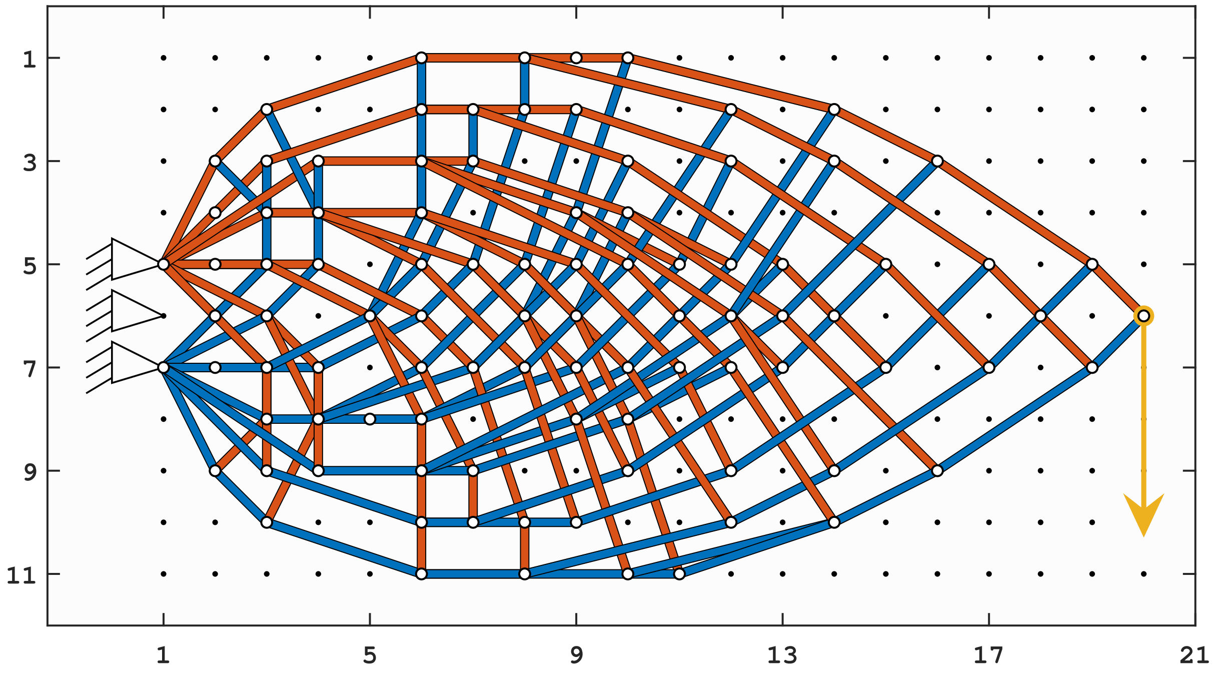
A.3 Upper Triangular 2D to Linear 1D Index Transformation Function

```
function out = H_ut(in1,in2)

if exist('in2','var')
    u = in1;
    v = in2;
    if u(:) >= v(:)
        error('First index must be less than second index');
    end
    w =  .5 .* (v-1) .* (v-2) + u;
    out = w;
else
    w = in1;
    v = ceil(.5 * (1 + sqrt(8*w + 1)));
    u = w - .5 .* (v-1) .* (v-2);
    out = [u;v]';
end
end
```

B Tension-Compression Truss Plots



(a) Categorical visualization of LP ① solution



(b) Categorical visualization of LP ② solution

Figure 5: Visualization of the solutions to our LP problems. Beam color represents the categorical classification of each beam. Orange beams are in tension while blue beams are in compression.