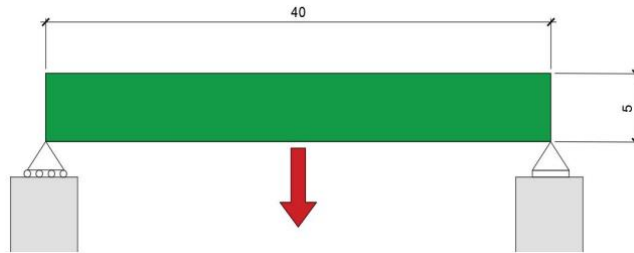


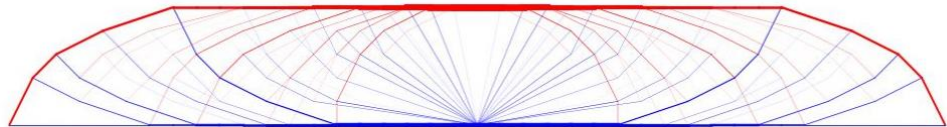
Methods for Truss Optimization (a work in progress)

By Saul Chaplin
Spring 2022

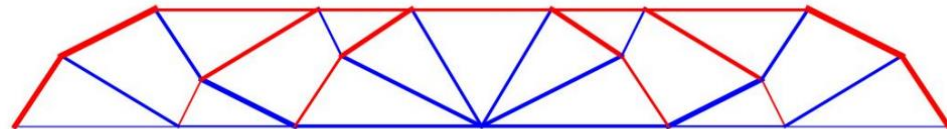
Problem: Weight minimization of truss to support loading. Subject to force equilibrium and tensile/compressive stress limits in material. Adding a cost per member, to account for constructability.



Member cost = 0



Member cost = .5



Member cost = 1

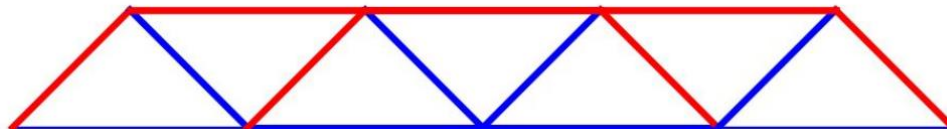


Figure: Example of truss optimization, as we move from a more optimized, less constructible bridge to a less optimized, more constructible bridge

CONTENTS

1.	Optimization for weight	3
1.1.	Case 1: Default.....	5
1.2.	Case 2: Max section area	6
1.3	Case 3: Double the height	7
1.4	Case 4: Add a no-go zone for ships to pass	8
1.5	Case 5: Optimizing for multiple load combinations	11
2.	Constructibility Optimization with Member Costs	12
3.	Stiffness via minimization of internal strain (coming soon)	14
4.	multiple materials with multiple costs (coming soon)	16
5.	Where to go next.....	17

1. OPTIMIZATION FOR WEIGHT

The optimization process for weight will be shown with an example. Say we have possible truss geometry where B is the number of bars, N is the number of nodes, and the material is A. Material A has a max tensile stress, $\sigma_{max,comp} = 1$ and a max compressive $\sigma_{max,tensile} = 1$. The max section size on the market of a member of material A is taken as 3. The density of material A is given as ρ . (Units will be added later). The force applied at the midpoint at the bottom is $F = 1$. The envelope of space (that will be referred to as the domain) that can be built in is shown below in figure 1.1.

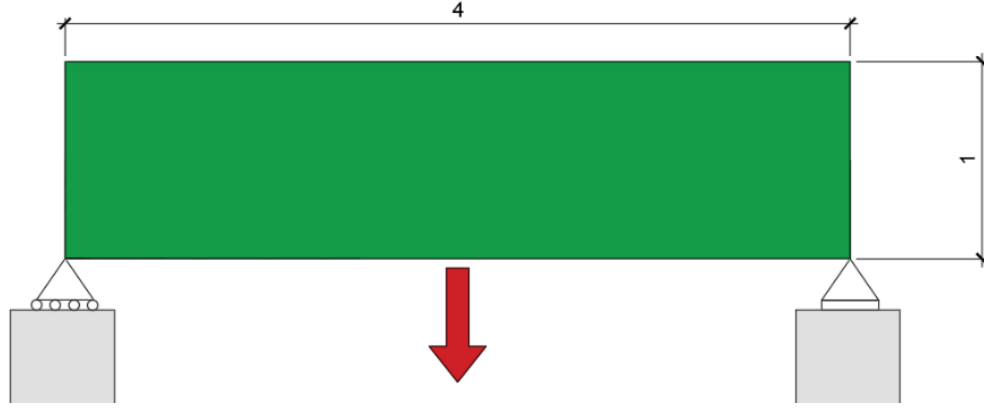


Figure 1.1: Truss envelope for first optimization task

For the first optimization task, we will attempt to minimize the weight of the truss, subject to the constraints that (1) a given loading pattern is supported, and (2) the stress in the members never exceeds the maximum material stress.

We can draw a grid of N nodes over the domain. We can find all the different iterations of bar connections, and call the number of bars B . The nodes and possible bars are shown in figure 1.2:

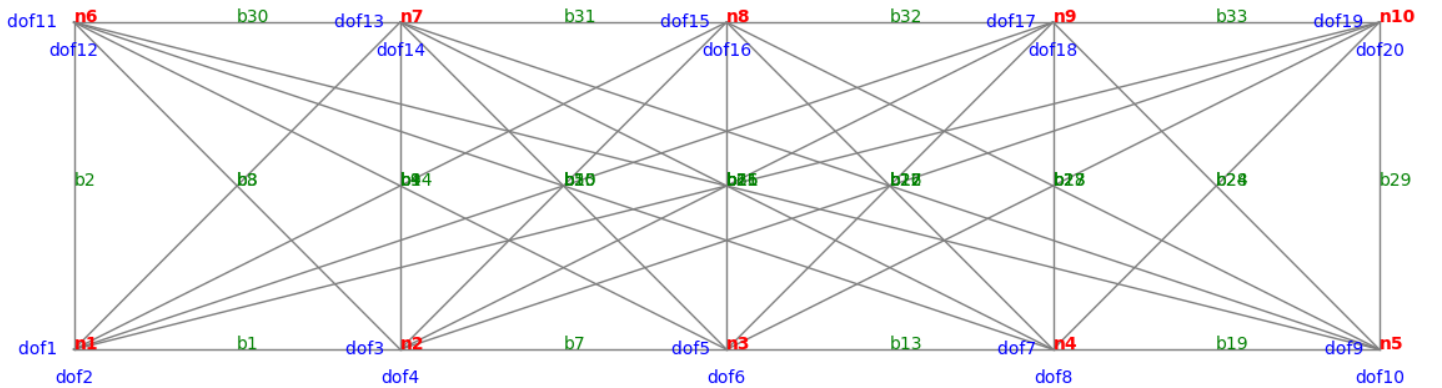


Figure 1.2: Possible nodes and members for truss optimization

Our goal is to minimize weight, so let's minimize mass. Let's define the objective function as

$$\text{minimize}(\text{mass}) = \text{minimize}(\rho * \text{volume})$$

Thus, we can minimize for volume, and we will be also minimizing for mass. We can write the optimization problem as:

$$\text{minimize}(\text{volume}) = \text{minimize}(\mathbf{a}^T \cdot \mathbf{l})$$

, where \mathbf{a} is a vector of the sizes of the sections of each member and \mathbf{l} is a vector of the lengths of each element. In English, we are simply summing up, for every member, the length times the area, to obtain the total volume. The variables of optimization (aka the variables changed to minimize the objective function here are \mathbf{a} and \mathbf{fint}).

The problem is subject to two sets of conditions

(1) Force equilibrium constraint: we have that for all free dof, the sum of the internal forces must be equal to the external forces. In equation form, that looks like:

$$\mathbf{B}_{2N \times B} * \mathbf{fint}_{B \times 1} = \mathbf{Fext}_{2N \times 1} \odot \mathbf{dof}_{2N \times 1}$$

, where:

\mathbf{B} is the connectivity matrix, with dimensions $2N \times B$, where the element b_{ij} is the force in dof i from one unit of tension in bar j , multiplied by the corresponding $\mathbf{dof}(i)$. A good example is bar 8. For bar 8, the b_{ij} can be written as

$$b_{18} = \frac{\sqrt{2}}{2} * \mathbf{dof}(1) = \frac{\sqrt{2}}{2} * 1$$

$$b_{28} = \frac{\sqrt{2}}{2} * \mathbf{dof}(2) = \frac{\sqrt{2}}{2} * 0$$

In the case of the truss example above, writing the first few elements, we have:

$$\mathbf{B}_{2N \times B} = \begin{matrix} \text{dof1} \\ \text{dof2} \\ \vdots \end{matrix} \left\{ \begin{matrix} \overbrace{\begin{bmatrix} 1 * 1 & 0 \\ 0 & 1 * 0 \end{bmatrix}}^{b1 \quad b2 \quad \dots} \\ \ddots \end{matrix} \right.$$

$\mathbf{fint}_{B \times 1}$ is a vector that contains all the internal forces of the bars. (Note: although the f_{int} is not in the objective function it is still a variable of optimization).

$\mathbf{Fext}_{2N \times 1}$ is a vector of the external forces applied at each degree of freedom, \odot denotes element-wise notation

$\mathbf{dof}_{2N \times 1}$ is a vector of 1s and 0s where $\mathbf{dof}(i) = 1$ if dof i is free and $\mathbf{dof}(i) = 0$ if dof i is fixed

(2) Section equilibrium constraint: We need the stress in the material in any member not to exceed the max compressive or tensile stress of the section. We can write this constraint as:

$$\text{for all } i \text{ in } [1, B], \quad -\sigma_{\text{max,comp}} * \mathbf{a}(i) \leq \mathbf{fint}(i) \leq \sigma_{\text{max,tension}} * \mathbf{a}(i)$$

(In English, this is basically just that for all the members, the total force cannot exceed the force at which the section fails/yields in compression or tension).

The optimization problem above is written in Python¹, using a code modified from He 2019². The solver CVXPY, an open-source Python package for convex optimization problems, is used. A well commented code called *case1to4.py* is available at <https://github.com/saul-chaplin/TrussOptimization.git> for the code for the following 4 cases. For those less familiar with Python, set-up instructions and packages to download are available in appendix B of He 2019. Plotting functions were written for the truss and the possible members, forces and supports were defined, max stresses were defined, the connectivity matrix **B** was found.

Note: to run this case, make sure case = 1, as shown below in the code *case1to4.py* and for the future cases, just change case = 2 or case = 3 or 4, etc...

```
119  ##YOU CAN CHANGE THE OPTIONS HERE ##
120  case = 1 #Change between different cases
121  text = 1 #Switch to show/hide text on graphs. 1 to show, 0 to hide.
```

The optimization problem is run to obtain the following optimized truss shape:

1.1. Case 1: Default

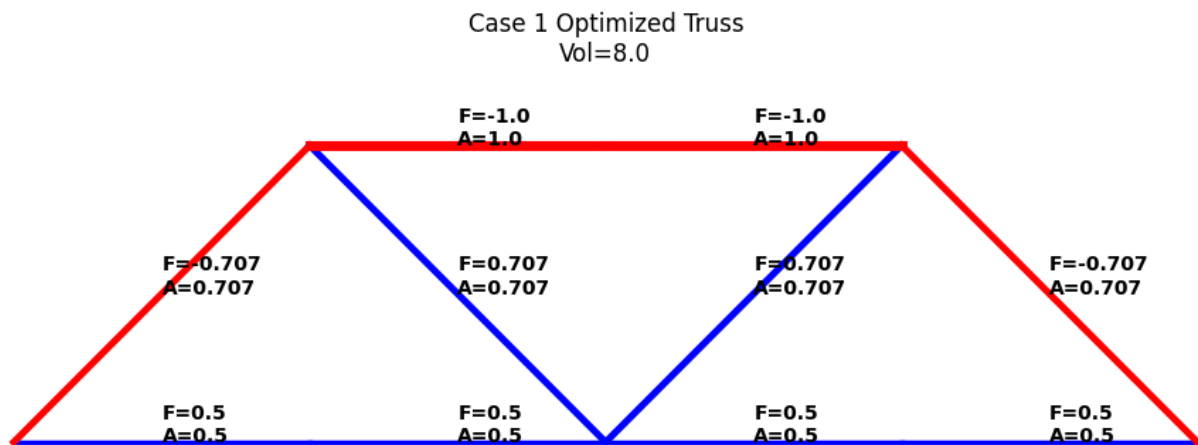


Figure 1.3: Optimized truss for weight

Some quick observations

- A quick numerical check reveals that the that each free node is in equilibrium, that the force of 1 occurring at node 3 (center bottom) is supported.
- Every section is either at its max tensile or compressive stress. This makes sense, because if they were not, then the optimization code would further reduce section size so that they were.
- Instead of continuous members across the top and the bottom, the members are “broken up” into 2 individual pieces. If you think about it, with our objective function, one member spanning the entire distance would provide exactly the same volume, and therefore there is no reason not to break up the member into two smaller pieces. It is notable, however, that (1) it would probably be easier to build the truss if the members were longer and there were less of them and (2) a joint in

¹ If you are new to Python (as I am!) I recommend this video <https://www.youtube.com/watch?v=rfscVS0vtbw>. Python is fairly similar to MATLAB, and you’ll pick it up quickly! It is fun, useful, and free!

² It is recommended that you read this paper.

a pin compression member creates instabilities. Therefore, this “problem” will be remedied in section 2 by adding a cost per member.

It is fun to fiddle around with the optimization by playing around with some input parameters.

1.2. Case 2: Max section area

The parameters from case 1 are unchanged except for the fact that a max section size is added, of $a_{max} = .5$. Note how this is below the area of many of the members from the Case 1 scenario.

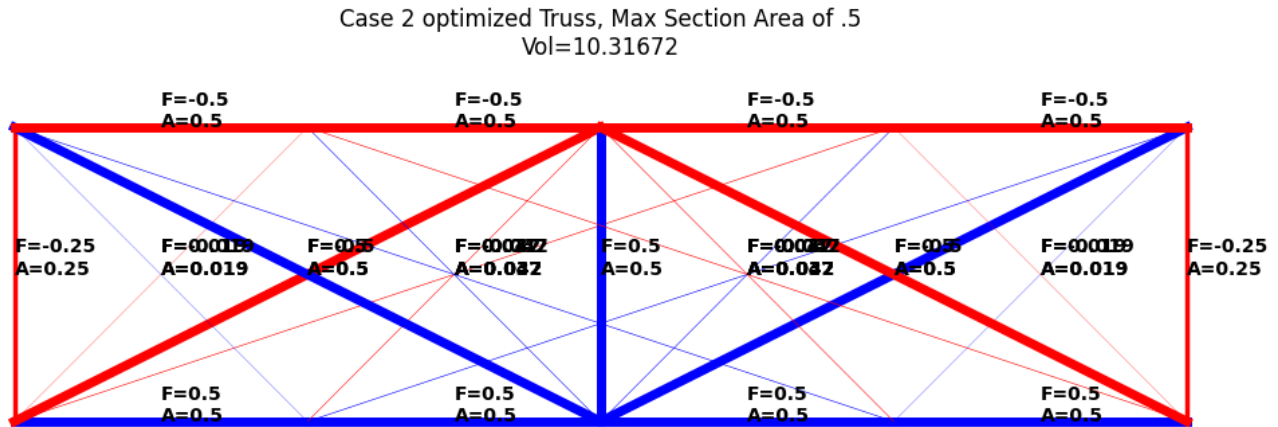


Figure 1.4: Output from a max section restrained truss

Comments:

- Redundant load paths have formed, because the determinant load path in case 1 is unable to take the load, as section sizes were limited to .5
- Objective function of the volume has climbed to 10.32 up from 8 (case 1). When we add another restriction, the objective must either increase or stay constant, and it has
- Tiny members have appeared. Those members are due to numerical errors in the optimization. Those members will be eliminated in part 2 when we add a member cost.

1.3 Case 3: Double the height

The parameters from case 1 are unchanged except that the height of the allowable building space was doubled from 1 to 2. The new bridge domain, and layout of possible members are shown in figure 1.5. The optimized truss is shown in figure 1.6.

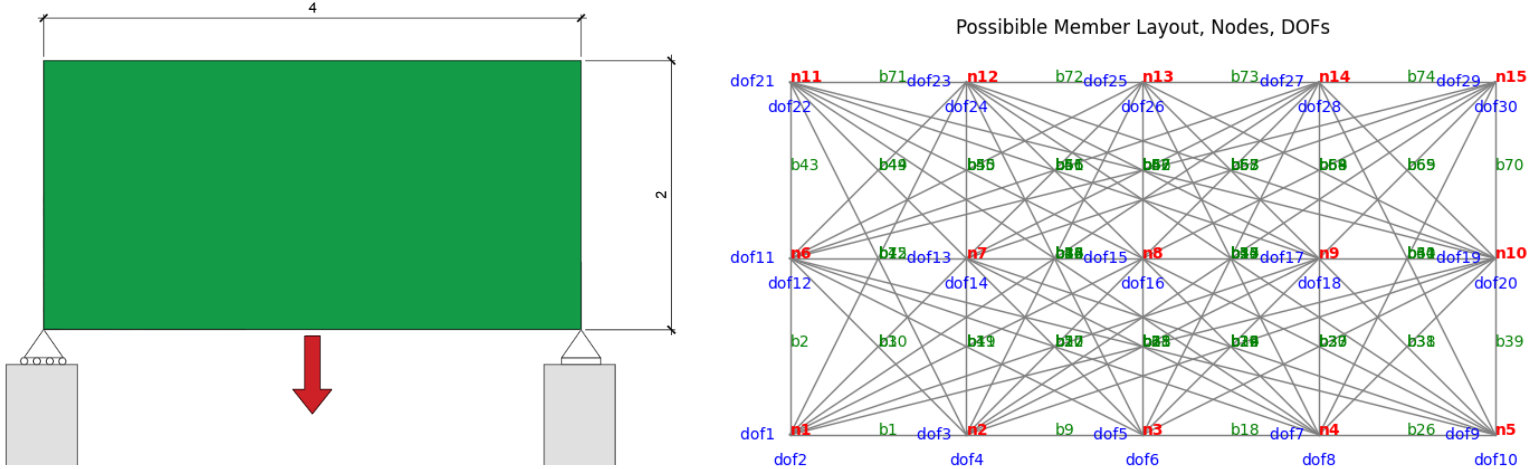


Figure 1.5: Bridge domain and possible members for a double-height scenario

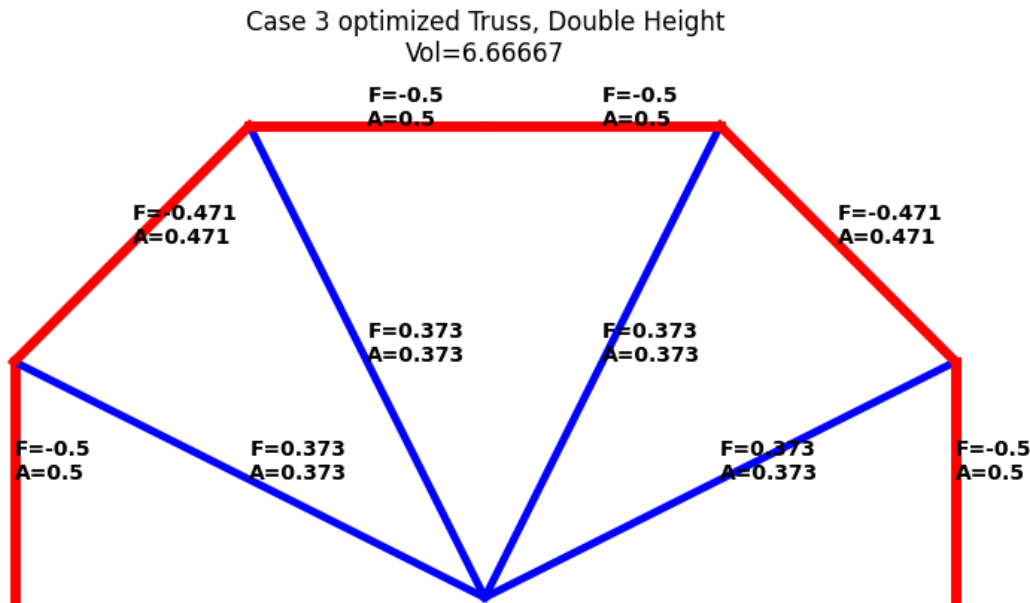


Figure 1.6: Optimized truss for double-height scenario

Comments:

- Volume has been reduced to 6.67, down from 8 in case 1. When we reduce the restrictions, (i.e. make the envelope larger), the objective function should either decrease or stay the same, and it has.
- The supports transfer no lateral force. Thus, the members to the left and right are perfectly vertical. Basically, we have a simplification of a half-circle arch. Also, the structure looks a bit like a half bike wheel, where the blue members are the spokes, in tension supporting the force which would be the weight of the bike carried through the hub.
- The structure, under pinned joints, has no lateral stability. If we were to design such a structure, the columns on the left and right would have to have to take significant moment at their bases.

1.4 Case 4: Add a no-go zone for ships to pass

To illustrate the versatility and limitations of the code, one can imagine a shipping channel like the one shown below in figure 1.7. A bridge is to be built above the channel, in a domain that provides enough room for the ship to pass underneath. The weight of the deck and the cars is simplified as point loads at deck level. A crane may be installed in the future at the midpoint, causing the need for a higher force resistance. Tollbooths will be built at either end, causing an increase in the force. The bridge will connect to approach tower on either side, and it is desired that the bridge itself be free-standing. The possible members, nodes, and dofs are shown in figure 1.8. The optimized truss is shown in figure 1.9.

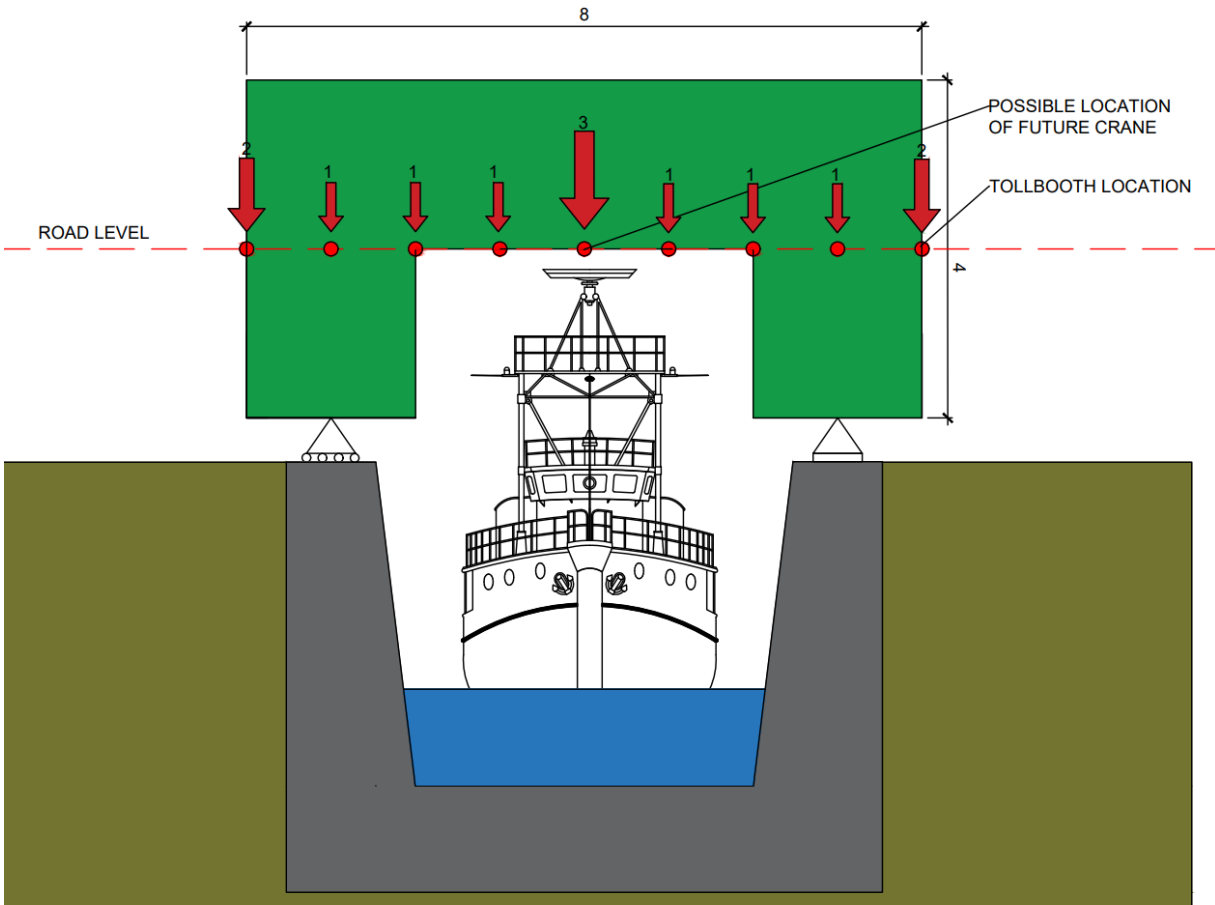


Figure 1.7: Domain, supports, and loads for case 5

Possible Member Layout, Nodes, DOFs

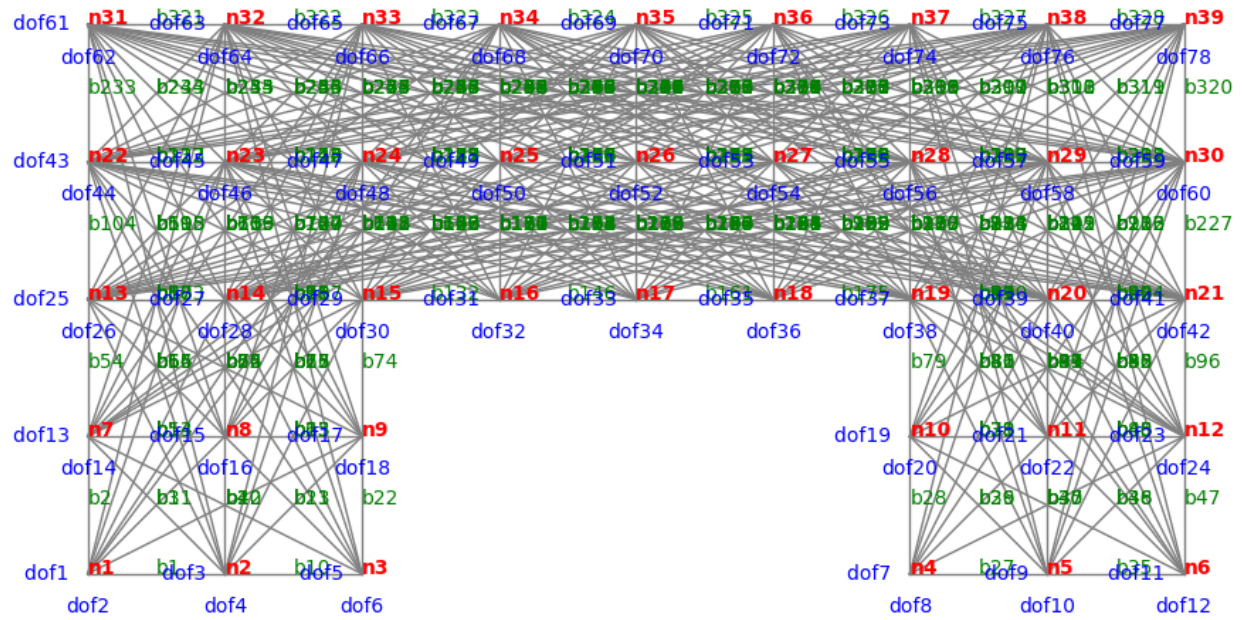


Figure 1.8: Layout of possible truss members, DOFs, and nodes

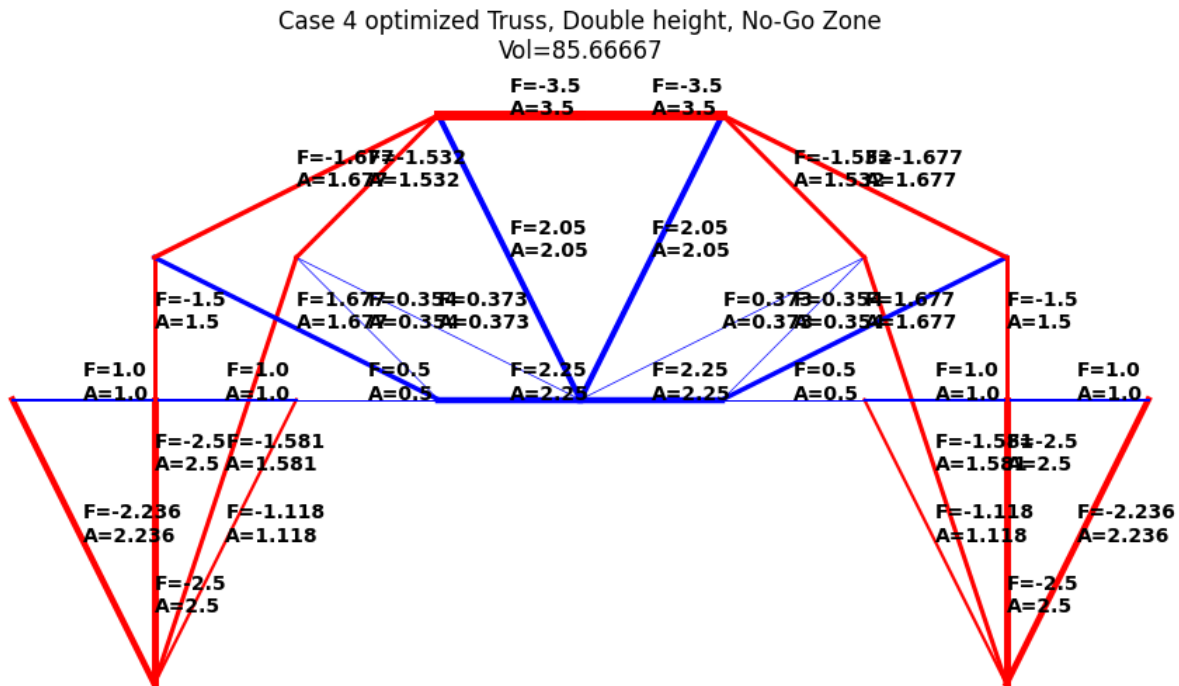


Figure 1.9: Optimized truss combination

Comments

- The shape is very weird looking. It is an optimized structure to support a deck that only puts loads at the specific given points in figure 1.7 and could not support loads in between the loading points. It is unclear where the given structure could support any lateral load.
- Several very small members are present that would make construction unnecessary difficult.
- Other things...

1.5 Case 5: Optimizing for multiple load combinations

(Coming soon!)

Example to be added. I derived the process mathematically and wrote it in AMPL (to be shown in section 3). Basically, just add another force-equilibrium constraint, and section stress constraint force that force combination. The process can be done in Python and is outlined in He 2019.

2. CONSTRUCTIBILITY OPTIMIZATION WITH MEMBER COSTS

To become more detailed.

Just change $\mathbf{l}_{B \times 1}$ the matrix of length of members, by adding a joint cost with value jc to each element. Our objective function is still:

$$\text{minimize}(\text{volume}) = \text{minimize}(\mathbf{a}^T \cdot \mathbf{l}_{B \times 1})$$

, but, in this case, $\mathbf{l}_{B \times 1} = \begin{bmatrix} l_{\text{member } 1} + jc \\ l_{\text{member } 2} + jc \\ \vdots \end{bmatrix}$

This method was originally suggested in Parkes 1978. The method is also outlined in He 2019.

Quick example (to be expanded) we can start with the case of a bridge, in a much larger problem than we had in part 1:

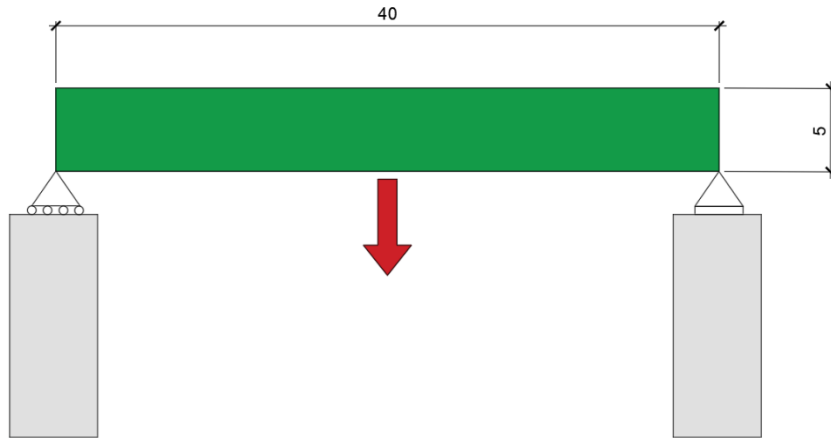


Figure 2.1: Truss envelope for first optimization task



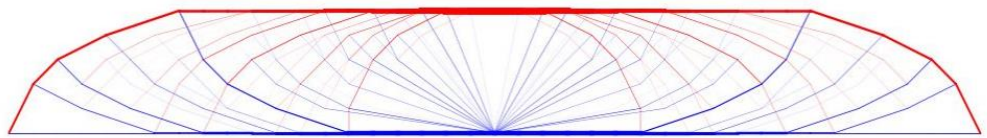
Figure 2.2: Possible nodes and members for truss optimization

The optimum truss was found using the code *BridgeWithMemberCost.py* available at <https://github.com/saul-chaplin/TrussOptimization.git>. In the code, to change member costs, just update the *jc* value in line 107.

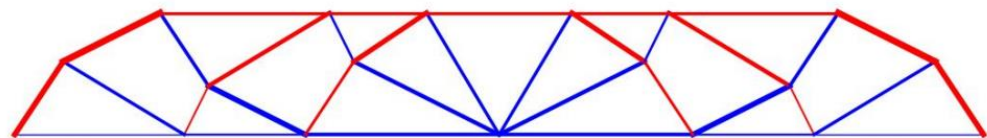
```
105  ##YOU CAN CHANGE THE OPTIONS HERE ##
106  case = 1 #Change between different cases
107  jc = 0 #Sets the joint cost value
108  plot_initial_layout = 0 #Plot initial layout
109  text = 0 #Switch to show/hide text on graphs. 1 to show, 0 to hide.
```

The following results were obtained:

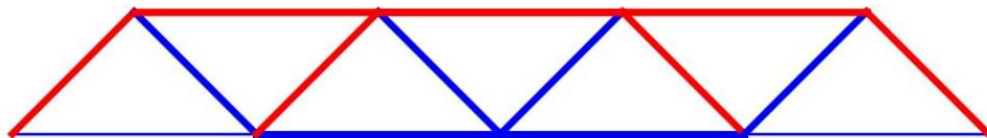
Member cost = 0



Member cost = .5



Member cost = 1



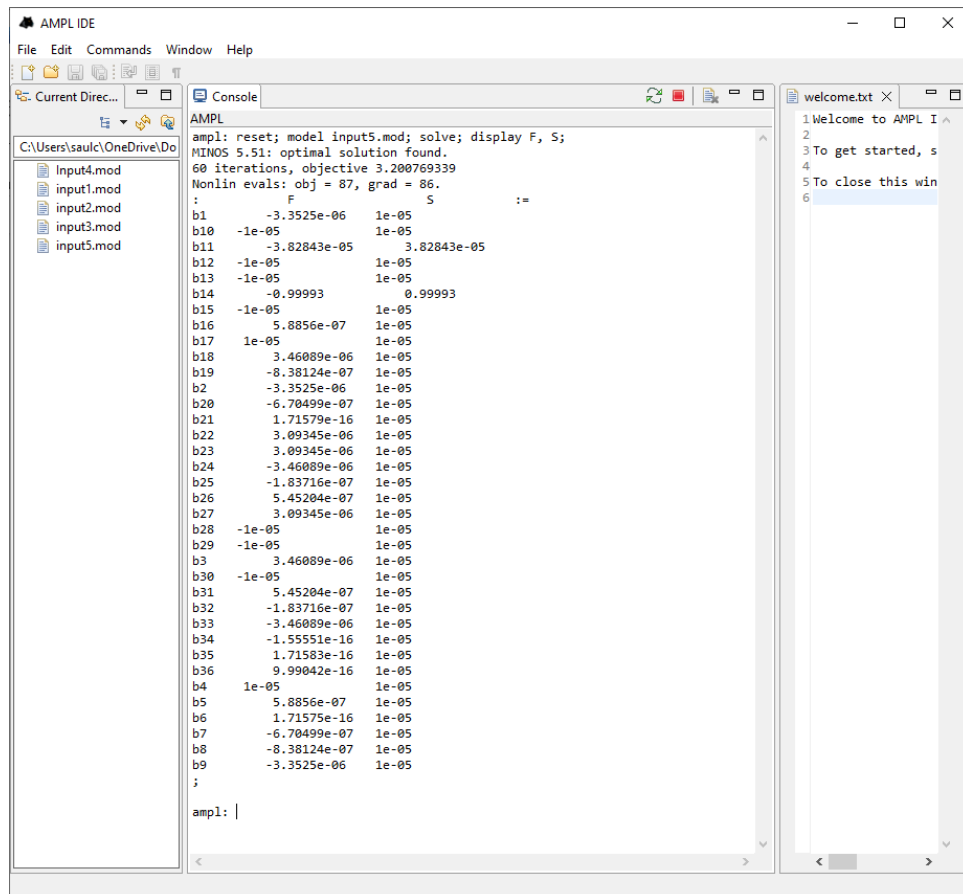
3. STIFFNESS VIA MINIMIZATION OF INTERNAL STRAIN (COMING SOON)

Optimization problem becomes nonlinear and much more complex when we minimize for stiffness and Python CVXP cannot solve any more. AMPL is a super powerfull optimization toolbox that can. (it is also used for everything from complex business decisions, to crazy engineering stuff, etc).

Download AMPL community edition: <https://ampl.com/ce/> .

AMPL handbook: <https://ampl.com/wp-content/uploads/BOOK.pdf>

Sneak peak of ampl coding for a non-linear optimization including weight, constructability, and stiffness (Details and derivations coming soon!!)



The screenshot shows the AMPL IDE interface. On the left is a file explorer showing a directory structure with files: Input4.mod, input1.mod, input2.mod, input3.mod, and input5.mod. The main console window displays the following text:

```
AMPL
ampl: reset; model input5.mod; solve; display F, S;
MINOS 5.51: optimal solution found.
60 iterations, objective 3.200769339
Nonlin evals: obj = 87, grad = 86.
:      F      S      :=
b1      -3.3525e-06      1e-05
b10     -1e-05           1e-05
b11     -3.82843e-05      3.82843e-05
b12     -1e-05           1e-05
b13     -1e-05           1e-05
b14     -0.99993         0.99993
b15     -1e-05           1e-05
b16     5.8856e-07       1e-05
b17     1e-05            1e-05
b18     3.46089e-06       1e-05
b19     -8.38124e-07      1e-05
b2      -3.3525e-06       1e-05
b20     -6.70499e-07      1e-05
b21     1.71579e-16       1e-05
b22     3.09345e-06       1e-05
b23     3.09345e-06       1e-05
b24     -3.46089e-06      1e-05
b25     -1.83716e-07      1e-05
b26     5.45204e-07       1e-05
b27     3.09345e-06       1e-05
b28     -1e-05           1e-05
b29     -1e-05           1e-05
b3      3.46089e-06       1e-05
b30     -1e-05           1e-05
b31     5.45204e-07       1e-05
b32     -1.83716e-07      1e-05
b33     -3.46089e-06      1e-05
b34     -1.55551e-16      1e-05
b35     1.71583e-16       1e-05
b36     9.99042e-16       1e-05
b4      1e-05            1e-05
b5      5.8856e-07        1e-05
b6      1.71575e-16       1e-05
b7      -6.70499e-07      1e-05
b8      -8.38124e-07      1e-05
b9      -3.3525e-06       1e-05
;

ampl: |
```

On the right, a small window titled 'welcome.txt' contains the following text:

```
1 Welcome to AMPL I
2
3 To get started, s
4
5 To close this win
6
```

Input files can look more like this. Recommend you get a text editor. Here I using VS.code, which I can get automatically after downloading anaconda navigator. (see <https://www.youtube.com/watch?v=rfscVS0vtbw>, the video I recommended for Python, where he installs Anaconda Navigator). You can also just download it straight from online website: <https://code.visualstudio.com/> You can install the AMPL extension that creates the pretty colors.

```

1  #Minimizing an objective function of the Volume, Constructibility, and Strain Energy
2
3  set MEMBERS;
4  set NODES;
5  param Tf = 1;
6  param Smax = 10; #creating a maximum section if we want
7  param long {MEMBERS} > 0;
8
9  param fx {NODES}; #External force in x
10 param fy {NODES}; #External force in y
11 param dofx {NODES}; #Whether or not x at the node is a DOF
12 param dofy {NODES}; #Whether or not y at the node is a DOF
13 param Cx {NODES, MEMBERS}; #
14 param Cy {NODES, MEMBERS}; #
15 param jc = 1; #Joint cost
16 param E = 10; #Young's Modulus
17 param Cd = 1; #How much we care about deflection
18
19 var S {j in MEMBERS} >= 0.00001, <= 10; # Section of each bar
20 var F {j in MEMBERS}; # Force in each bar
21 minimize tri_obj: sum {j in MEMBERS} (S[j]*(long[j]+jc)+Cd*((F[j])^2)*long[j]/(S[j]*E));
22 subject to Fx {i in NODES}: sum {j in MEMBERS} F[j] * Cx[i,j] = -fx[i]*dofx[i]; #Node force equilibrium in x
23 subject to Fy {i in NODES}: sum {j in MEMBERS} F[j] * Cy[i,j] = -fy[i]*dofy[i]; #Node force equilibrium in y
24 subject to max_axial {j in MEMBERS}: F[j] <= Tf*S[j]; #Max tensile stress
25 subject to min_axial {j in MEMBERS}: F[j] >= -Tf*S[j]; #Max compressive Stress
26
27 data; ##### DATA STARTS HERE #####
28 set MEMBERS := b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11 b12 b13 b14 b15 b16 b17 b18 b19 b20
29 set NODES := n1 n2 n3 n4 n5 n6;
30 param: long :=
31 b1 1.0
32 b2 2.0
33 b3 1.0
34 b4 1.4142135623730951
35 b5 2.23606797749979
36 b6 2.0
37 b7 2.23606797749979
38 b8 2.8284271247461903
39 b9 1.0
40 b10 1.4142135623730951
41 b11 1.0
42 b12 1.4142135623730951
43 b13 2.23606797749979
44 b14 2.0
45 b15 2.23606797749979
46 b16 2.23606797749979
47 b17 1.4142135623730951
48 b18 1.0
49 b19 2.8284271247461903
50 b20 2.23606797749979
51 b21 2.0

```

4. MULTIPLE MATERIALS WITH DIFFERENT COSTS, WEIGHTS, PROPERTIES (COMING SOON)

5. WHERE TO GO NEXT

- 3D optimization for everything
- Non-“discrete” domain optimization using finite elements domain, using homogenization, and then penalization to get something buildable.
- Figure out how tf SIMP optimization works. Here is a sick paper which references it’s use for trusses: https://www.e3s-conferences.org/articles/e3sconf/pdf/2019/66/e3sconf_eece18_04017.pdf
- Can I generalize for frame structures? Optimize moment frames? Allow members to take moment (so, for example, you can load in the middle of a beam).
- Soooo much more....

REFERENCES

Agrawal A, Verschueren R, Diamond S, Boyd S (2018) A rewriting system for convex optimization problems. J Control Decision 5(1):42–60. <https://www.cvxpy.org/>

"A Python script for adaptive layout optimization of trusses", L. He, M. Gilbert, X. Song, Struct. Multidisc. Optim., 2019. Article Link: <https://link.springer.com/article/10.1007/s00158-019-02226-6>
Code download link: [Python Script for Truss Layout Optimization \(shef.ac.uk\)](#)

Parkes E (1978) Joints in optimum frameworks. Int J Solids Struct 11(9):1017–1022. <https://www.sciencedirect.com/science/article/pii/002076837590044X>

(by my prof at Universidad Catolica in Santiago, Chile, uses homogenization method)

- Gutiérrez, Sergio. OPTIMAL STRUT-AND-TIE MODELS USING FULL HOMOGENIZATION OPTIMIZATION METHOD. https://www.researchgate.net/publication/257194208_Optimal_Strut-and-Tie_Models_Using_Full_Homogenization_Optimization_Method
- Experimental evaluation of optimized strut-and-tie models for a dapped beam. [Experimental evaluation of optimized strut-and-tie models for a dapped beam - Oviedo - 2016 - Structural Concrete - Wiley Online Library](#)

Other sweet articles:

Fairclough, Helen et. Al. "Theoretically optimal forms for very long-span bridges under gravity loading." <https://royalsocietypublishing.org/doi/10.1098/rspa.2017.0726>

https://www.e3s-conferences.org/articles/e3sconf/pdf/2019/66/e3sconf_eece18_04017.pdf

Zegard, Tomás, y Glaucio H. Paulino. "GRAND3 — Ground Structure Based Topology Optimization for Arbitrary 3D Domains Using MATLAB". Structural and Multidisciplinary Optimization, vol. 52, no 6, diciembre de 2015, pp. 1161–84. DOI.org (Crossref), <https://doi.org/10.1007/s00158-015-1284-2>

Bike Parking

\vec{P}_x , and \vec{P}_y is supported, without causing the stress in any member to exceed the lower or upper bounds of the allowable stress envelope.

We define the objective function as

$$c(\vec{S}, \vec{F}_{int}) = \rho * (trsp(\vec{S}) \cdot \vec{L}) \text{ where:}$$

\vec{S} is a vector with dimension B where each entry, s_n , takes a value of the size of the cross-sectional area of the section. For example, $\vec{S} = [1, 10.45, 8.33, \dots, 0, 3.011]$

\vec{F}_{int} is a vector of dimension B where each entry f_n takes the value of the internal force in member n (positive in tension, negative in compression)

\vec{L} is a constant vector of dimension B where the entry L_n has the value of the length of the corresponding bar n .

The variables in the objective function are subject to the following constraints for the force

$$\begin{aligned} \mathbf{C}_x * \vec{F}_{int} &= \vec{P}_x \\ \mathbf{C}_y * \vec{F}_{int} &= \vec{P}_y \end{aligned}$$

\vec{P}_x is a constant vector of dimension N where the element k, n_k is the external force applied on node k in the direction +x (horizontal)

\vec{P}_y is the same with force applied in the direction +y

\mathbf{C}_x is a matrix of dimension NXB where the element c_{ij} is the force caused in node i in the direction +x by 1 unit of tension in element j

\mathbf{C}_y is a matrix of dimension NXB where the element c_{ij} is the force caused in node i in the direction +y by 1 unit of tension in element j

The constraints of the force are such that the stress in the bars does not exceed the maximum of minimum tensile stress:

$$\forall i \in [1, B] \quad \frac{-30}{s_i} \leq f_i \leq \frac{30}{s_i}$$

The constraint for the max section size is

$$\forall i \in [1, B] \quad 0 \leq s_i \leq 3$$

And voila! We are ready to write some code.

Objective function:

$$\text{minimize}(\text{weight}) = \text{minimize} \left(\rho * (\text{trsp}(\vec{S}) \cdot \vec{L}) \right) \text{ donde}$$

\vec{S} es vector de areas, cross seccionales is a vector with dimension B where each entry, s_n , takes a value of the size of the cross-sectional area of the section. For example, $\vec{S} = [1, 10.45, 8.33, \dots, 0, 3.011]$

$$Cn_{B \times 3} = \begin{bmatrix} \text{Node cord. of node 1 mem. 1} & \text{Node cord. of node 2 mem. 1} & \text{Len. of mem. 1} \\ \text{"for member 2"} & \text{"for member 2"} & \text{"for member 2"} \\ \dots & \dots & \dots \end{bmatrix}$$

$$Nd_{N,2} = \begin{bmatrix} \text{node 1 x} & \text{node 1 y} \\ \text{node 2 x} & \text{node 2 y} \\ \dots & \dots \end{bmatrix}$$

$$dof_{2N \times 1} = \begin{bmatrix} 1 \\ 1 \\ \dots \end{bmatrix} \text{ (I do not undertand the point of the dof in this problem ...)}$$

$$PML = \begin{bmatrix} i \text{ node index} & j \text{ node index} & L & \text{True or False} \end{bmatrix}$$

Note: if the 4 column is set to true, then that element is included in the optimization code

(I do not

aplicada en nodo k en la dirección de +x (horizontal)

\vec{P}_y es un vector constante de dimensión N donde el elemento k corresponde a la fuerza externa aplicada en nodo k en la dirección de +y (vertical)

C_x es un matriz de dimensión $N \times B$ donde el elemento c_{ij} es la fuerza en la dirección de +x en nodo i por una unidad de tensión en barra j

C_y es un matriz de dimensión $N \times B$ donde el elemento c_{ij} es la fuerza en la dirección de +y en nodo i por una unidad de tensión en barra j

Con un reticulado dado donde B es el número de barras, N es el número total de nodos, con material \underline{a} y material \underline{b}

La función objetiva, de minimizar el costo, es definido como la función:

$$c(\vec{ID}, \vec{S}, \vec{F}_{int}) = 1 * (\vec{ID} \cdot (trsp(\vec{S}) * \vec{L})) + 20 * ((\vec{1} - \vec{ID}) \cdot (trsp(\vec{S}) * \vec{L})) \text{ donde:}$$

\vec{ID} es un vector variable discreto de dimensión B donde cada entrada, id_n toma un valor discreto entre 0 y 1. El 1 en lugar id_n denota que barra n , b_n , es hecho de material \underline{a} mientras el 0 denota que está hecha de material \underline{b} . Por ejemplo: $\vec{ID} = [1 \ 0, 0, 1, \dots, 0, 1]$

\vec{S} es un vector variable continuo dimensión B donde cada entrada s_n toma un valor del tamaño de la sección n , en unidades de cm^2 . Por ejemplo: $\vec{S} = [1, 10.45, 8.33, \dots, 0, 3.011]$

\vec{F}_{int} es un vector variable de dimensión B donde cada entrada f_n toma el valor de la fuerza interna en barra n (positivo es tracción).

\vec{L} es un vector constante de dimensión $1 \times N$ donde entrada s_n toma el valor de la longitud de barra n , b_n .

Los variables en la función objetiva son sujetos a las siguientes condiciones:

$$C_x * \vec{F}_{int} = \vec{P}_x$$

$$\mathbf{C}_y * \overrightarrow{F_{int}} = \overrightarrow{P_y}$$

$\overrightarrow{P_x}$ es un vector constante de dimensión N donde el elemento k corresponde a la fuerza externa aplicada en nodo k en la dirección de $+x$ (horizontal)

$\overrightarrow{P_y}$ es un vector constante de dimensión N donde el elemento k corresponde a la fuerza externa aplicada en nodo k en la dirección de $+y$ (vertical)

\mathbf{C}_x es un matriz de dimensión NXB donde el elemento c_{ij} es la fuerza en la dirección de $+x$ en nodo i por una unidad de tensión en barra j

\mathbf{C}_y es un matriz de dimensión NXB donde el elemento c_{ij} es la fuerza en la dirección de $+y$ en nodo i por una unidad de tensión en barra j

El estrés en barra b , σ_b , en cualquier barra de material a tiene que ser en entre $.3 \text{ kN/cm}^2$ y -30 kN/cm^2 . El estrés en barra b , σ_b , en cualquier barra de material b tiene que ser entre 30 kN/cm^2 y -30 kN/cm^2 . Para ponerlo en una forma matemática

$$\sigma_b \in \begin{cases} [-30, .3] & \text{si } id_b = 1 \\ [-30, 30] & \text{si } id_b = 0 \end{cases}$$

El tamaño máximo de la sección de barra b , s_b , necesita ser entre 0 y 3 cm^2 si la barra es de material a y entre 0 y 30 cm^2 si la barra es de material b. Matemáticamente:

$$s_b \in \begin{cases} [0, 3] & \text{si } id_b = 1 \\ [0, 30] & \text{si } id_b = 0 \end{cases}$$

