Truss Topology: Maximum Load Determination of a Truss by Linear Programming

MCEN 5125 Project #1

Hanwen Zhao

Abstract

In this report we describe how to find the minimum number of beams by linear programming (LP). The solution to two different cost functions are presented with numerical results.

1 Introduction

Numerical optimization is an extremely powerful tool for solving big and complex problem that would be difficult or costly to solve by conventional methods. In this report, we will solving a truss topology problem by formulate is as a Linear Programming(LP) problem so that we can use the state-of-the-art solvers that are available in Matlab to solve it efficiently.

The original problem states that we have 11x20 grid shown in Figure 1 represent all the possible link locations. The node with the red arrow pointing down indicates the location of the load which the solution truss has to support. On the left, we have three anchor points are indicated by triangle signs. An engineer would interested in what would be the minimum number of beams to support this situation.

Let's do some simple analysis, with the 11x20 size grid, we have $11*20 = 220$ possible points, since each beam would require 2 points, we can calculate the total number of possible beams by using following equation.

$$C_n^k = \frac{n!}{(n-k)! * k!} \tag{1}$$

With $n = 220$ and $k = 2$, we have $C_{220}^2 = 24090$ total number of possible beams. It is an enormous number of beam to think or analyze. If we want to setup the problem in the conventional way, we will have tremendous equations to solve and worry about any missing equations. Therefore, it is a great opportunity to demonstrate the ability of linear programming. In the following sections, we will start with a small 2x4 grid and find the pattern to set up the LP for the 11x20 size grid and solve it numerically with two different setups.

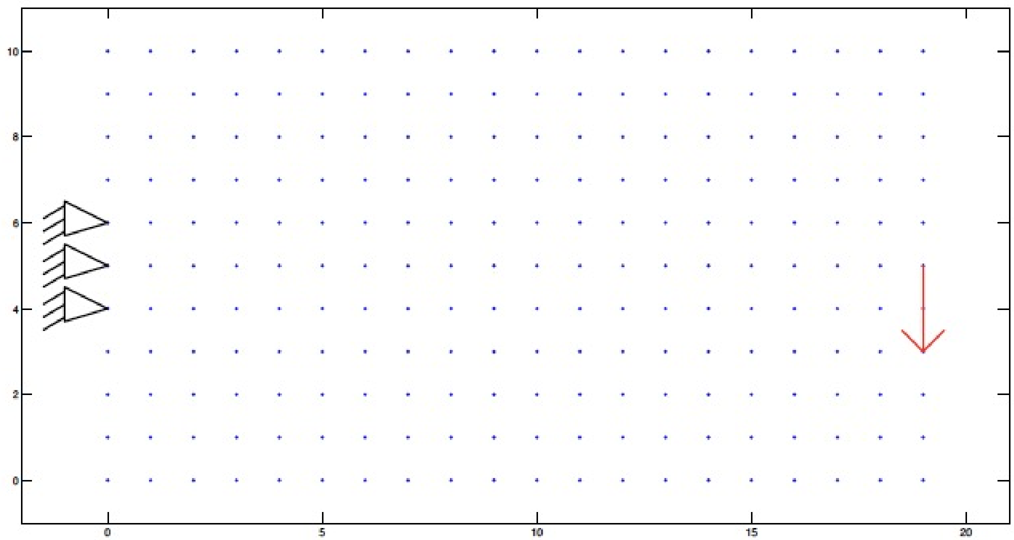

Figure 1: Truss Topology Problem

2 LP Formation

First, let's summarize the general truss topology problem, assume we have m numbers of beam and n numbers of nodes.
For each member of beam, we need to consider:

- possibly different cross section area
  - area x_i
  - shape of the cross section area(round, tube, I shape)
- different lengthl_i
- material properties
  - u_i is force in beam i, and we define the tension to be positive, compression to be negative
  - stress $\sigma = \frac{force}{area} = \frac{u_i}{x_i}$
  - we assume the material is "even", it fails at the same compression of tension stress: $-S_y \leqslant \sigma \leqslant S_y$

And for each node, we need to consider the following:

- where p<n members are anchors
- at each node, we will have external forces, where $f_i = \begin{bmatrix} f_{xi} \\ f_{yi} \end{bmatrix}$

There are several different way to consider a truss topology problem:

- For a given topology, find the maximum load.

- For a given force, find the lightest truss that will hold the force.
- For a given force, find the minimum number of beams.

Our problem falls into the second and the third categories since the topology is unknown to us. And we are assuming all the beams have the same shape and cross section area of 1. Therefore, we have decide to solve the problem by find the minimum number of beams.
Now we can start to analyze our static force equilibrium for each now. First, we assume all the anchors are strong enough so that they will not fail.

For each free node $i = 1, 2, \cdots n$ where n = total number of nodes - number of anchors, we have

$$\sum_{j=i}^{m} u_j \begin{bmatrix} cos\theta_{ij} \\ sin\theta_{ij} \end{bmatrix} + \begin{bmatrix} f_{x_i} \\ f_{y_i} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{2}$$

where

$$n_{ij} = \begin{bmatrix} cos\theta_{ij} \\ sin\theta_{ij} \end{bmatrix} = \begin{bmatrix} \frac{dy_{ji}}{\sqrt{dx_{ji}^2+dy_{ji}^2}} \\ \frac{dx_{ji}}{\sqrt{dx_{ji}^2+dy_{ji}^2}} \end{bmatrix} \tag{3}$$

and

$$\begin{bmatrix} dx_{ji} = x_j - x_i \\ dy_{ji} = y_j - y_i \end{bmatrix} \tag{4}$$

With all the information above, we can transform the truss topology problem into LP form:

$$\begin{array}{ll} \text{minimize} & \hat{c}^T x \\ \text{subject to} & \sum_{m}^{j=1} u_j * n_{ij} + f_i = 0 \quad i = 1, 2 \cdots n \\ & S_y \leqslant \sigma \leqslant S_y \end{array} \tag{5}$$

where

$$x = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix} \tag{6}$$

For unweighted problem, our cost function

$$c_{unweighted} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \tag{7}$$

For weighted problem, where we take the consideration of the length of each beam,

$$c_{weighted} = \begin{bmatrix} l_1 \\ l_2 \\ \vdots \\ l_m \end{bmatrix} \tag{8}$$

Also since we are calculating the l_1 norm, we need to introduce the slack variable **t** so that

$$\begin{bmatrix} u_1 \leqslant t_1 \\ -u_1 \leqslant t_1 \\ u_2 \leqslant t_1 \\ -u_2 \leqslant t_1 \\ \vdots \\ u_m \leqslant t_1 \\ -u_m \leqslant t_1 \end{bmatrix} \tag{9}$$

After rearrangement, we can conclude the problem in the following LP form:

$$\begin{aligned} \text{minimize} \quad & \hat{c}^T x \\ \text{subject to} \quad & \hat{A}x = \hat{b} \\ & \begin{bmatrix} I & 0 \\ -I & 0 \\ I & -I \\ -I & -I \end{bmatrix} x \leqslant \begin{bmatrix} S_y \\ S_y \\ 0 \\ 0 \end{bmatrix} \end{aligned} \tag{10}$$

where

$$x = \begin{bmatrix} \vec{u} \\ \vec{t} \end{bmatrix} \tag{11}$$

And the unweighted cost function becomes

$$c_{unweighted} = \begin{bmatrix} \vec{0} \\ \vec{1} \end{bmatrix} \tag{12}$$

Weighted cost function becomes

$$c_{unweighted} = \begin{bmatrix} \vec{0} \\ \vec{l} \end{bmatrix} \tag{13}$$

3 Numerical Solution

3.1 2x4 Grid

However, we have not find the pattern for the force equilibrium matrix A. Let's start with the small 2x4 grid. The grid as shown in Figure 2.

We assign a coordinate for each node, the bottom left node has the coordinate of (1,1), and the top right node has coordinate of (2,4). We will use these coordinates to calculate n_{ji} and length of each beam l_m.
Calculate the force equilibrium equations by hand, with the following equations:

$$n_{ij} = \begin{bmatrix} cos\theta_{ij} \\ sin\theta_{ij} \end{bmatrix} = \begin{bmatrix} \frac{dy_{ji}}{\sqrt{dx_{ji}^2 + dy_{ji}^2}} \\ \frac{dx_{ji}}{\sqrt{dx_{ji}^2 + dy_{ji}^2}} \end{bmatrix} \tag{14}$$

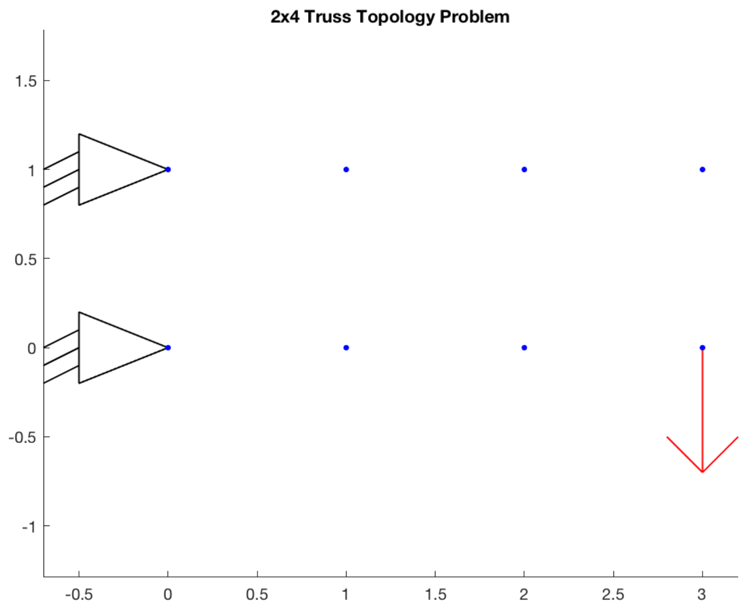

Figure 2: 2x4 Truss Topology Problem

we can get a 16x56 matrix. However, by looking at the x-direction force equilibrium equation matrix, we can see a pattern, from row 2 to end, it is a diagonal matrix with the value of row 1.

$$A_{node1_{fx}} = \begin{bmatrix} -1 & -1 & -1 & 0 & -0.7071 & -0.8944 & -0.9487 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.7071 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.8944 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.9487 \end{bmatrix} \tag{15}$$

After finishing building matrix A in both x-direction and y-direction, we have everything we need to solve the problem. First, let's solve the 2x4 grid with unweighted cost function as shown in Figure 3.

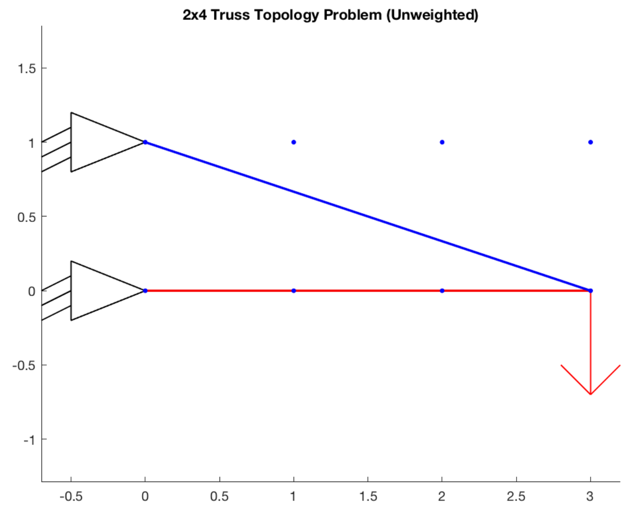

Figure 3: Unweighted solution for 2x4 grid.

And solve the 2x4 grid with weighted cost function as shown in Figure 4.

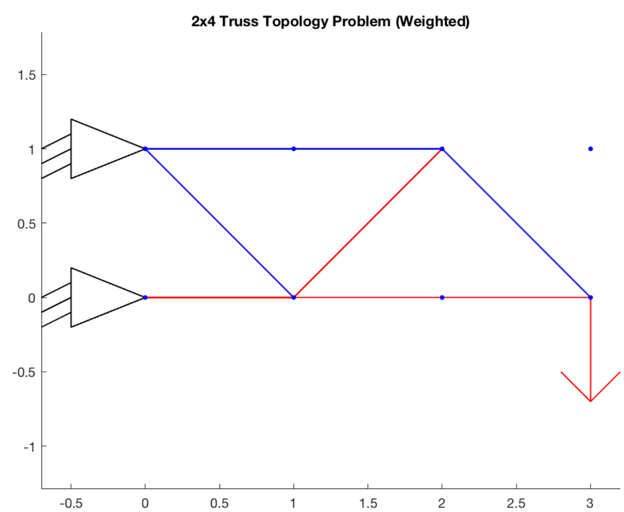

Figure 4: Weighted solution for 2x4 grid.

Notice that the blue color indicates tension and red color indicates compression, the color on . By compare two solutions, we can see that the unweighted solution has longer beams since there is no penalty for longer beam.

3.2 11*20 Grid

Now we can apply the strategies from above and solve for the big grid, since we will have very big matrix to solve, we can use sparse function to compress the matrix in order to save some memory.

First, let's solve the 11x20 grid with unweighted cost function as shown in Figure 5.

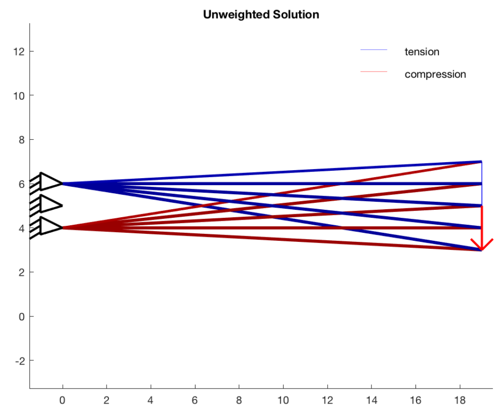

Figure 5: Unweighted solution for 11x20 grid.

Then solve the 11x20 grid with weighted cost function as shown in Figure 6.

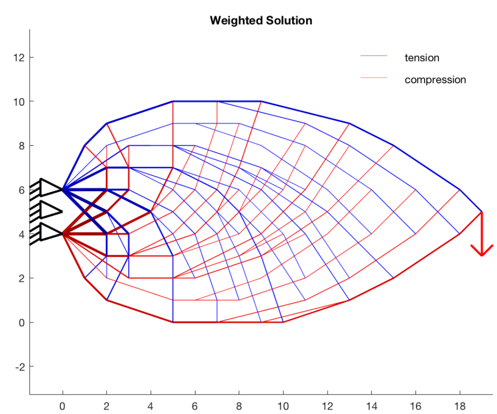

Figure 6: Weighted solution for 11x20 grid.

With a larger scale problem, we can easily see the difference between the weighted solution and unweighted solution. In the unweighted solution, the beams are extremely long, which sometimes is not practical in reality. With the length of each beam as the cost function, the solutions contains reasonable length of beams, which is more practical.

4 Conclusion

In conclusion, we presented the LP formulation for finding the minimum member of beams to support the force with the corresponding anchor points. It was shown through two numerical example how to use the LP formulation to get the minimum beam member. It was interesting that we can reduce the beam member down to 190 with total number of 24090 possible beams. Further constraints in the future could be added, such as using LP to find the best location for anchor points and including different shape of the cross section area beams.

5 Appendix

5.1 MATLAB Code

```
% MCEN 5125
% Project #1: Truss Topology
% Hanwen Zhao
% MEID: 650-703

tic
clc
close all
% define some variables
% grid size 11 rows * 20 columns
m = 11;
n = 20;
% maximum strss since yield strength is 8 and area is 1
Sy = 8;
% some general variables
numNodes = m*n;
numBeams = nchoosek(numNodes,2); % 24090 possible beams
% point number counted on row-major
anchorPointNum = [81 101 121];
forcePointNum = [120];
forceMag = -4;
% create matrix to store coordinates
pNodes = fliplr(combvec(1:1:n,1:1:m)');
% allocate memory
v = strings(m*n,1);
for i = 1:length(pNodes)
    % if the number smaller than 0, add the 0 in the front ex. "1" to "01"
    if pNodes(i,1)<10
        temp1 = strcat("0",num2str(pNodes(i,1)));
    else
        temp1 = num2str(pNodes(i,1));
```

```
    end
    if pNodes(i,2)<10
        temp2 = strcat("0",num2str(pNodes(i,2)));
    else
        temp2 = num2str(pNodes(i,2));
    end
    v(i) = strcat(temp1,temp2);
end
% use matlab built in to generate coordniates for all possible beams
x = nchoosek(v,2);
% create container to store coordniates
coordniates = zeros(numBeams,4);
% separate string back to number ex "0101" to [1 1]
for i = 1:numBeams
    firstNum = convertStringsToChars(x(i,1));
    secondNum = convertStringsToChars(x(i,2));
    coordniates(i,:) = [str2num(firstNum(1:2)),str2num(firstNum(3:4)),str2num(secondNum(1:2)),str2num(secondNum(3:4))];
end
% create Aeq matrix
Aeqx = sparse(zeros(numNodes,numBeams));
Aeqy = sparse(zeros(numNodes,numBeams));
% calculate all coefficients
beamCoeffx = zeros(1, numBeams);
beamCoeffy = zeros(1, numBeams);
weight = zeros(numBeams,1);
% calcualte force in x and y directions, as well as the distance
for i = 1:numBeams
    dy = coordniates(i,4) - coordniates(i,2);
    dx = coordniates(i,3) - coordniates(i,1);
    weight(i) = sqrt(dx^2+dy^2);
    beamCoeffx(i) = dy/sqrt(dx^2+dy^2);
    beamCoeffy(i) = dx/sqrt(dx^2+dy^2);
end
% fill the coefficients to matrix
colOffset = 1;
rowOffset = 1;
for i = fliplr(1:1:m*n-1)
    currentCoeffx = beamCoeffx(colOffset:colOffset+i-1);
    currentCoeffy = beamCoeffy(colOffset:colOffset+i-1);
    Aeqx(rowOffset:rowOffset+i,colOffset:colOffset+i-1) = [currentCoeffx;-diag(currentCoeffx)];
    Aeqy(rowOffset:rowOffset+i,colOffset:colOffset+i-1) = [currentCoeffy;-diag(currentCoeffy)];
    rowOffset = rowOffset+1;
```

```
    colOffset = colOffset+i;
end
% allocate memory for Aeq matrix, use sparse to save memory
Aeq = sparse(zeros(2*numNodes,2*numBeams));
Aeq(:,1:numBeams) = [Aeqx;Aeqy];
Aeq([anchorPointNum, anchorPointNum+numNodes],:) = 0;
beq = sparse(zeros(2*numNodes,1));
beq(forcePointNum+numNodes) = forceMag;
% built inequality matrix
Aiq = [speye(numBeams), -speye(numBeams);
       -speye(numBeams), -speye(numBeams)];

biq = [zeros(numBeams,1);
        zeros(numBeams,1)];
% unweighted cost function
f = [zeros(numBeams,1);ones(numBeams,1)];
% weighted
f_weight = [zeros(numBeams,1);weight];
% call linprog for unweighted
u_unweighted = linprog(f,Aiq,biq,Aeq,beq,-Sy*ones(2*numBeams,1),Sy*ones(2*numBeams,1));
u_unweighted = u_unweighted(1:numBeams);
% filter out small forces
resultBeamIndex1 = find(abs(u_unweighted)>0.01);

% call linprog for unweighted
u_weighted = linprog(f_weight,Aiq,biq,Aeq,beq,-Sy*ones(2*numBeams,1),Sy*ones(2*numBeams,1));
u_weighted = u_weighted(1:numBeams);
% filter out small forces
resultBeamIndex2 = find(abs(u_weighted)>0.01);

%% run section to redraw plots
figure
hold on
for i = 1:length(resultBeamIndex1)
    index = resultBeamIndex1(i);
    width = 0.5 + abs(u_unweighted(index) * 0.3);
    red = 1 - abs(u_unweighted(index))*0.5*0.1;
    blue = 1 - abs(u_unweighted(index))*0.5*0.1;
    if u_unweighted(index) >= 0
        line([coordniates(index,2)-1,coordniates(index,4)-1],[coordniates(index,1)-1,coordniates(index,3)-1],'Color',[red 0 0],'LineWidth',width)
    elseif u_unweighted(index) < 0
```

```
        line([coordniates(index,2)-1,coordniates(index,4)-1],[coordniates(index,1)-1,coordniates(index,3)-1],'Color',[0 0 blue],'LineWidth',width)
    end

end
plot([19 19],[5 3],'r','Linewidth',2)
plot([19 18.5],[3 3.5],'r','Linewidth',2)
plot([19 19.5],[3 3.5],'r','Linewidth',2)
plot([0 -1 -1 0],[6 5.7 6.5 6],'k','Linewidth',2)
plot([0 -1 -1 0],[5 4.7 5.5 5],'k','Linewidth',2)
plot([0 -1 -1 0],[4 3.7 4.5 4],'k','Linewidth',2)
plot([-1.5 -1],[3.5 3.8],'k','Linewidth',2)
plot([-1.5 -1],[3.8 4.1],'k','Linewidth',2)
plot([-1.5 -1],[4.1 4.4],'k','Linewidth',2)
plot([-1.5 -1],[4.5 4.8],'k','Linewidth',2)
plot([-1.5 -1],[4.8 5.1],'k','Linewidth',2)
plot([-1.5 -1],[5.1 5.4],'k','Linewidth',2)
plot([-1.5 -1],[5.5 5.8],'k','Linewidth',2)
plot([-1.5 -1],[5.8 6.1],'k','Linewidth',2)
plot([-1.5 -1],[6.1 6.4],'k','Linewidth',2)
axis equal
title('unweighted')
text(15.5, 12, 'tension');
text(13.5, 12.3, '_____', 'Color', 'blue');
text(15.5, 11, 'compression');
text(13.5, 11.3, '_____', 'Color', 'red');
title('Unweighted Solution')

figure
hold on
for i = 1:length(resultBeamIndex2)
    index = resultBeamIndex2(i);
    width = 0.5 + abs(u_weighted(index) * 0.3);
    red = 1 - abs(u_weighted(index))*0.5*0.1;
    blue = 1 - abs(u_weighted(index))*0.5*0.1;
    if u_weighted(index) >= 0
        line([coordniates(index,2)-1,coordniates(index,4)-1],[coordniates(index,1)-1,coordniates(index,3)-1],'Color',[red 0 0],'LineWidth',width)
    elseif u_weighted(index) < 0
        line([coordniates(index,2)-1,coordniates(index,4)-1],[coordniates(index,1)-1,coordniates(index,3)-1],'Color',[0 0 blue],'LineWidth',width)
    end

```

```
end
plot([19 19],[5 3],'r','Linewidth',2)
plot([19 18.5],[3 3.5],'r','Linewidth',2)
plot([19 19.5],[3 3.5],'r','Linewidth',2)
plot([0 -1 -1 0],[6 5.7 6.5 6],'k','Linewidth',2)
plot([0 -1 -1 0],[5 4.7 5.5 5],'k','Linewidth',2)
plot([0 -1 -1 0],[4 3.7 4.5 4],'k','Linewidth',2)
plot([-1.5 -1],[3.5 3.8],'k','Linewidth',2)
plot([-1.5 -1],[3.8 4.1],'k','Linewidth',2)
plot([-1.5 -1],[4.1 4.4],'k','Linewidth',2)
plot([-1.5 -1],[4.5 4.8],'k','Linewidth',2)
plot([-1.5 -1],[4.8 5.1],'k','Linewidth',2)
plot([-1.5 -1],[5.1 5.4],'k','Linewidth',2)
plot([-1.5 -1],[5.5 5.8],'k','Linewidth',2)
plot([-1.5 -1],[5.8 6.1],'k','Linewidth',2)
plot([-1.5 -1],[6.1 6.4],'k','Linewidth',2)
axis equal
title('weighted')
text(15.5, 12, 'tension');
text(13.5, 12.3, '_____', 'Color', 'blue');
text(15.5, 11, 'compression');
text(13.5, 11.3, '_____', 'Color', 'red');
title('Weighted Solution')

toc
```