**Tribhuvan University**

**Institute of Engineering**

**Pulchowk Campus**

**Department of Civil Engineering**

**MSc in Structural Engineering**

**A Project Report on**

**Structural Optimization of Trusses Using**

**Graphic Statics and Genetic Algorithm**

**Submitted By:**

**Bijay Ban (075MSStE001)**

**Nabin Tandan (075MSStE006)**

**Nishan Thapa (075MSStE008)**

**Pramod Tiwari (075MSStE013)**

**In partial fulfillment of the requirement for the degree of**

**Master of Science**

**In**

**Structural Engineering**

**February 2021**

**Tribhuvan University**

**Institute of Engineering**

**Pulchowk Campus**

**Department of Civil Engineering**

**Masters of Science in Structural Engineering**

**CERTIFICATE**

This is to certify that project entitled “Structural Optimization of Trusses Using Graphic Statics and Genetic Algorithm”, submitted by Bijay Ban, Nabin Tandan, Nishan Thapa and Pramod Tiwari in partial fulfillment of the requirements for the degree of Master of Science in Structural Engineering has been examined and declared successful.

…………………………. ……………………

Project Supervisor, MSStE Co-ordinator

Head of Department Prof. Dr. Gokarna Bahadur Motra  
Associate Prof. Dr. Bharat Mandal

**ABSTRACT**

Truss has been used as a structural system since the past. For economic design of truss, various methods have been developed and used. The field is in continuous development and newer methods are being developed. With the rapid development of computational power, newer algorithms are being developed so that the optimization process could be done by the usage of sophisticated computers available nowadays. In this study, we explored some of the aspect of the optimization of roof truss using graphic statics along with genetic algorithm. Optimization of truss is done by using load path as objective function which is product of line of force diagrams with line of form diagrams. Several parameters of truss are investigated to get the intuitive insight in several types of truss used predominantly. Importance of topology optimization for the ultimate optimized truss is underscored in the study.

**ACKNOWLEDGEMENT**

We are grateful to IOE, Pulchowk Campus, Department of Civil Engineering, for providing us with an opportunity to carry out the group project. It has been a great prospect for us to explore new ideas working as a team.

We are highly indebted to our project supervisor, Associate Professor Dr. Bharat Mandal, for his continuous guidance, critical discussions and constant supervision throughout the project. His experience and instructions helped us carry out the project work smoothly when faced with difficulties.

Our deepest appreciation goes to all those who directly or indirectly contributed in making this project successful.

Bijay Ban (075MSStE001)

Nabin Tandan (075MSStE006)

Nishan Thapa (075MSStE008)

Pramod Tiwari (075MSStE013)

**TABLE OF CONTENTS**

**CERTIFICATE…………………………………………………………………… i**

**ABSTRACT………………………………………………………………….…… ii**

**ACKNOWLEDGEMENT……………………………………………………….. iii**

**TABLE OF CONTENTS………………………………………………………… iv**

[1 Introduction 1](#_Toc64444824)

[1.1 Background 1](#_Toc64444825)

[1.2 Need of Study 1](#_Toc64444826)

[1.3 Objectives of the Study 2](#_Toc64444827)

[1.4 Scope and Limitations of the Study 2](#_Toc64444828)

[2 Literature Review 3](#_Toc64444829)

[2.1 Structural Optimization of Trusses 3](#_Toc64444830)

[2.2 Structural Optimization of Trusses 3](#_Toc64444831)

[2.3 Graphic Statics 3](#_Toc64444832)

[2.4 Genetic Algorithm 6](#_Toc64444833)

[3 Methodology 9](#_Toc64444834)

[4 Graphic statics Model: sample examples 11](#_Toc64444835)

[4.1 Stepwise Procedure of Solving Truss Through Graphic Statics 11](#_Toc64444836)

[4.2 Comparison of Graphic Statics and ETABS Software 14](#_Toc64444837)

[5 Analysis results and discussion 16](#_Toc64444838)

[6 Conclusions 22](#_Toc64444839)

[7 FURTHER RECOMMENDATIONS 22](#_Toc64444840)

[8 References 23](#_Toc64444841)

[9 ANNEXES 24](#_Toc64444842)

[9.1 ANNEX I – Summary of Truss Types Used in Analysis 24](#_Toc64444843)

[9.2 ANNEX II – MAJOR CLUSTER CODING IN Grasshopper PYTHON (ghPython) 28](#_Toc64444844)

[9.3 ANNEX III: Snapshots 35](#_Toc64444845)

# Introduction

## Background

Optimized design has always been the prime focus of engineering. Similarly, we structural engineers seek the structural systems that are capable to perform intended purpose keeping the project cost to the minimum as possible. For this, many people have developed several methods for the optimization of the structural systems such as ground structure method, continuum method, graphic statics, etc. These methods have their own merits and limitations and the quest for further refined method is a perpetual process.

Structural truss is a system of triangulated framed structures in which the arrangement of members and connections at their ends are such that the external loads are applied to the joints or panel joints. Truss are only able to transfer the axial force so they support the load basically by the axial force in the members. Trusses have been the prime choice as a structural system for long span bridge and long span roof as they provide cheaper and elegant option with proper utilization of material. In common practices, Pratt truss, K truss and Warren truss are used as the truss topology. These trusses are used ubiquitously due to the fact that they are easier to erect and construct in the field. But with the rapid development of technologies and mechanization of construction process, it has now become possible to construct various other topology of trusses which would have been difficult and not considered previously. Thus, we should now explore novel topologies of optimized trusses.

## Need of Study

Structural optimization is the most important techniques for economic design of the structures. This study focuses primarily on the importance of the structural optimization of the roof trusses. The structures are designed for safety conditions but not much consideration is given in the optimization aspect. The trusses are designed simply using the topology that are used in the existing structures without exploring the newer topologies for overall optimization of the roof trusses. Long span roof trusses contain numerous steel members. For constructional ease, every member could not be optimized for different sizes; the design should be limited within some limited variation of sections. This results in sections which are not equally stressed that ultimately leads to heavier material usage than the optimum required leading towards the extravagance of a lot of money. So, it has been the utmost need for exploring the options for topology, shape and size optimization such that the steel sections used in the trusses can be more or less equally stressed resulting in the optimum usage of the materials used i.e., “real optimized structural truss”.

## Objectives of the Study

The objectives of the study are as follows:

* To suggest the optimized truss sections for various spans of trusses.
* To introduce the method of Graphic Statics as a possible method of truss optimization.

## **1.4** Scope and Limitations of the Study

In order to achieve these objectives, the analysis of the truss is done with the help of graphic statics along with genetic algorithm for the optimization. The analysis of the trusses is done with help of graphic statics in the Rhino application.

The limitations of the study can be summarized as:

* + The study is conducted with loading on dominant direction only. Multiple load combinations in multiple directions are not considered in the study.
  + The joints and connection plates are not directly integrated in the study. They are somehow integrated with loading factor.
  + The study does not consider the condition for buckling of the members. Also, dynamic properties of the loading are not taken into consideration.
  + The optimization is done using material weight as objective function. Exact estimation of costs including erection, labour charges, etc. has not been considered.

# Literature Review

## Structural Optimization of Trusses

## Structural Optimization of Trusses

Since the past, various methods of truss optimization have been proposed i.e., shape optimization, size optimization and topology optimization. Shape optimization works to search the best nodal positions of predefined nodes of the truss structure. Size optimization iterates the design procedure in order to fulfill the objective function of minimum cross-sectional area. Topology optimization works on the addition and removal of elements and nodes in a predefined domain in order to achieve minimum weight structure. Size and shape optimization are commonly used by designers in Nepal whereas the topology optimization is not commonly practiced.

One of the most commonly used approach is topology optimization by the means of material distribution problem which is typically based on solid isotropic material with penalization model. Using this approach, resulting design can have any size, shape and connectivity. This solution provides the valuable insight about the geometry of the layout of the member of the optimal structure. However, one of the issues associated with the solution is its associated discretization.

Ground structures approach is another popular structural topology optimization method. In this method, the design problem typically consists of assuming a base or ground structure with a given layout of members where the optimization can be conducted as a sizing problem with the cross-sectional area as the design variables. In this method, the final designs are already discretized but often with thousands of members. Due to the inclusion of thousands of members, the inability of the change of design domain, shape or connectivity are the limitations of this method.

Graphic statics method is the other promising alternative for topology optimization of truss. This study follows this method for the structural optimization of the long span trusses. The method of graphic statics is described in the section below.

## Graphic Statics

Graphic statics is the method to solve and analyze the member stress in truss. In Graphic Statics, the member lengths and forces can be determined solely using geometry. The required diagrams can be constructed with simple drafting tools (straight edges, triangles, and a scale) or by the use of simple equations for lines and the intersection of lines required to solve a truss.

Maxwell established that for certain trusses, the nodes and polygons that represent the geometry of the truss have reciprocal polygons and nodes in the force domain (Maxwell, 1870). Every node in the geometry domain maps into a polygon in the force domain, every polygon in the geometry domain maps into a node in the force domain and every line representing the line of action of each truss member maps into a reciprocal line in the force domain. The mapping used by Maxwell resulted in the reciprocal lines being perpendicular to each other. The use of a different mapping (a hyperbola rather than a paraboloid of revolution) by (Cremona, 1879) results in the reciprocal lines being parallel. Because these two figures (the form diagram and the force diagram) are reciprocal, the mapping can also go from the force domain to the geometry (form) domain. The length of each of the lines in the force domain are proportional to the axial force in the reciprocal line representing the truss member. The creation of a force diagram from a form diagram (with its external applied forces) is called Graphic Statics. The process of Graphic Statics used to be a standard method of analyzing trusses and could be done using simple drafting tools. Graphic Statics is no longer in common usage for analysis, having been replaced by more mathematical tools, but it still can be used as a design tool and, as discussed below, as a tool in creating optimal structures.

As discussed in(Beghini et al., 2014; Mitchell et al., 2016), Graphic Statics provides the information needed for minimizing the load path (or volume) of a truss with specified connectivity. The form diagram provides the length of each member and the length of each line in the force diagram provides the force in each member. Because of the reciprocal relationships observed by Maxwell, only simple graphical techniques are needed to determine the length of each member and its axial force; there is no need to compute a stiffness matrix or to solve a large system of equations.

The optimization of truss can be carried by minimizing the value of a load path as in (Ohlbrock & D’Acunto, 2020) by manipulating the force diagram. Having carried out optimization in force diagram rather than form result the structure to be always in equilibrium and no need to form the stiffness matrix as in force method which demand high computational time.

The objective function for the structural optimization in graphic statics is as follows:

where Vrepresents the total volume of the structure, σ is the value of stress, Pi is the internal force and Li is the length of the *i*th member, respectively

In Graphic Statics, the member *i* with length Li has a reciprocal line in the force diagram with length Li that is proportional to the force Pi in the member. Thus, the objective function can be rewritten as follows:

which is equal to the summation of the products of the lengths of the members in the form diagram and the lengths of their reciprocal members in the force diagram. This value can easily be calculated based solely on the geometry of the two diagrams.

The approach to the structure optimization in this project consists of using graphic statics to define the variables of the optimization process. The optimization problem is conducted in the force domain using force diagram as opposed to the previous approach which conducts the optimization in the geometric domain using the form diagram. In this approach, there is no need to triangulate the domain with members of very small areas, which can create numerical difficulties. Similarly, the results are always feasible if the members form closed polygons in the force domain (i.e., equilibrium is guaranteed). Using graphic statics, there is no need to compute or assemble stiffness matrices; only simple graphical relationships are needed. The final location of loads or supports do not need to be specified a priori when subjected to certain relationships but can change as the optimal solution is found. Moreover, they can readily accommodate different tensile and compressive stresses and the equilibrium constraints in the force diagram reduces the number of design variables required. As in (Lee et al., 2015, 2016) combining shape grammars and graphic statics, the proposed methodology enables the following: (1) rapid generation of diverse, yet statically equilibrated discrete structures; (2) exploration of various design alternatives without any biases toward pre-existing typologies; (3) customization of the framework for unique formulations of design problems and a wide range of applications; and (4) intuitive, bidirectional interaction between the form and forces of the structure through reciprocal diagrams.

## Genetic Algorithm

A genetic algorithm is solver that uses “evolutionary techniques.” This is done by generating a population of solutions based on genomes (variable subject to change) reacting to a fitness (desired parameter to be minimized or maximized). An effective solution is found, keeping “fit” genomes in a generation and breeding them with other favorable genomes in the following generation, as well as eliminating non-favorable solutions.

This process is very similar to natural selection in the real world since Galapagos iterates, or breeds, multiple generations of solutions until it finds what it believes to be the fittest solution. The image to the left shows the basic genetic algorithm process.

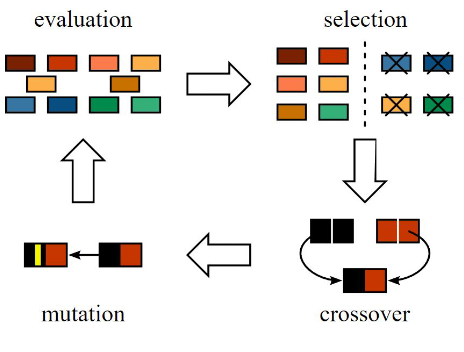


Figure General overview of Evolutionary solver mechanism

**GALAPAGOS CONNECTIONS**

The left image shows Galapagos connecting to sliders (maroon arrows) which alter parameters in this file. The fitness connection (green arrow) decides what parameter should be minimized or maximized. In our case, fitness is connected the total volume of the structure.

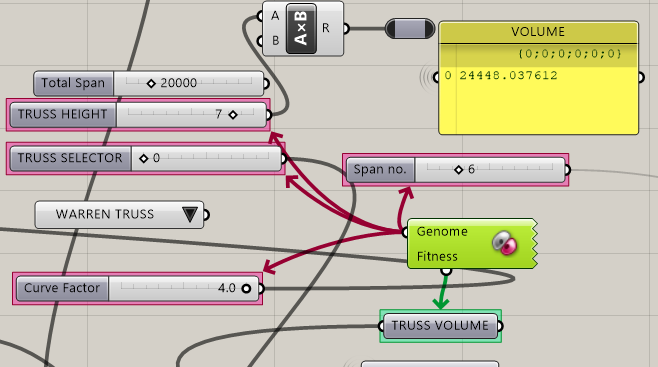


Figure Galapagos Genome and Fitness connection

**GALAPAGOS EVOLUTIONARY SOLVER**

Generic :

Fitness : Minimize

Runtime Limit : 30 minutes

Evolutionary Solver:

Population Size : 20 per generation

Initial Boost : 2

Maintain Population : 10%

Inbreeding : 75%

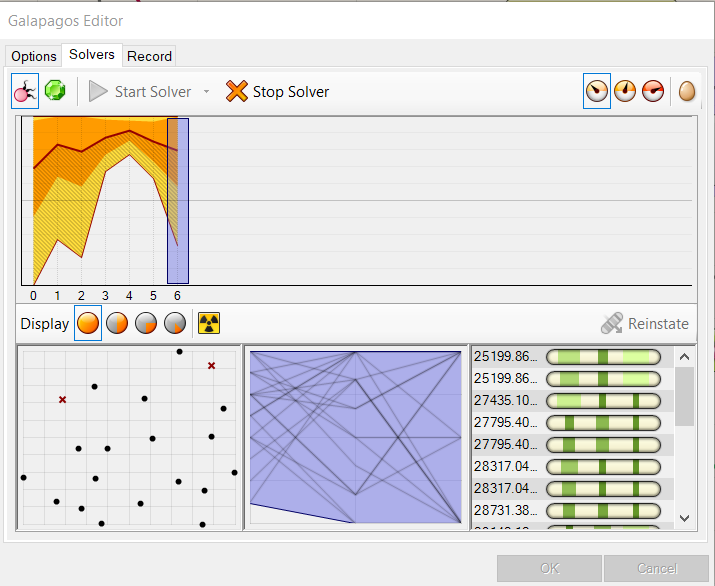


Figure Galapagos Output and User Interface

FITNESS FUNCTION

Minimize ( Fitness ) function of volume

Minimize

where,

L\_TD = Length of line in FORM Diagram

L\_FD = Length of line in FORCE diagram

C\_BIAS = 0.8 for Compressive Force

= 1 for Tensile Force

J\_BIAS = 0.05 Joint bias 4% of load on member

# Methodology

First of all, a specified general geometry and connectivity of a structure (known as form diagram) is prepared. Then, the force diagram of the corresponding form diagram is prepared following the procedure of graphic statics. In force diagram, the node degrees of freedom restrained by reciprocal relationships are found out. Then, design variables are assigned to the remaining nodes that are not restrained by reciprocal relationship. Then optimization is carried out using the genetic algorithm in the force diagram. The force diagram is updated through optimization process and the force diagram is determined from the updated form diagram.

The methodology can be expressed as a flowchart as shown below:

GROUND STRUCTURE

(FORM DIAGRAM)

FORCE DIAGRAM

ASSIGN DESIGN VARIABLE

CALCULATE OBJECTIVE

FUNCTION (LOAD PATH)

YES

CONVERGE?

NO

MODIFIED FORCE DIAGRAM

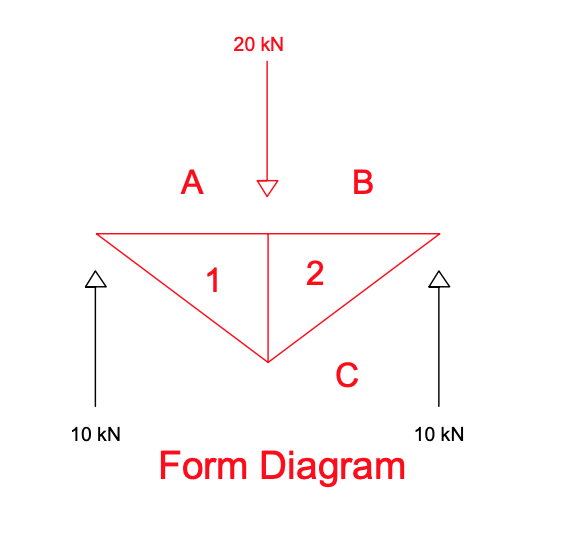
MODIFIED FORM DIAGRAM

# Graphic statics Model: sample examples

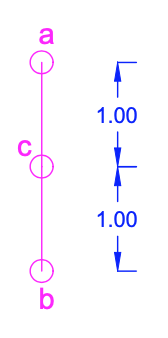
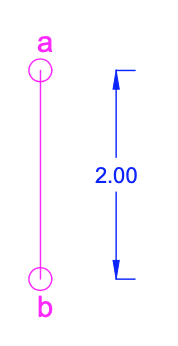
## Stepwise Procedure of Solving Truss Through Graphic Statics

The detail procedure for solving truss through graphic statics can be described with the help of following example:

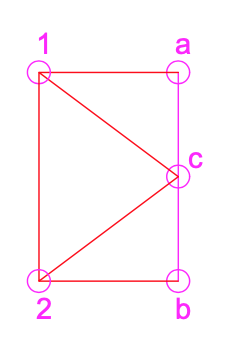
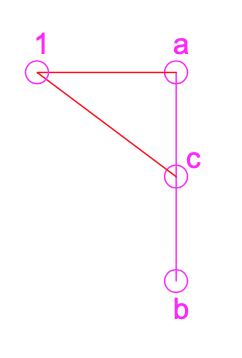
**Step I:** First of all, the external reaction forces of the truss are determined and the form diagram is prepared. The form diagram is labelled in different external regions with forces as boundary of the region. In the example, A, B and C are the regions. The internal regions are labelled as 1 and 2 with internal members as boundary.



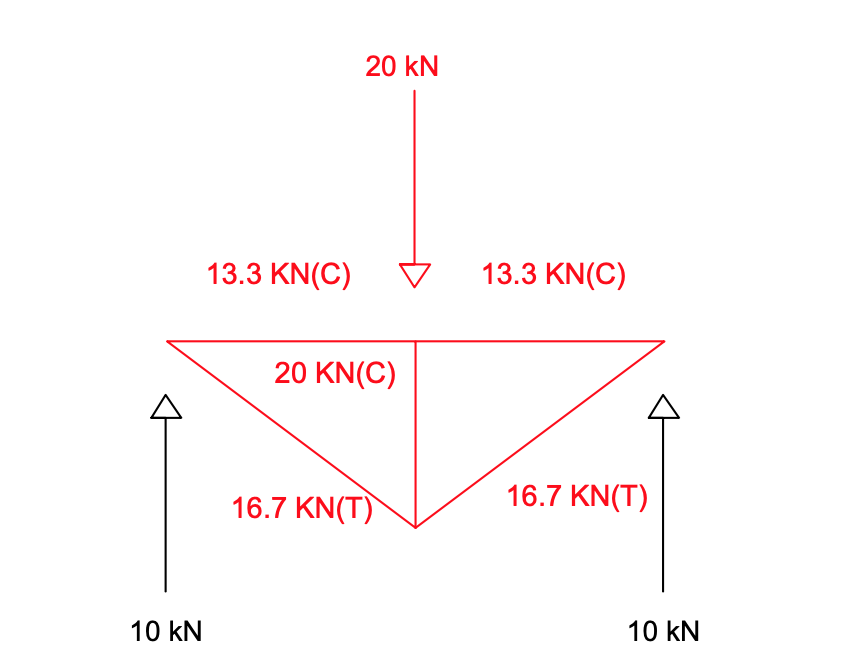
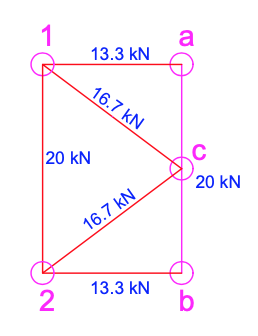
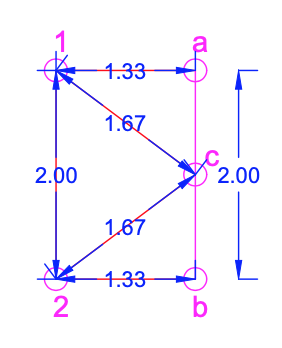
**Step II:** The next step is to prepare force diagram. The forces and region in the form diagram are represented by the lines and points in the force diagram respectively. Fixed scale is taken for the conversion of forces to lines i.e., for instance, in the example, 1m length in the force diagram is represented for 10kN force in form diagram. Thus, point ‘a’ is fixed and point ‘b’ is then 2m below point ‘a’. Also, point ‘c’ is 1m above point ‘b’.



**Step III:** The point ‘1’ is determined by the intersection of lines drawn from point ‘c’ and ‘a’ in force diagram as the region ‘1’ lies in between the region ‘A’ and ‘C’ in the form diagram. The lines in the force diagram are drawn parallel to the lines in the form diagram to respective member to the region as shown in figure below. Similarly, the point ‘2’ is determined as well.



**Step IV:** A closed polygon is formed in the force diagram if the truss is in equilibrium. The length of the lines then represents the internal forces of the truss members. The length of the force diagram is multiplied by the initial scale considered to calculate the forces in the truss members. Then, through reciprocal process, the forces are assigned to the member of the trusses in the form diagram. Thus, the truss is solved purely with graphical method without solving any complex mathematical equations.



In this method, a closed polygon is formed if the truss is in equilibrium. Similarly, the reverse is also always true. For any closed polygon in force diagram, it represents a form diagram of the truss which is always in equilibrium. Also, the length of the lines in the force diagram represents the forces in the members of the truss. So, a great intuition can be developed in the designers through this method in order to design the truss with required internal forces that is already in equilibrium.

## Comparison of Graphic Statics and ETABS Software

A sample truss as shown in figure below is taken. The truss is then modeled in the commercial finite element method-based software ETABS 18.0. The same truss is solved via graphic statics method. The loads are taken in the nodes as shown in the figure. The dead load is not taken into consideration (it can be incorporated through adding the joint loads appropriately). The internal forces in the truss members are compared from both the methods and they are found to be matched exactly.

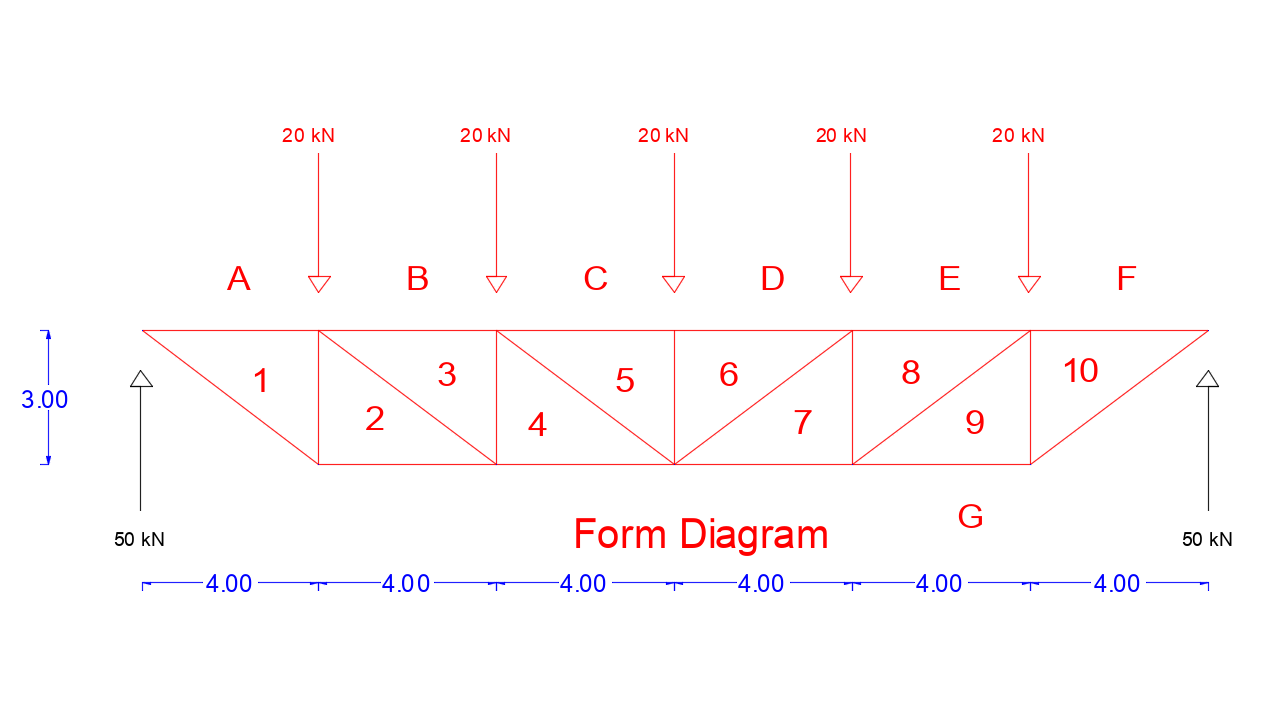


Figure 4 Form Diagram

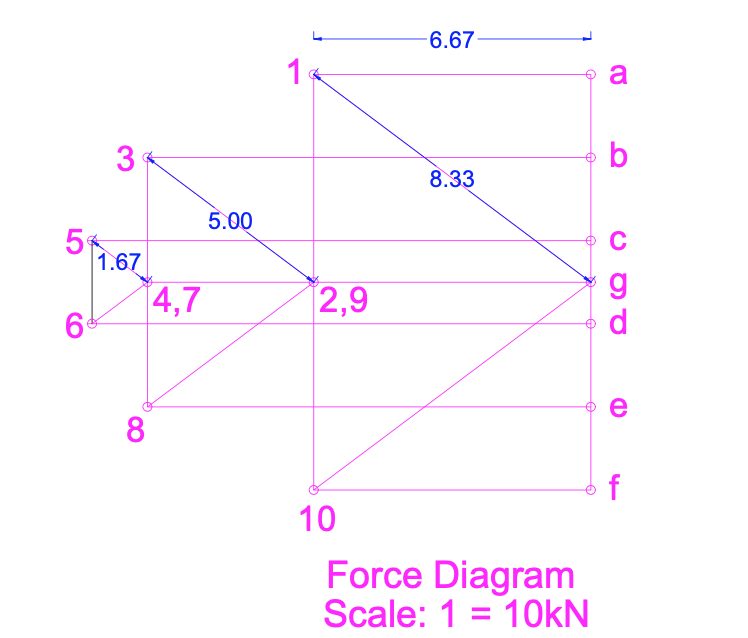


Figure 5 Force Diagram

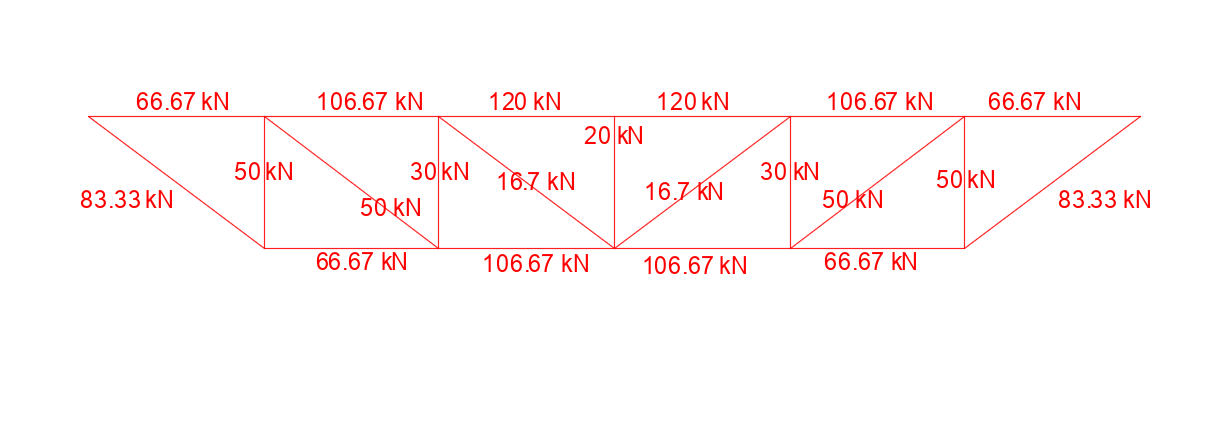


Figure 6Form Diagram with axial force determined from Graphic Statics

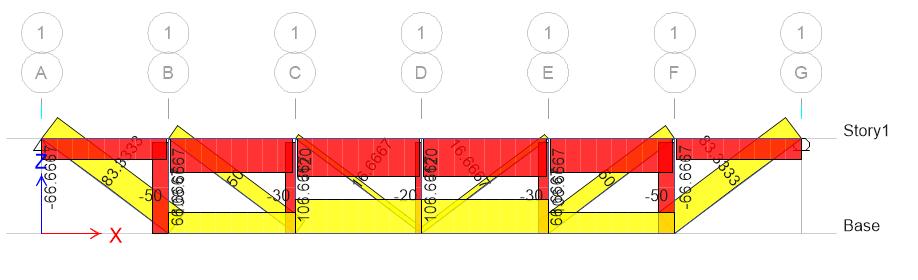


Figure 7 Force in Members determined from ETABS

# Analysis results and discussion

The result of analysis of trusses varying in their topology, span, height, and top chords curvatures along with result in graphic form are presented below: The analysis has been done in parametric work environment- Rhino, - Grasshopper using Graphic Statics to determine the force in the members of truss in equilibrium. Further, the optimization for varying height, span, top chords curvatures and truss types has been done in evolutionary solver-Galapagos Solver available in Rhino environment as a plugin.

Table 1 Fitness Values of Trusses for Span length 20 m

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type of Truss | Height (m) | No. of Spans | | | |
| 4 | 6 | 8 | 10 |
| Howe | 4 | 36326 | 35424 | 37031 | 39512 |
| Pratt | 4 | 33894 | 31047 | 31820 | 33842 |
| Warren | 4 | 33894 | 31053 | 30778 | 31409 |
| Howe | 8 | 32754 | 37413 | 43527 | 50156 |
| Pratt | 8 | 27891 | 28657 | 33107 | 38808 |
| Warren | 8 | 27891 | 28658 | 31022 | 33944 |
| Howe | 12 | 38047 | 46827 | 56810 | 70944 |
| Pratt | 12 | 30753 | 33695 | 42467 | 52951 |
| Warren | 12 | 30753 | 33694 | 38910 | 45939 |

Table 2 Fitness Values of Trusses for Span Length 60m

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Type of Truss | Height (m) | No. of Span | | | | |
| 4 | 6 | 8 | 10 | 12 |
| Howe | 10 | 360238 | 339263 | 344911 | 359909 | 379288 |
| Pratt | 10 | 341998 | 306432 | 305834 | 317357 | 334523 |
| Warren | 10 | 341998 | 306450 | 298027 | 299126 | 304690 |
| Howe | 8 | 417468 | 379757 | 374857 | 381492 | 393530 |
| Pratt | 8 | 402876 | 353491 | 343596 | 347451 | 357718 |
| Warren | 8 | 402876 | 353491 | 337341 | 332873 | 333857 |
| Howe | 12 | 326949 | 318831 | 333281 | 355654 | 381735 |
| Pratt | 12 | 305061 | 279433 | 286389 | 304592 | 328015 |
| Warren | 12 | 305061 | 279433 | 277007 | 282702 | 292201 |

Table 3 Fitness Value of Trusses for Span Length 80 m

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type of Truss | Height | Span Number | | | |
| 6 | 8 | 10 | 12 |
| Pratt | 12 | 580657 | 571895 | 585812 | 610418 |
| Warren | 12 | 580657 | 559389 | 556625 | 562662 |
| Pratt | 10 | 658257 | 635822 | 638767 | 653706 |
| Warren | 10 | 658257 | 625400 | 614444 | 613909 |

Table 4 Optimized Parameter of Truss for Various Span Length

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Span Length | Height | Span No | Ground Truss | Weight |
| 20 | 8 | 4 | Warren | 27891 |
| 60 | 12 | 8 | Warren | 277007 |
| 80 | 16 | 8 | Warren | 492508 |
| 100 | 18 | 8 | Warren | 800000 |

Table 5 Optimized Parameter of Truss For Various Span Length allowing Top Chords Curvature

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Span Length | Height | Span No | Ground Truss | Parametric | Weight |
| 20 | 6 | 10 | Pratt | 4 | 23726 |
| 60 | 12 | 14 | Pratt | 1 | 250025 |
| 80 | 16 | 14 | Pratt | 0.8 | 444653 |
| 100 | 18 | 16 | Pratt | 0.6 | 740076 |

*Note: Fitness Values represent the total volume of the truss*

Figure 8 Variation of Truss Fitness wrt No. of Division & Height For 20m Span

Figure 9 Variation of Truss Fitness wrt No. of Division & Height For 60 m Span

**INTERPRETATION OF RESULTS**

The results suggest that with increase in the truss height for same span and no. of divisions of span, the fitness value, initially decreases up to a certain point and afterwards the fitness value begins to increase. As the height of trusses increase, they are more prone to wind effects. Thus, height of trusses has been constrained with commonly adopted span to depth ratios of 6:1 to 15:1.

For a constant height of a given truss, with the increase in the number of divisions of span, at first fitness value goes on decreasing to certain point then it begins to increase afterward, similar to case where no. of divisions is variable. As for the Pratt truss of Span length 60m and height 10m, its seen that the fitness value at 4 no of division is 341998 goes on decreasing to 305834 at no. of divisions 8, after which its value increases, as seen in 10 no. of divisions where fitness value goes to 317357.

As seen from the results, the Howe Truss performs significantly poorly than other trusses for dominant UDL load case, in all considered truss cases. Warren truss performs better when top chord profile is straight, whereas, Pratt truss has lower fitness value for same span and height but parabolic top chord profile. For a Span of 60m, and straight top chord profile, Warren truss of height 12m with 8 no. of divisions of span had minimum fitness value compared to other, whereas, for parabolic top chord profile, Pratt Truss of 12m height and 14 no. of divisions of span had the minimum fitness value. The fitness value for these cases change by 10% when top chord profile change to parabolic.

Furthermore, if curvature of top chord is allowed to change, for minimum fitness value truss top chord adopts the shape parabolic arch. In this truss shape, the stresses in the vertical struts are minimized and stresses in the inclined struts are zero. Compared to the truss of same height and horizontal profile, the truss with top chord parabolic for all cases have lower fitness value. The percentage difference between these two shapes range from 15% for 20m span truss of height 8m to 7% for 100m span truss of height 18m. This suggests that arch shape trusses give minimum weight under Uniformly distributed load as dominant load case.

# Conclusions

* The calculations show that weight of a truss for given no. of spans decrease as we increase the height until a certain point from which it decreases. However, higher depths are prone to wind loading and limit buildability, thus are not desired.
* Comparing to straight profile upper chord, parabolic shape of the top chord in truss reduces the overall weight for same topology.
* Combination of Graphic Statics for structural analysis and Genetic Algorithm for optimization can be used with computational ease, to determine the optimum shape of common bridge and roof trusses. The project presents a system that allows users to generate truss shapes and topologies that are efficient for dominant load cases.

# FURTHER RECOMMENDATIONS

* Effect of buckling and weight of joints on the structure are represented using bias factors, which could be further refined to more sophisticated cases.
* Load cases in structure is limited to uniformly distributed load, assigned to truss bottom chord endpoints as per tributary length. Further dominant load cases in other directions can be applied for more practical design.
* Optimization of Graphic Statics – Force Diagram can result in interesting and construction friendly forms, which are not explored in this study, could be developed.
* Extension of 2-D analysis to 3-D analysis can be performed.

# References

Beghini, L. L., Carrion, J., Beghini, A., Mazurek, A., & Baker, W. F. (2014). Structural optimization using graphic statics. *Structural and Multidisciplinary Optimization*, *49*(3), 351–366. https://doi.org/10.1007/s00158-013-1002-x

Cremona, L. (1879). *The reciprocal figures in static graphics*. https://books.google.com/books?hl=en&lr=&id=RQqBgRgpZ10C&oi=fnd&pg=PA1&ots=7pHRUdc0Sj&sig=2MWEiHqTJ5hhKacYm1nZnwAnz2U

Lee, J., Fivet, C., & Mueller, C. (2015). Modelling with Forces: Grammar-Based Graphic Statics for Diverse Architectural Structures. In *Modelling Behaviour* (pp. 491–504). Springer International Publishing. https://doi.org/10.1007/978-3-319-24208-8\_41

Lee, J., Mueller, C., & Fivet, C. (2016). Automatic generation of diverse equilibrium structures through shape grammars and graphic statics. *International Journal of Space Structures*, *31*(2–4), 147–164. https://doi.org/10.1177/0266351116660798

Maxwell, J. C. (1870). I.–On Reciprocal Figures, Frames, and Diagrams of Forces. *Transactions of the Royal Society of Edinburgh*, *26*(1), 1–40. https://doi.org/10.1017/S0080456800026351

Mitchell, T., Baker, W., McRobie, A., & Mazurek, A. (2016). Mechanisms and states of self-stress of planar trusses using graphic statics, part I: The fundamental theorem of linear algebra and the Airy stress function. *International Journal of Space Structures*, *31*(2–4), 85–101. https://doi.org/10.1177/0266351116660790

Ohlbrock, P. O., & D’Acunto, P. (2020). A Computer-Aided Approach to Equilibrium Design Based on Graphic Statics and Combinatorial Variations. *CAD Computer Aided Design*, *121*, 102802. https://doi.org/10.1016/j.cad.2019.102802

.

# ANNEXES

## ANNEX I – Summary of Truss Types Used in Analysis



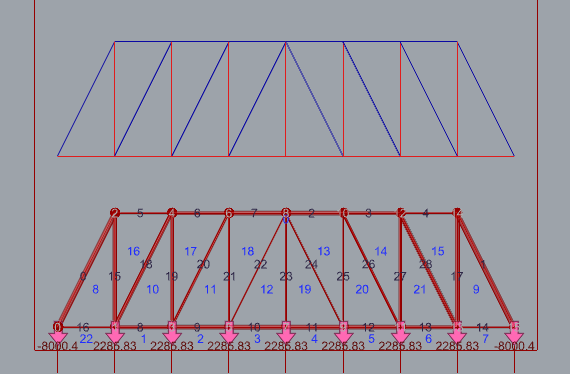
****

Figure 10: Howe Truss Form Diagram

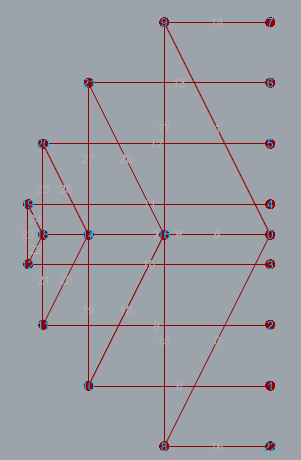
****

Figure 11: Howe Truss Force Diagram

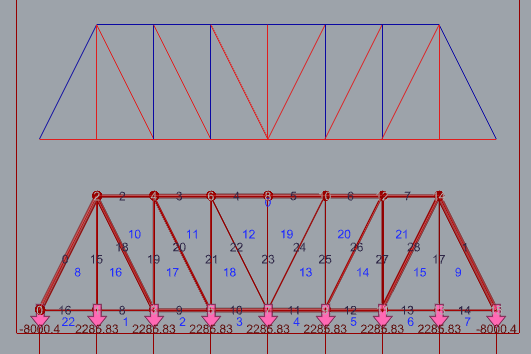


Figure 12: Pratt Truss Form Diagram

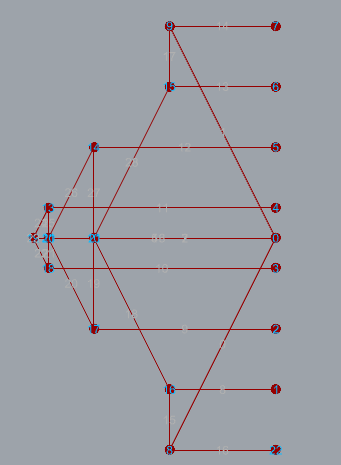


Figure 13: Pratt Truss Force Diagram

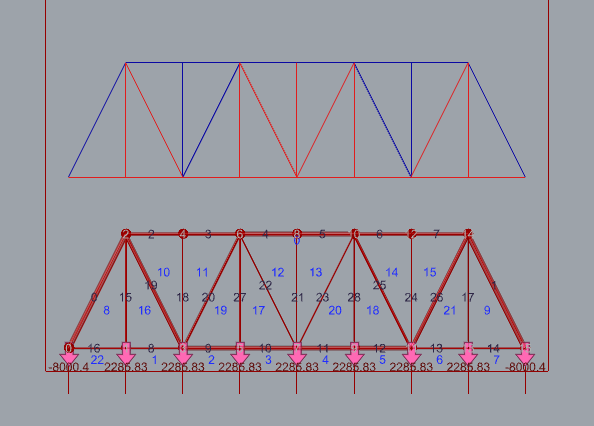


Figure 14: Warren Truss Form Diagram

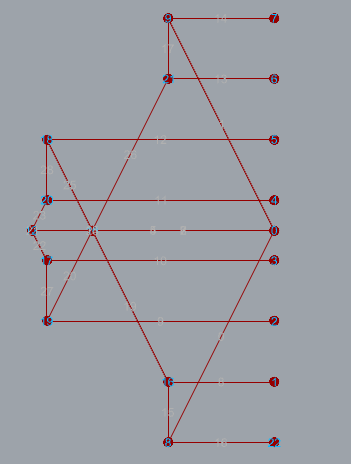
****

Figure 15: Warren Truss Force Diagram

## ANNEX II – MAJOR CLUSTER CODING IN Grasshopper PYTHON (ghPython)

**CLUSTER 1 –**

**CLUSTER NAME : MAIN SOLVER**

**import** rhinoscriptsyntax **as** rs

**import** Rhino.Geometry **as** rg

**import** scriptcontext **as** sc

#Creates a copy of an array.

**def copy\_func**(original):

copy = []

**print** original

**for** i **in range**(0,**len**(original)):

copy.**append**(original[i])

return copy

#load\_vec\_copy = copy\_func(lfl)

#MODULE 1

#list the end and start point of lines as per their indices

**def list\_endpoints**(a):

lleng = **len**(x) #get length of list

lfm = [[]] #form diagram line endpoints -2d null matrix

lfs=[] #form diagram slope lines

#Get End points of the lines in form diagram

**for** i **in range**(0, lleng): # LOOP

lfm.**append**([])

ls = rs.**CurveStartPoint**(x[i]) #get start point from force diagram

le = rs.**CurveEndPoint**(x[i]) # get end point from force diagram

lsp = rs.**coerce3dpoint**(ls) #convert guid to point

lep = rs.**coerce3dpoint**(rs.**AddPoint**(le)) #convert guid to point

lfm[i].**append**(lsp) #append to list of points

lfm[i].**append**(lep) #append to list of points

#get the slope of the line

**if abs**(lep[0] - lsp[0]) >= 0.01:

slope **=** (lsp[2]-lep[2])**/**(lsp[0]-lep[0])

lfs.**append**(**round**(slope,2))

**elif abs**(lep[0] - lsp[0]) <= 0.01:

lfs.**append**(321)

#list\_endpoints(1)

# MODULE 2

# Add length of each line in force diagram

#lfl = [] #length of each lines in force diagram

**def force\_list**(a):

rlen = **len**(fd\_line)

**for** i **in range**(0,rlen):

chk=0

**for** j **in range** (0, **len**(y)):

**if float**(i) == y[j]:

lfl.**append**(z[j])

chk =1

**if** chk==0:

lfl.**append**(321)

return lfl

#force\_list(1)

#print lfl

#Create List of points(dummy) for corresponding regions in force diagram. #Points are currently in y plane.

pt\_fd = [] # force diagram region co-ordinates.

**def dummy\_fd\_points**(a):

**for** i **in range**(0,**len**(regd)):

pt\_fd.**append**(rg.**Point3d**(0,1,0))

pt\_fd[start\_reg] =rs.**coerce3dpoint**(fdsp)

return pt\_fd

**dummy\_fd\_points**(1)

#Create a tracker for construction line work done.

tracker\_cline= [321]

# Create a list for lines in force diagram.

tracker\_ptsk = [10]

fdl = []

intersecting\_lines =[]

cline\_all=[[],[]]

#gets the clockwise direction of regions with external loads

**def clock**(r1,r2):

**for** i **in range**(0,**len**(rav)):

**for** j **in range**(1,**len**(rav[i])):

**if** rav[i][j-1] == r1:

**if** rav[i][j]== r2:

return "anti-clock"

**for** j **in range**(1,**len**(rav[i])):

**if** rav[i][j-1] == r2:

**if** rav[i][j]== r1:

return "clock"

#graphic = clock(2,10)

#print graphic

#print rav

# returns vector of intersection of regions provided

**def vbregion**(r1,r2):

curl = rs.**IntersectBreps**(act\_reg[r1],act\_reg[r2])

curlvec = rs.**VectorCreate**(rs.**CurveStartPoint**(curl),rs.**CurveEndPoint**(curl))

return curlvec

fdline\_data = []

# returns line index of intersection of regions provided

**def l\_index**(r1,r2):

curl = rs.**IntersectBreps**(act\_reg[r1],act\_reg[r2])

**for** k **in range**(0,**len**(fd\_line)):

**if** rs.**CurveMidPoint**(curl) == rs.**CurveMidPoint**(fd\_line[k]) :

line\_index = k

return line\_index

#get co-ordinate of lines with known forces & draw line.

**def getcord**(r1,r2):

**for** i **in range**(0,1):

curl\_index = **l\_index**(r1,r2)

curl\_vec = **vbregion**(r1,r2)

cur\_cord = pt\_fd[r1]

curl\_mag = lfl[curl\_index]

curl\_clock = **clock**(r1,r2)

**if** curl\_clock == "clock":

v\_clock = -1

**elif** curl\_clock =="anti-clock":

v\_clock= 1 # print curl\_vec

**if** lfl[curl\_index] != 321:

line =rg.**Line**(cur\_cord,curl\_vec \* v\_clock,curl\_mag)

line\_guid = sc.doc.Objects.**AddLine**(line)

cord = rs.**CurveEndPoint**(line\_guid)

pt\_fd[r2]=cord

#tracker\_ptsk.append(r2)

return cord

#getcord(10,2):

# Draw Cline or Fxdline -with coordinate from a point.

**def lines\_and\_pt**(curreg):

neigh\_reg =regd[curreg]

**for** i **in** neigh\_reg:

l\_cmon = **l\_index**(i,curreg)

**if** lfl[l\_cmon] !=321:

neigh\_cord = **getcord**(curreg,i)

pt\_fd[i] = neigh\_cord

**elif** lfl[l\_cmon] == 321:

r1= curreg

r2 = i

**if** pt\_fd[r2][1]==1:

curvec= **vbregion**(r1,r2)

curpt = pt\_fd[curreg]

curl = rg.**Line**(rs.**coerce3dpoint**(curpt),curvec,10000)

cline\_all[0].**append**(curl)

cline\_index = [r1,r2]

cline\_all[1].**append**(cline\_index)

tracker\_cline.**append**(curreg)

return cline\_all

**def knownpts**(zz):

cur\_reglist = []

**for** i **in range**(0,**len**(pt\_fd)):

**if** pt\_fd[i][1] ==0:

a=1

**if** i **in** tracker\_cline:

a=0

**if** a==1:

cur\_reglist.**append**(i)

return cur\_reglist

#cur\_reglist = knownpts(1)

#pick new regions to draw neighbouring or known points/ construction lines

**def draw\_new\_region**():

cur\_reglist = **knownpts**(1) #get known list from list of points

**for** j **in** cur\_reglist:

**lines\_and\_pt**(j)

#draw\_new\_region()

#find list of intersecting regions

**def intersecting\_clinelist**():

int\_index =[]

in\_index =[]

int\_cline =[]

int\_cline\_list =[]

**for** i **in range**(0,**len**(cline\_all[0])):

line1 = cline\_all[1][i][0]

line2 = cline\_all[1][i][1]

**for** j **in range**(i+1,**len**(cline\_all[0])):

**if** line1==cline\_all[1][j][0]:

index = [line2,cline\_all[1][j][1]]

in\_index.**append**(index)

int\_cline.**append**([i,j,line1])

**if** line1==cline\_all[1][j][1]:

index = [line2,cline\_all[1][j][0]]

in\_index.**append**(index)

int\_cline.**append**([i,j,line1])

**if** line2==cline\_all[1][j][0]:

index = [line1,cline\_all[1][j][1]]

in\_index.**append**(index)

int\_cline.**append**([i,j,line2])

**if** line2==cline\_all[1][j][1]:

index = [line1,cline\_all[1][j][0]]

in\_index.**append**(index)

int\_cline.**append**([i,j,line2])

**for** k **in range**(0,**len**(int\_cline)):

**if** rs.**IntersectBreps**(act\_reg[in\_index[k][0]],act\_reg[in\_index[k][1]]):

int\_index.**append**(in\_index[k])

int\_cline\_list.**append**(int\_cline[k])

return int\_cline

#find region point - coordinate by intersection of two cosecutive regions

**def pts\_intersection**(line1,line2,k):

a=cline\_all[0][line1]

b=cline\_all[0][line2]

pt\_intersection = rs.**LineLineIntersection**(a,b)

**if** pt\_intersection:

pt\_fd[k]=pt\_intersection[0]

return pt\_intersection

#find point and add to list after intersection

**def second\_type\_line**():

**for** i **in range** (0,20):

in\_list = **intersecting\_clinelist**()

**for** j **in** in\_list:

#print in\_list

pts\_intersect = **pts\_intersection**(j[0],j[1],j[2])

#in\_list = intersecting\_clinelist()

**draw\_new\_region**()

return pts\_intersect

# Create The force Diagram.

**if** RUN:

**lines\_and\_pt**(start\_reg) #get the system started!!!

**second\_type\_line**()

**CLUSTER 2 : GET LOADS AND OUT REGIONS**

**CLUSTER DETAIL : CREATE REGIONS AND ASSIGN UNIFORMLY DISTRIBUTED LOADS TO BOTTOM CHORD AND REACTION AT END POINTS.**

**import** rhinoscriptsyntax **as** rs

up\_reg =[0]

bot\_reg = []

**def top\_or\_bottom**():

max\_check = 0

a=0

**for** i **in range**(0,**len**(reg)):

z= rs.**SurfaceAreaCentroid**(reg[i])

zco = a\*z[0][0]

check = z[0][2] - zco

**if** check <0:

bot\_reg.**append**(i)

**if** check > 0:

**if** check > max\_check:

max\_check = check

up\_reg[0]=i

return 1

**top\_or\_bottom**()

fdl = []

**for** i **in range**(0,**len**(lines)):

fdl.**append**(321)

uload = span\*udl**/**(span\_no-1)

rxn = span\*udl/2

load\_line = []

**def assign\_load**():

**for** j **in range**(0,**len**(bot\_reg)):

**for** k **in range**(j,**len**(bot\_reg)):

curl = rs.**IntersectBreps**(reg[bot\_reg[j]],reg[bot\_reg[k]])

**if** curl:

**for** i **in range**(0,**len**(lines)):

**if** rs.**CurveMidPoint**(curl) == rs.**CurveMidPoint**(lines[i]):

fdl[i]=uload

load\_line.**append**(i)

**for** m **in range**(0,1):

**for** k **in** bot\_reg:

curl = rs.**IntersectBreps**(reg[k],reg[up\_reg[0]])

**if** curl:

**for** i **in range**(0,**len**(lines)):

**if** rs.**CurveMidPoint**(curl) == rs.**CurveMidPoint**(lines[i]):

fdl[i]= - rxn

load\_line.**append**(i)

**assign\_load**()

CLUSTER 3 : GET FORCES IN FORCE DIAGRAM

**import** rhinoscriptsyntax **as** rs

fdl\_len = []

fdl\_vtx = []

fdl\_line=[]

**for** i **in** tr\_line:

fdl\_len.**append**(0)

fdl\_vtx.**append**(0)

fdl\_line.**append**(0)

#Generate length and vertices of force diagram.

**def fdline\_vertex**() :

**for** j **in range**(0,**len**(reg)):

**for** k **in range**(0,**len**(reg)):

curl = rs.**IntersectBreps**(reg[j],reg[k])

**if** curl:

**for** i **in range**(0,**len**(tr\_line)):

**if** rs.**CurveMidPoint**(curl) == rs.**CurveMidPoint**(tr\_line[i]):

fdl\_len[i] = **round**(rs.**Distance**(pt\_fd[j],pt\_fd[k]))

fdl\_vtx[i]= [j,k]

fdl\_line[i]= rs.**AddLine**(pt\_fd[j],pt\_fd[k])

return fdl\_len

**fdline\_vertex**()

#get volumne of the line = lfd \* ltd

vol = 0

**for** i **in range**(0,**len**(tr\_line)):

vol = vol + rs.**CurveLength**(tr\_line[i])\*fdl\_len [i]

vol = vol/10000

**CLUSTER 4 : BIAS ADDITION**

**CLUSTER DETAIL : ADDITION OT BIAS TO ORIGINAL VOLUME**

**import** rhinoscriptsyntax **as** rs

#get volumne of the line = lfd \* ltd

vol = 0

j\_bias = joint\_bias

**for** i **in range**(0,**len**(tr\_line)):

**if** force[i] == 2:

b\_bias= buckling\_bias

**if** force[i] == 1:

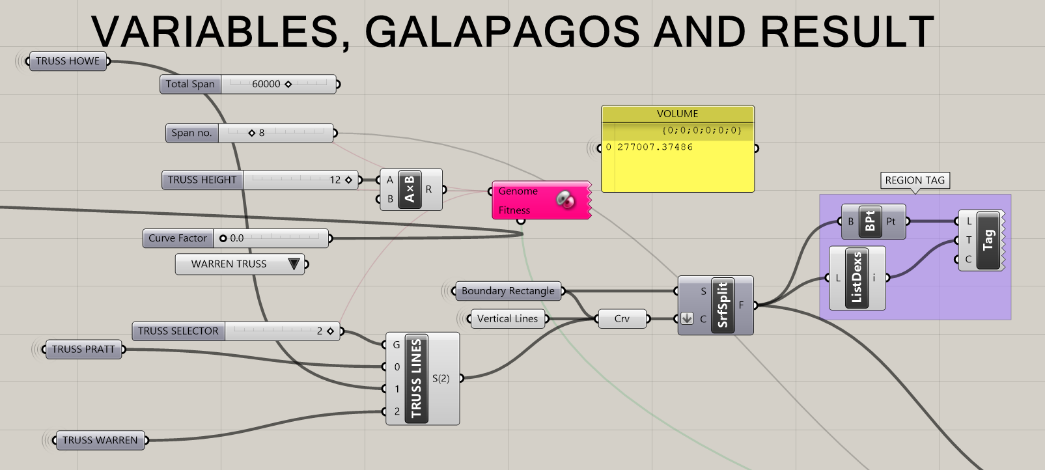
b\_bias = 1

vol = vol + rs.**CurveLength**(tr\_line[i])\*fdl\_len[i]**\*** (1**+**(1-b\_bias)+j\_bias)

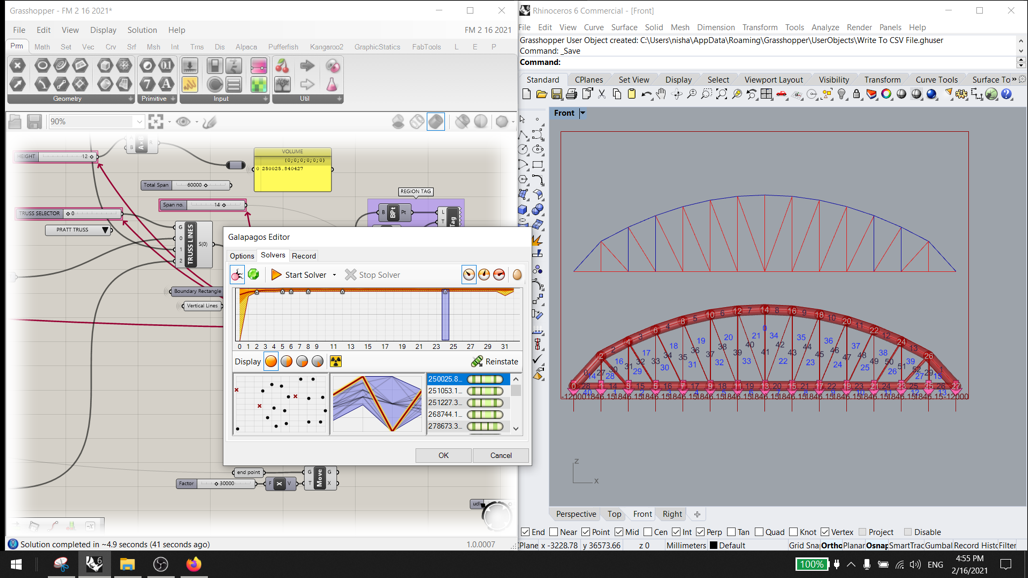
vol = vol/10000

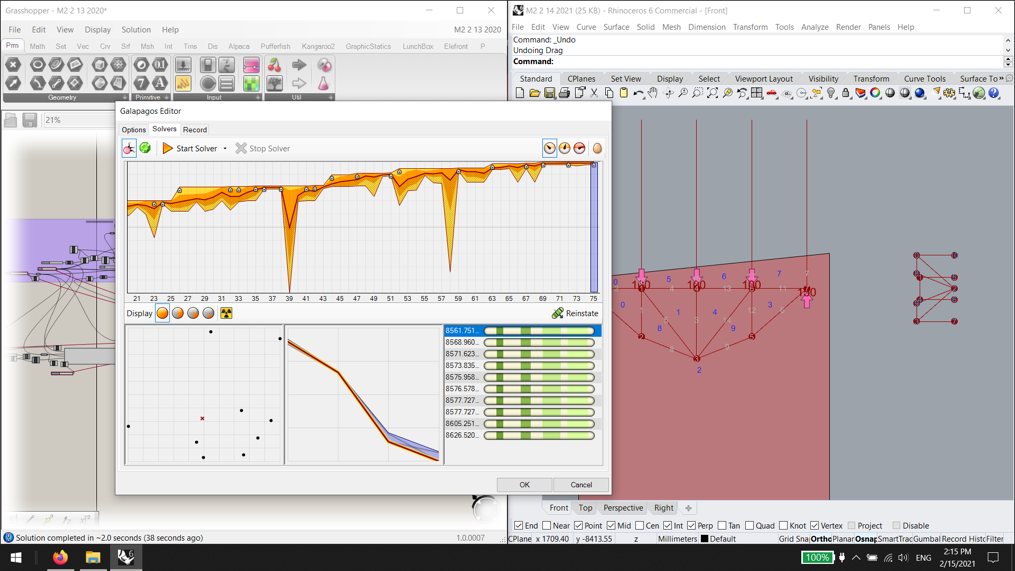
**print** vol

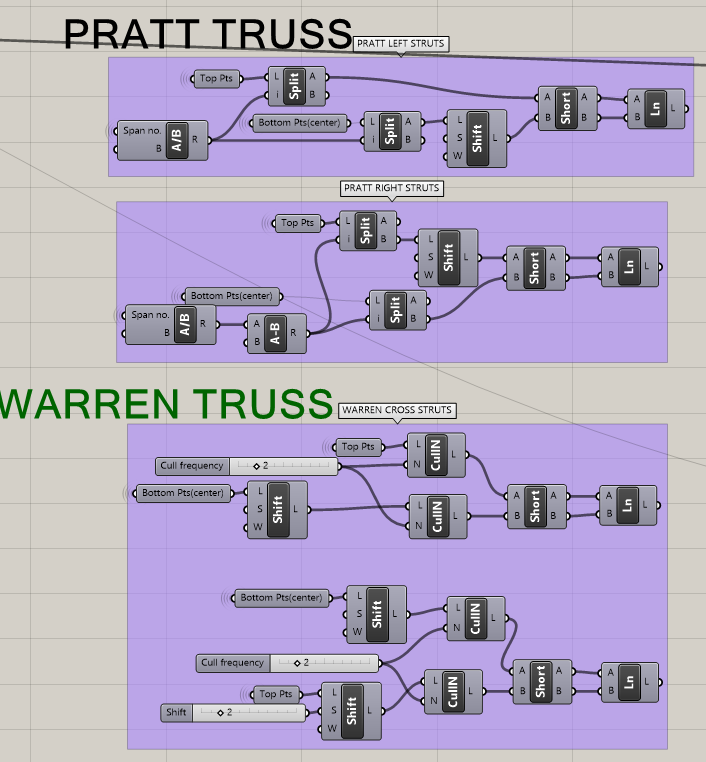
## ANNEX III: Snapshots

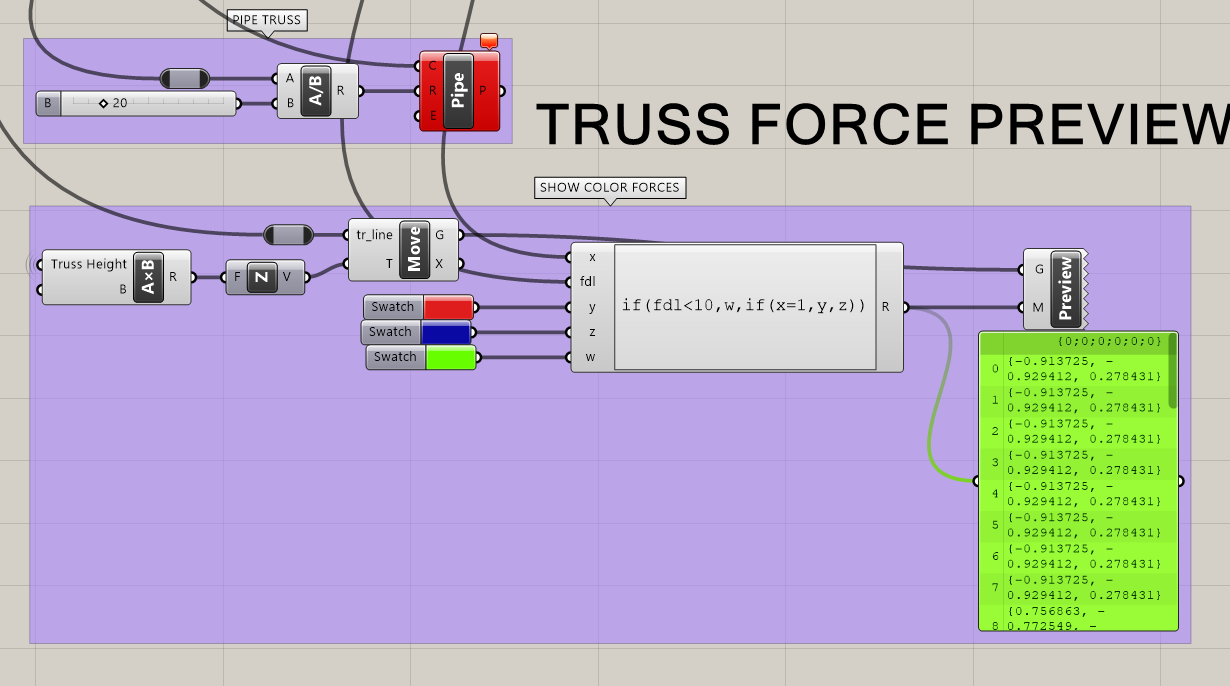
****

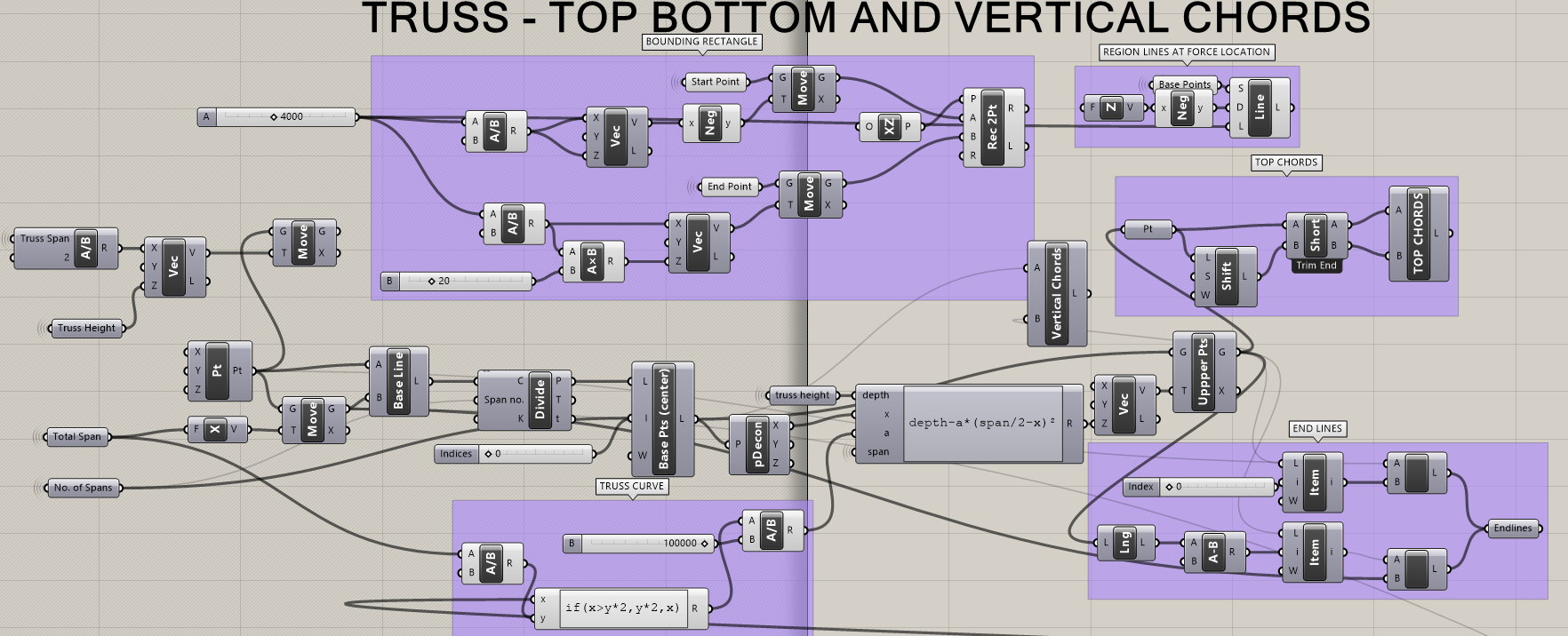
****

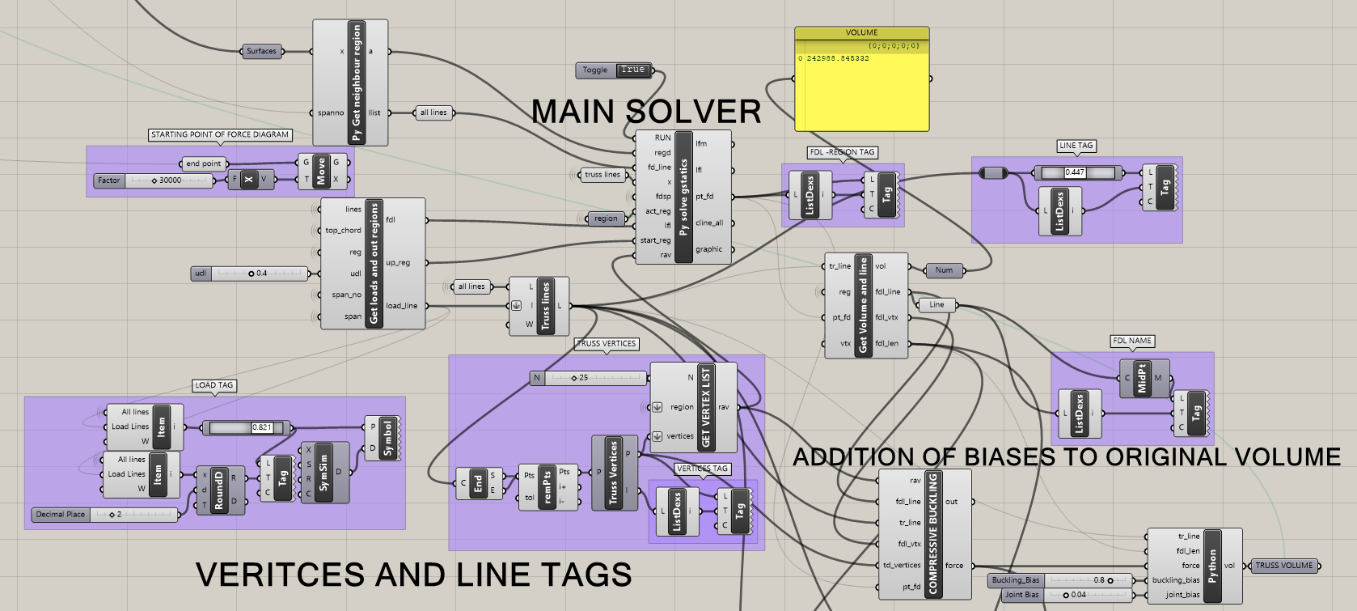
****

****

****

****

****

****