

**KARADENİZ TECHNICAL UNIVERSITY
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES**

DEPARTMENT OF CIVIL ENGINEERING

**INTELLIGENT OPTIMIZATION OF STEEL STRUCTURES THROUGH BIM-BASED VISUAL
PROGRAMMING PLATFORM AND TOOLS**

MASTER THESIS

Feyzullah YAVAN

**OCTOBER 2023
TRABZON**



**KARADENİZ TECHNICAL UNIVERSITY
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES**

DEPARTMENT OF CIVIL ENGINEERING

**INTELLIGENT OPTIMIZATION OF STEEL STRUCTURES THROUGH BIM-BASED
VISUAL PROGRAMMING PLATFORM AND TOOLS**

Feyzullah YAVAN

**This thesis is accepted to give the degree of
"MASTER OF SCIENCE"**

**By
The Graduate School of Natural and Applied Sciences at
Karadeniz Technical University**

The Date of Submission : 04 / 10 / 2023

The Date of Examination : 23 / 10 / 2023

Supervisor : Prof. Dr. Vedat TOĞAN

Co-Supervisor : Prof. Dr. Reza Maalek

Trabzon 2023

ACKNOWLEDGEMENT

I would like to start my thesis by expressing my gratitude to my thesis supervisors, Prof. Dr. Vedat Toğan from Karadeniz Technical University (KTU) for guiding the thesis process and providing new perspectives as well as rapid feedback throughout the process, and Prof. Dr. Reza Maalek from Karlsruhe Institute of Technology (KIT) for encouraging me to work in this subject and introducing me to parametric design and the endless possibilities that come with visual programming and Dynamo. Working under their supervision was very rewarding in various aspects that are not limited to only this thesis subject and their assistance, counsel, and constructive criticism throughout my thesis journey are the main reasons for the success of this thesis. Their willingness to mentor me is something I sincerely appreciate.

Furthermore, a big thanks to Karlsruhe Institute of Technology and the Institute of Technology and Management in Construction for the opportunities that they provide for my research.

Also deserving of praise are the members of the Dynamo forum who are constantly trying to help and respond to new questions and debate other options.

Lastly and most importantly, I want to extend my most profound appreciation to my entire family for their constant support, parental advice, encouragement, and incomparable love not only during my studies but also throughout my whole life. Their belief in me has been an endless source of motivation, and I am truly blessed to have them by my side.

Feyzullah YAVAN

Trabzon 2023

STATEMENT OF ETHICS FOR THESIS

I hereby declare that I have completed the study titled "Intelligent Optimization of Steel Structures through BIM-based Visual Programming Platform and Tools," which I submitted as my Master's Thesis, under the supervision of my advisor Prof. Dr. Vedat TOĞAN and co-advisor Prof. Dr. Reza MAALEK. I confirm that I collected the data myself, conducted the analyses, accurately cited all information obtained from other sources both within the text and in the references section, adhered to scientific research and ethical principles throughout the study process, and accept any legal consequences that may arise if the contrary is proven. 23/10/2023

Feyzullah YAVAN

TABLE OF CONTENTS

	<u>Page Number</u>
ACKNOWLEDGEMENT	iii
STATEMENT OF ETHICS FOR THESIS	iv
TABLE OF CONTENTS	v
SUMMARY	viii
ÖZET	ix
LIST OF FIGURES	x
LIST OF TABLES	xiii
LIST OF SYMBOLS AND ABBREVIATIONS	xiv
1. INTRODUCTION	1
1.1. Problem Statement	3
1.2. Research Objective and Scope	4
1.3. Research Methodology	5
1.4. Outline of the Thesis	5
2. BACKGROUND	6
2.1. Computational Design	8
2.2. Generative Design	9
2.3. Parametric modeling	10
2.4. Structural Optimization Types and Methodologies	12
2.4.1. Single-Objective Optimization	12
2.4.2. Multi-Objective Optimization	13
2.4.3. Brute Force Method	15
2.4.4. Shape Optimization	15
2.4.5. Size Optimization	16
2.4.6. Topology Optimization	17
2.4.7. Objective Function	19
2.4.8. Penalty Function	20
2.5. Meta-Heuristics Algorithms	22
2.5.1. Genetic Optimization	23
2.5.2. Ant Colony Optimization	25
2.5.3. Particle Swarm Optimization Algorithm	26

2.6. Life Cycle Assessment.....	26
2.7. Tools	27
2.7.1. Autodesk Revit	27
2.7.2. Tally plug-in for Revit.....	28
2.7.3. Autodesk Robot Structural Analysis	28
2.7.4. Dynamo	29
2.7.5. Visual Programming.....	29
2.7.6. Utilized Dynamo Packages	33
2.7.7. Python.....	34
3. METHODOLOGY	37
3.1. Prepare model on Dynamo	38
3.1.1. Step 1: Input Parameters.....	38
3.1.2. Step 2: Generation of the Model Geometry in Dynamo	39
3.2. Perform Calculation and Retrieve Results.....	42
3.2.1. Step 3: Analytical Model Setup.....	42
3.2.2. Step 4: FEM Analysis with the Help of RSA.....	46
3.3. Evaluate the Results.....	46
3.3.1. Step 5: Collecting Results from RSA and Calculation of Weight Score	46
3.3.2. Step 6: Colorized Visualization of the Wire Model Created on Dynamo...	50
3.4. Structural Optimization.....	52
3.4.1. Core Node.....	53
3.4.2. Creation of Initial Population	53
3.4.3. Creating The Parents and Getting the Results.....	57
3.4.4. Selection Operation	64
3.5. Revit Importer.....	66
3.5.1. Importing the Best Result to Revit.....	66
3.5.2. Assembling the Connection Details	69
3.5.3. Perform LCA and Cost Analysis.....	71
4. EXAMPLES AND RESULTS	79
4.1. 10-Bar 2D Truss.....	80
4.2. 36-Bar 3D Truss.....	84
4.3. 120-Bar Dome Truss.....	88
4.3.1. LCA and Cost Analysis for 120-Bar Truss	96
5. DISCUSSION.....	98

6. CONCLUSION.....	100
7. SUGGESTION	101
8. REFERENCES	102
9. APPENDICES	110
AUTOBIOGRAPHY.....	149



SUMMARY

INTELLIGENT OPTIMIZATION OF STEEL STRUCTURES THROUGH BIM-BASED
VISUAL PROGRAMMING PLATFORM AND TOOLS

Feyzullah YAVAN

Karadeniz Technical University
The Graduate School of Natural and Applied Science
Civil Engineering Graduate Program
Supervisor: Prof. Dr. Vedat TOĞAN
Co-Supervisor: Prof. Dr. Reza MAALEK
2023, 110 Pages, 54 Pages Appendix

Since our planet doesn't have unlimited resources, the importance of using the existing resources carefully becomes increasingly evident over time. That is why this thesis investigates the optimization of structures using nature-inspired algorithms - specifically focusing on Genetic Algorithm (GA)- by utilizing Visual Programming (VP) and Building Information Modelling (BIM) tools to get minimum-weighted structure while satisfying stress and displacement constraints. While GAs have been extensively studied in the past, the integration of GAs with VP has remained relatively unexplored. Thus, this research also aims to make an integration between them to use the capabilities of Dynamo, Revit, and RSA which will provide the best possible design outcome.

The research methodology involves employing Dynamo to create parametric trusses, calculating their fitness values by using Finite Element Analysis (FEA) results, choosing the best solution based on fitness values, and creating a new population using GA. For FEA, RSA software has been employed to get stresses, displacements, and the weight of the structure. To further enhance the proposed methodology and demonstrate the advantages of BIM, the optimized model is imported into Revit, enabling the user to analyze aspects such as Life Cycle Assessment (LCA), cost details, etc.

The outcomes of this thesis offer a valuable understanding of combining VP, GAs, and BIM environments for optimization purposes. The findings not only demonstrate the feasibility of combining these tools in the real world but also emphasize that this integration provides the opportunity to increase both structural efficiency and sustainability factors.

Key Words: Structural Optimization, Genetic Algorithm, Parametric Modelling, Visual Programming, Dynamo, Revit, Robot Structural Analysis

ÖZET

ÇELİK YAPILARIN BIM TABANLI GÖRSEL PROGRAMLAMA PLATFORMU VE
ARAÇLARI İLE AKILLI OPTİMİZASYONU

Feyzullah YAVAN

Karadeniz Teknik Üniversitesi
Fen Bilimleri Enstitüsü
İnşaat Mühendisliği Anabilim Dalı
Danışman: Prof. Dr. Vedat TOĞAN
Eş Danışman: Prof. Dr. Reza MAALEK
2023, 110 Sayfa, 54 Sayfa Ek

Gezegeneğimiz sınırsız kaynaklara sahip olmadığından, mevcut kaynakları dikkatli bir şekilde kullanmanın önemi zaman içinde giderek daha belirgin hale gelmektedir. Bu nedenle bu tez, Görsel Programlama (VP), Yapı Bilgi Modellemesi (BIM) ve Sonlu Elemanlar Analizi (FEA) kullanılarak gerilme ve yer değiştirme kısıtlamalarını sağlayan minimum ağırlıklı yapıyı Genetik Algoritma kullanarak elde etmeyi amaçlamaktadır. Bu amaçla anılan programların aralarındaki entegrasyon VP tabanlı kodlamalar ile sağlanmıştır. Araştırma yöntemi, farklı girdi setleriyle parametrik kafes sistemler oluşturmak için Dynamo'nun kullanılmasını, FEA sonuçlarını kullanarak her bir oluşturulan yapının ağırlık skorunu hesaplamayı, ağırlık skorlarına dayalı olarak en iyi çözümü seçmeyi ve metaheuristik algoritmalar kullanarak yeni bir popülasyon oluşturmayı içermektedir. Bu tezde Genetik Algoritma (GA) operatörleri, en iyi yapısal çözümleri bulmak için görsel programlama çerçevesi içinde uygulanmaktadır. FEA yöntemi, gerilmelerin, yer değiştirmelerin ve yapının ağırlığının elde edilmesi için RSA yazılımı aracılığıyla kullanılmıştır. Önerilen yöntemi daha da geliştirmek ve BIM teknolojisinin avantajlarını göstermek amacıyla, optimizasyon sonucunda elde edilen son model Revit'e aktarılmış ve bu sayede Yaşam Döngüsü (LCA) ve maliyet gibi analizlerin yapılması mümkün olmuştur.. Bu entegrasyon, yapısal verimlilik ve sürdürülebilirlik faktörlerini bir arada değerlendiren kapsamlı bir değerlendirme imkanı sunmaktadır. Bu tezin sonuçları, GA'ların, VP'nin ve BIM ortamlarının optimizasyon amaçları için birleştirilmesinin olasılığına dair değerli bir anlayış sunmaktadır. Bulgular, yalnızca bu araçların birleştirilmesinin gerçek dünyada uygulanabilir olduğunu göstermekle kalmayıp, aynı zamanda bu entegrasyonun optimize edilmiş model üzerinde LCA ve maliyet analizi gibi yönleri ele alma imkanı sunduğunu da vurgular.

Anahtar Kelimeler: Yapısal Optimizasyon, Genetik Algoritma, Parametrik Modelleme, Görsel Programlama, Dynamo, Revit, Robot Structural Analysis

LIST OF FIGURES

	<u>Page Number</u>
Figure 1. Simple generative design scheme	9
Figure 2. Inputs for parametric design	10
Figure 3. Example of parametric design.....	11
Figure 4. A hypothetical decision-making chart (adapted from (Deb & Deb, 2014))	14
Figure 5. Shape optimization.....	16
Figure 6. Size optimization.....	17
Figure 7. Topology optimization example for a curved column	17
Figure 8. Topology optimization	18
Figure 9. QNCC (URL-9, 2023).....	18
Figure 10. Example of visual programming.....	30
Figure 11. Example of textual programming.....	30
Figure 12. Example of created building on Revit by Dynamo.....	31
Figure 13. Rotated ellipses for the Absolute Tower.....	31
Figure 14. Example of wall maker nodes	32
Figure 15. The final version of the model on Revit.....	32
Figure 16. Example of a custom node	34
Figure 17. Example of Python usage in Dynamo	35
Figure 18. General overview of the Dynamo script	38
Figure 19. Input parameters for the truss.....	39
Figure 20. Geometries explained in this section.....	39
Figure 21. Usage of the Dynamo nodes	40
Figure 22. Nodes inside the "x bracing" custom node	40
Figure 23. Creating the adaptive bracings	41
Figure 24. Truss type input.....	41
Figure 25. Nodes to create the geometry lines of the truss.....	42
Figure 26. Assigning the sections on Dynamo	43
Figure 27. Creating the bars, sections, and materials on RSA	43
Figure 28. Assigning the bar-end releases.....	44
Figure 29. Definition of the Supports and Releases in RSA	44
Figure 30. Defining the supports on Dynamo	45

Figure 31. Definition of the loads for RSA	45
Figure 32. RSA analysis on Dynamo	46
Figure 33. Extracting the stresses from the analysis	47
Figure 34. Changing units and grouping the stresses	47
Figure 35. Extracting the deformations from the analysis	48
Figure 36. Extracting the self-weight from the analysis.....	48
Figure 37. Python script used for evaluating and ranking the results.....	50
Figure 38. Necessary nodes to visualize the model on Dynamo	51
Figure 39. The truss after visualization	51
Figure 40. Visualization of trusses created based on inputs.....	52
Figure 41. The core node of optimization	53
Figure 42. Python script to create an initial population.....	54
Figure 43. Initial population's fitness calculator.....	55
Figure 44. The “LoopCheck” node and explanation	55
Figure 45. Content of the “LoopBodyVar1” custom node.....	56
Figure 46. Iteration-Based Input Retrieval	56
Figure 47. Fitness result of the initial population.....	57
Figure 48. Nodes to run the GA operators	57
Figure 49. Nodes that “LoopBodyVar2” contains.....	58
Figure 50. Next population generator.....	58
Figure 51. The Python code of main operators	60
Figure 52. The results of the generation node	64
Figure 53. Offspring list and the selection operator	65
Figure 54. Final results of the optimization.....	65
Figure 55. Custom node for importing the structure to Revit	66
Figure 56. Learning the family name of the cross-sections.....	67
Figure 57. Python script that chooses the best individual and gives its parameter	67
Figure 58. Calling the families by the string	68
Figure 59. Creating the elements on Revit	68
Figure 60. The Revit model of optimized structure.....	69
Figure 61. Truss model after importing to the Revit	69
Figure 62. Nodes for adding the connection details	70
Figure 63. (b) The structure after adding the connection details.....	71
Figure 64. Tally add-ins in the Revit.....	71

Figure 65. Type of the study for the LCA	72
Figure 66. The object of study	72
Figure 67. The project browser	72
Figure 68. Tally database.....	73
Figure 69. Defining the components and quantity takeoffs.....	73
Figure 70. Table of Contents of the LCA.....	74
Figure 71. Bar graphs showing mass and impact category breakdowns	74
Figure 72. Pie chart of the global warming potential	75
Figure 73. A section from the appendix of the report	76
Figure 74. Schedule properties	77
Figure 75. Adding calculation parameter	77
Figure 76. The bill of quantities table in the Revit.....	78
Figure 77. 10-bar truss.....	80
Figure 78. Weight score change for 10-bar truss by generation number	83
Figure 79. The final version of the 10-bar truss	83
Figure 80. 36-bar truss.....	84
Figure 81. The script for creating the 36-bar truss	84
Figure 82. Weight score change for 36-bar truss by generation number	87
Figure 83. The final version of the 36-bar truss	87
Figure 84. 120-bar dome truss configuration	88
Figure 85. 120-bar dome truss loading conditions	89
Figure 86. Calculation of slenderness for buckling.....	91
Figure 87. Python code for critical stress	92
Figure 88. Critical stresses after calculations	92
Figure 89. The script for creating a 120-bar dome truss	93
Figure 90. Updated weight score calculator for buckling	93
Figure 91. Weight score change for 120-bar truss by generation number	95
Figure 92. The final version of the 120-bar dome truss	95
Figure 93. 120 bar truss on Revit	96
Figure 94. Summary of LCA.....	96

LIST OF TABLES

	<u>Page Number</u>
Table 1. Overview of Python scripts written throughout the thesis.	36
Table 2. Optimization processes.....	37
Table 3. Python inputs and explanations for the weight score calculation.....	50
Table 4. Python inputs and explanations for the initial population creator	54
Table 5. Python inputs and explanations for the GA generator.....	61
Table 6. Python inputs and explanations for the Selection node.....	65
Table 7. Information about the design problems.....	79
Table 8. Device specifications.....	79
Table 9. Discrete cross-sectional area list and names	81
Table 10. Comparison of 10-bar truss results.....	82
Table 11. Discrete cross-sectional area list and names	85
Table 12. Comparison of 36-bar truss results.....	86
Table 13. Discrete cross-sectional area list and names	90
Table 14. Grouping of 120-bar dome members	90
Table 15. Comparison of 120-bar dome truss results.....	94
Table 16. Results per Life Cycle Stage	97
Table 17. Revit bills of quantity table	97

LIST OF SYMBOLS AND ABBREVIATIONS

Symbol	Definition
ACO	: Ant Colony Optimization
AEC	: Architectural Engineering and Construction
AISC	: American Institute of Steel Construction
BIM	: Building Information Modelling
CAD	: Computer-Aided Design
CO ₂	: Carbon Dioxide
CHS	: Circular Hollow Section
CSI	: Construction Specifications Institute
EAs	: Evolutionary Algorithms
EPDs	: Environmental Product Declarations
FEA	: Finite Element Analysis
GA	: Genetic Algorithm
GPT	: Generative Pre-trained Transformer
HSRO	: Hollow Structural Round Sections
LCA	: Life Cycle Analysis
LEED	: Leadership in Energy and Environmental Design
LCH	: Life Cycle Handprint
MOOP	: Multi-Objective Optimization Problem
PSO	: Particle Swarm Optimization
QNCC	: Qatar National Convention Centre
RB	: Round Bars
RSA	: Robot Structural Analysis
SOO	: Single Objective Optimization
TSP	: Traveling Salesman Problem
TO	: Topology Optimization
VP	: Visual Programming

1. INTRODUCTION

While computing technologies improve with time, the optimization concept is gaining value with it particularly in the construction field because it ensures the efficient use of resources, enhances structural integrity, and maximizes overall performance. The current project designing workflow in the Architecture, Engineering, and Construction (AEC) sector is divided between architects and engineers, often resulting in compromises or inefficient designs. For engineers, structural optimization in the design process is a logical step because typically, engineering design aims to minimize the use of natural resources and keep costs low while satisfying the design constraints (Beghini et al., 2014) while architects are more concerned about aesthetics. Besides, parametric modeling is increasingly used by architects for design flexibility, but these models are typically not used for engineering analysis (Davis & Peters, 2013) because, even though there are various structural analysis tools available for engineers, they often require non-parametric models and lack integration with architectural branches. Making a connection between these software and parametric models requires significant technical knowledge and programming skills from the user (Vermeulen, 2018). To achieve these problems and leave the repetitive tasks to computers, in this thesis optimization concept has been extensively studied and a method has been proposed to optimize structures by using new technologies that will reduce costs, minimize errors, explore a larger solution space, in other words help to find the “fittest” solution for both disciplines by bringing together these different areas in the early design phase which easy and more efficient to make changes. Moreover, the workflow will be simple to use for users on the contrary of other created scripts despite it includes BIM, visual programming, textual programming, and parametric modeling techniques. Also, since updating the projects is less costly in the early design phase and using minimum material has a direct effect on the cost, implementation of the proposed optimization technique in the industry should be easier because, in the present industry, companies tend to prioritize the implementation of automated workflows only if there is a clear financial advantage for the company, rather than solely for the benefit of the project (Holzer, 2011). Moreover, because of the enormous amount of CO₂ emissions in the civil engineering business, decreasing environmental impacts has recently been another key goal of the governments due to the increased attention on environmental issues and

sustainable development (Choi et al., 2017). To address these problems, the suggested workflow includes LCA and cost analysis by utilizing BIM tools such as Revit and Tally to examine the environmental and economic effects of optimized designs, supporting sustainability and cost-effectiveness in construction projects.

To explain why using BIM tools is important for optimization purposes, it is necessary to first understand the BIM concept. BIM is a technique that facilitates the design and construction of structures within a single 3D model for AEC professionals, real estate developers, manufacturers, and other construction workers by creating and managing data throughout the design, building, and operating processes from beginning to end of the project by using software such as Revit, Dynamo, Navisworks, and ArchiCAD, etc. Briefly, BIM creates a digital twin of ongoing projects to provide elaborate digital depictions that are controlled in a cloud-based environment for immediate collaborative work and communication. Due to the clear and exact representations supplied by BIM, the project is easier to observe, understand, and update which aids in enhanced choices, greater environmental consciousness, and increased efficiency and cost-effectiveness in construction projects (URL-1, 2023). For this reason, this thesis also focuses on the integration of BIM and Meta-heuristic search algorithms, especially GA, to optimize designs of the AEC sectors. This integration will improve the project-creating process by generating multiple optimized design alternatives by using Meta-heuristic optimization algorithms and providing detailed reports about the project by using BIM tools. Consequently, architects and engineers can give more time to different aspects of a project and encourage innovation in their field by utilizing automation and streamlining repetitive tasks. Additionally, by using optimization, less weighed and more functional designs can be constructed, which have a vital role for countries that have limited sources, resulting in reduced expenses and embodied CO₂ (Daloğlu & Armutçu, 1998).

The reason why intelligent metaheuristic algorithms are used for this thesis even though there are numerous optimization types is because of their flexibility and ability to handle complicated optimization problems compared with conventional optimization methods, inclusive of the Brute Force approach (Machairas et al., 2014), which has been investigated in section 2.4.3. To deal with this assignment, in recent times a growing consciousness has been directed towards using algorithms stimulated by using nature, consisting of GAs that mimic the evolutionary methods that occur in nature (Goldberg, 1989), or Ant Colony optimization which mimics the ant's conduct to reach the food. Even

though metaheuristic algorithms had been significantly studied in the past, their integration with VP, BIM, and adaptive modeling strategies remains noticeably unexplored as mentioned before. The next sections introduce the problem statement, scope, objective, methodology, and outlines of the thesis to guide the reader.

1.1. Problem Statement

For the reasons stated above, there has been an increasing demand for structures that are both optimized and efficient in their performance. This growing trend is a challenge for many businesses to remain competitive and keep up with the changing market. It is crucial to include new technology and integrate structural optimization strategies in the design phase due to the fact about 70% of the overall cost of a structure is determined during the design phase (Toğan, 2004). Although the concept of structural optimization using metaheuristic algorithms is not new, it is not widely used in today's projects. The main reasons for this are the complexity of integrating it into the project creation process, specifically for engineers who lack experience in optimization and programming, and the need for new software programs or algorithms that are developed outside the BIM environment (Vermeulen, 2018). Also, there are proposed scripts by Autodesk to optimize structures by using Dynamo packages however, in the course of time, some packages grow old and exceed the point of life support so they don't work in updated Dynamo or for RSA anymore and cause lots of problems for those who want to use BIM tools for structural analysis.

These reasons brought the question about the thesis aim as “Can a visual programming script, which is user friendly and efficient, be prepared on Dynamo to create various design options using different metaheuristic algorithms and perform FEM analysis using RSA to evaluate fitness values and rank the design options for optimization purposes then choose best of them and transfer it on Revit for LCA, cost and project management purposes?” After this question, in this thesis an alternative script, which is easier to understand for engineers who are new to the concept of visual programming or textual programming, has been presented to optimize structures by utilizing software tools like Dynamo, Revit, and Robot Structural Analysis, which are part of the BIM approach, while decreasing dependency on dynamo packages to prevent possible errors and accelerating the optimization process, leading to more efficient structural design.

1.2. Research Objective and Scope

This thesis primarily aims to provide an easy way to optimize structures by utilizing BIM tools (Dynamo, RSA, etc.) and offering engineers a streamlined workflow for various types of optimizations and algorithms by simply modifying the Python script on Dynamo. After obtaining the "fittest" structure, the corresponding structural model is automatically generated in Revit using Dynamo to utilize BIM's advantages like achieving the cost and LCA details of the optimized structure.

The created genetic algorithm and codes are intended to demonstrate the applicability of the Python script in Dynamo, test the effectiveness of the script, and showcase the integration of nowadays technologies like Generative Pre-trained Transformer (GPT) in civil engineering workflows. ChatGPT, a groundbreaking advancement that entered our lives at the beginning of this year, possesses exceptional talent in transforming human ideas into code and serves as a bridge between scientists and programming. To showcase these advantages and encourage young scientists to utilize these new technologies, ChatGPT has been employed as an additional tool in the process of creating the code used in this thesis, even though the code was reviewed and the final version was created by the author. Thus, the aim is to highlight another aspect of language models and how beneficial and important they can be as assistants during software development or the creation of something new. The significance of adopting technological advancements like AI in the construction industry is evident, and it holds the potential to reshape entire industries. It is necessary to mention that the objective of the study is not to develop the best genetic algorithm and the most efficient code or improve the existing optimization algorithms due to limitations and time constraints. Although better algorithms and codes can be developed and readily implemented into the proposed script, these issues are beyond the scope of the present research. Also because of time limitation multi-objective optimization has not been performed with the proposed script but it can also be implemented to the script in future works. The structures analyzed consist of 2D and 3D truss structures. Complex structures which are created by more elements and Frame structures are not included in this study due to limitations. To sum up, a Single-Objective GA Solver was developed using Python, visual programming, and BIM tools. The effectiveness of the proposed methodology was validated through optimization benchmark problems from open literature.

1.3. Research Methodology

A workflow has been developed to answer and solve optimization problems by using BIM tools in this study. The work is carried out by developing the optimization script on Dynamo while supporting the script with python codes. The provided script enables users to add and use different types of algorithms for optimization. After the optimization with the help of Dynamo, the “fittest” structure is sent to Revit and created on Revit to get LCA, embodied CO₂, and cost details. The data transfer between Dynamo and Robot structural analysis is performed by a custom node called “Structural Analysis for Dynamo” which detailed information can be found in section 0.

1.4. Outline of the Thesis

The structured outline of the thesis can be found in the following section:

- Section 1: The introduction provides an informal summary, highlighting the problem statement, explaining the motivation for the study, and presenting the research methods.
- Section 2: Background provides important information about literature and tools that have been used during the thesis. First, it starts with a brief overview of the present development of structural optimization types and methodologies, computational design techniques, and the LCA concept then it finishes with the explanations of tools and software used during the thesis.
- Section 3: The methodology explains the methods used, including the codes, and describes how to build the script proposed in this thesis for the readers.
- Section 4: Experimental results show the examples and their results that are used to validate the proposed methodology. It includes three benchmark problems which are taken from the literature and comparison of the results.
- Section 5: The discussion section is the section where the results obtained from the thesis are commented on.
- Section 6: Conclusions is the section where the summary and outcomes of the thesis can be found.
- Section 7: The suggestion section outlines potential areas for further search or ideas to improve the current research that has been done during the thesis.

2. BACKGROUND

The aim of designers is to find the best solution or outcome among various possible choices while taking into account certain limitations. In that regard, many optimization algorithms have been developed. Most of these developed algorithms use nature as their main source of theory and attempt to mimic nature. Simulated annealing (SA), the first example of such algorithms that mimics the temperature change of metals, was developed by Kirkpatrick et al. in 1984 (Kirkpatrick, 1984). Then JH Holland (Holland, 1992) developed a genetic algorithm (GA) that is inspired by evolutionary theory. Also, GA has been utilized to reach optimum solutions for examples in this thesis too. Dorigo et al. (Dorigo et al., 2006) developed ant colony optimization (ACO), which mimics the behavior of ants. Kennedy et al. (Kennedy & Eberhart, 1995) proposed particle swarm optimization (PSO), a quite strong metaheuristic algorithm that mimics swarm behavior. Over time, many other optimization techniques have been developed. Further information can be found in the referenced review article prepared by Dede et al. (Dede et al., 2019). Among these algorithms, GA, have been extensively detailed due to its usage in this thesis, while others such as PSO and ACO are briefly discussed to offer varied examples of metaheuristic algorithms in the upcoming sections.

On the other hand, researchers focused on structural optimization with the aim of achieving optimum performance from a structure while lowering costs and material usage. Schmit (Schmit, 2012) was one of the first to harness the power of computers for structural optimization. Then, in 1991, Saka and Ulker (Saka & Ulker, 1991) implemented GA into the optimization methodology and created an algorithm to optimize space truss structures while taking into consideration of nonlinear behavior of structures. Rajeev and Krishnamoorthy (Rajeev & Krishnamoorthy, 1992) presented a methodology to transform constrained problems into unconstrained ones and used discrete sets of solutions for problems to show the efficiency of GAs in handling discrete variables. Groenwold et al. (Groenwold et al., 1999) also used discrete optimization and added slenderness and buckling to the design problems and solved them with GA in 1999. Toğan and Daloğlu (Toğan & Daloğlu, 2008) adopted the member grouping strategy while using GA and proposed a methodology to set the initial population to reach the optimum solution in structural optimization earlier. Although these studies demonstrate the implementation of

metaheuristic algorithms in the structural optimization process, scholars used their own programs to perform the optimization process. This approach caused inefficiency when there was a need to optimize a new structure or make it more widely accessible among designers. To address this challenge, in 2016, Cicconi et al. (Cicconi et al., 2016) employed SAP2000 to conduct FEM analysis during the optimization process with a specific focus on optimizing large steel frame structures, particularly oil & gas modules, to reduce costs and weight. Additionally, Kaveh et al. (Kaveh et al., 2014) employed SAP2000 via an open application programming interface (OAPI) for the shape and size optimization of large-scale barrel vault frames. Similarly, Artar and Carbas (Artar & Carbas, 2021) established integration between Matlab and SAP2000 using OAPI to optimize steel truss bridges with the aim of minimizing weight. Another example related to this subject is the one conducted by Mirniazmandan et al. (Mirniazmandan et al., 2018) in 2018. In this study, the scholars used Rhino and Grasshopper to generate models and Karamba, which is a plug-in to the Rhino like Grasshopper, for structural analysis in order to perform structural optimization by utilizing GA operators. Along the same lines, Sotiropoulos and Lagaros (Sotiropoulos & Lagaros, 2020) utilized Grasshopper to create geometry and developed a tool in C# to transfer structure to SAP2000 to perform topology optimization. Moreover, at the conference that took place in Hong Kong in 2019, Lin et al. (Lin et al., 2019) proposed a framework that utilizes Dynamo for Visual Programming and OpenSees for measuring structural performance by using FEM of created models. Additionally, there is another study by Han et al. (Han, Cao, et al., 2019) that developed a PSO algorithm for Dynamo by using Matlab and C# programming languages and tested the algorithm with simple mathematical functions.

Up to this point, a brief explanation has been provided regarding the open literature on the subject in this thesis. However, concepts such as types of structural optimization, methodologies, metaheuristic algorithms, and the algorithms utilized have not been explained in detail. That is why the remaining part of this section will focus on explaining in detail the subjects of Computational Design, Structural Optimization, and Life Cycle Assessment, which are important for understanding the concept of this thesis.

2.1. Computational Design

Computational Design is a method that uses the computer's power to employ computational algorithms, tools, and techniques to create an enhanced design process. It aims to improve the design process with the power of computation for a more effective and optimized design solution. To do that, it uses computational algorithms, tools, and techniques like parametric modeling, scripting, simulation, optimization, and data analysis for generating and evaluating design solutions. Briefly, as Dieter Vermeulen said, "Computational design is giving shape with the means of the power of a computer and scripting (Vermeulen & Eng, 2015)". The Key Centre of Design Computing and Cognition at the University of Sydney in Australia was one of the first to propose this term and has led the study, teaching, and consulting of design and computational technologies for nearly sixty years (late 1960s to now) (Gero, 1992). Traditionally, designers would rely on their knowledge and intuition while using computer-aided design (CAD) programs. However, this old approach restricts the number of design options that could be explored and the potential for providing better solutions due to its time- and resource-intensive nature (URL-2, 2023). That's why a move from traditional CAD to BIM, which solved most of the problems faced with CAD, has been seen in the last few years. With this move, new complexities have arisen, and the present BIM-focused workflow is now encountering issues. Computational Design, a valuable tool for enhancing efficiency, productivity, and performance, leading to the creation of stronger designs when integrated into the design workflow, has evolved as a solution to these new challenges (Vermeulen & Eng, 2015). It enables designers to explore a wide variety of possibilities, analyze the performance and behavior of structure, and make informed decisions based on both quantitative and qualitative criteria. By combining the creative intuition of designers with computational power, innovative and efficient design solutions can be generated. Also, without having to learn how to write code, computational design equips designers with the ability to harness programming. This is because, instead of using text-based code, the majority of computational design tools employ visual programming in which users link the outputs of one node to its inputs to build a program that moves between nodes via connectors as explained in section 2.7.4. The outcome is usually a flowchart, an illustrative depiction of the design process, or a parametric model. Design modeling programs like Autodesk Revit, Tekla Structures, Rhino, and Trimble Quadri insert visual programming features to

software generally with plug-ins like Grasshopper which can work with Tekla, Quadri, and Rhino, and Dynamo which is used in this thesis, compatible with Revit (URL-2, 2023).

2.2. Generative Design

According to Thomas Fischer and Christiane Herr, generative design is a technique in which the operator does not directly change a material's or product's properties during the design process. Instead, the operator inputs design goals and parameters like constraints and templates and uses generative systems that allow them to make interaction with products and components to autonomously generate the number of design alternatives and investigate potential design solutions while taking into account elements like performance, structural requirements, materials, manufacturing techniques, and cost limitations to get the fittest design option among created ones (Fischer & Herr, 2001; URL-3, 2023; URL-4, 2023). In simple terms, it generates several design choices by the inputs that are given to the algorithm. Then it repeats the Generate, Evaluate, and Evolve process (see Figure 1) over and over to learn from each iteration and refine or improve the results until desired outcomes have been reached (URL-4, 2023),(Vermeulen, 2018). If the AEC design challenges are considered, it has potential since it eliminates exhaustive repetitive tasks and helps designers' creativity. Especially for engineers and architects, it is nearly impossible to search for all solutions. In that point, generative design helps to find the "fittest" solution from generated designs due to its potential of overcoming conventional "non-automated" design methods by using computers' power to achieve complex tasks faster and with more accuracy (Chaszar & Joyce, 2016). It increases the designer's creativity by generating several design options which creates a wide range of variations that include "happy accidents". These "happy accidents" sometimes cause unexpected and serendipitous outcomes that positively affect the design process (Chaszar & Joyce, 2016).

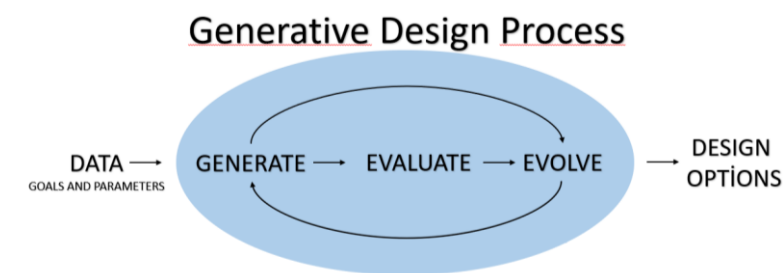


Figure 1. Simple generative design scheme

2.3. Parametric modeling

Parametric modeling is under the adaptive design umbrella which is a design strategy that focuses on developing flexible solutions that can respond to changing conditions or requirements as can be understood from the “adaptive” name. Correspondingly, parametric modeling is a process that enables alterations to model geometry and other features by only adjusting the inputs of the model. It typically uses software such as Grasshopper and Dynamo and Rhino, a 3D drafting program, which originated from AutoCAD, was the first to develop a parametric modeling concept (Fu, 2018). These technologies share features like node associations which allow for the flow of output data to successive nodes, forward-only flow without cyclical dependencies, etc. Example of a parametric design, which is created by using Revit and Dynamo, and its inputs are given in Figure 2 and Figure 3.

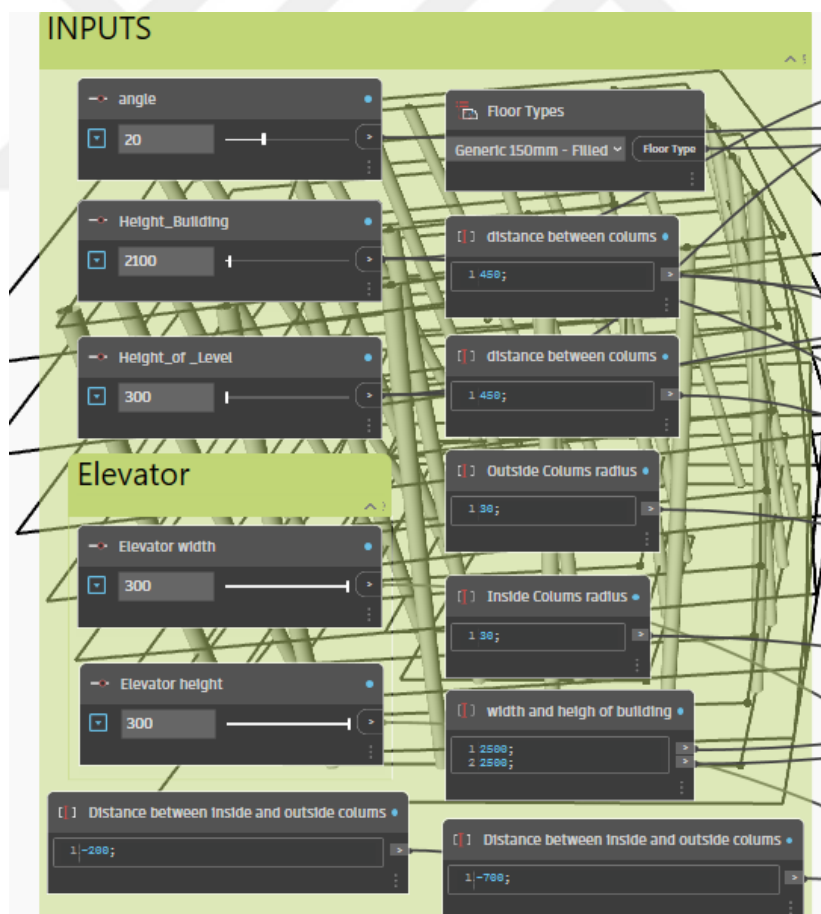


Figure 2. Inputs for parametric design

Parametric modeling eliminates the necessity of redrawing the model every time a modification is needed because updating input parameters is sufficient to change all geometry or shape of the model. A demonstration of this feature is presented in Figure 3 for better understanding. In that model, parameters like the number and height of stories, number, size and shape of columns, floor area, etc. can be updated by easily changing the inputs. This saves AEC workers a lot of time, especially during the design stage (Fu, 2018). For example, if the planner needs to modify the height, section type, material type, and length of a truss, conventionally, one needs to update each parameter for each element one by one which will take lots of time. However, using parametric modeling, designers only need to change one or two parameters, and the rest are updated automatically. Thus, by changing the parameters, even though these parameters are predefined, designers can explore different design variations and quickly generate multiple design options while saving lots of time (Chaszar & Joyce, 2016).

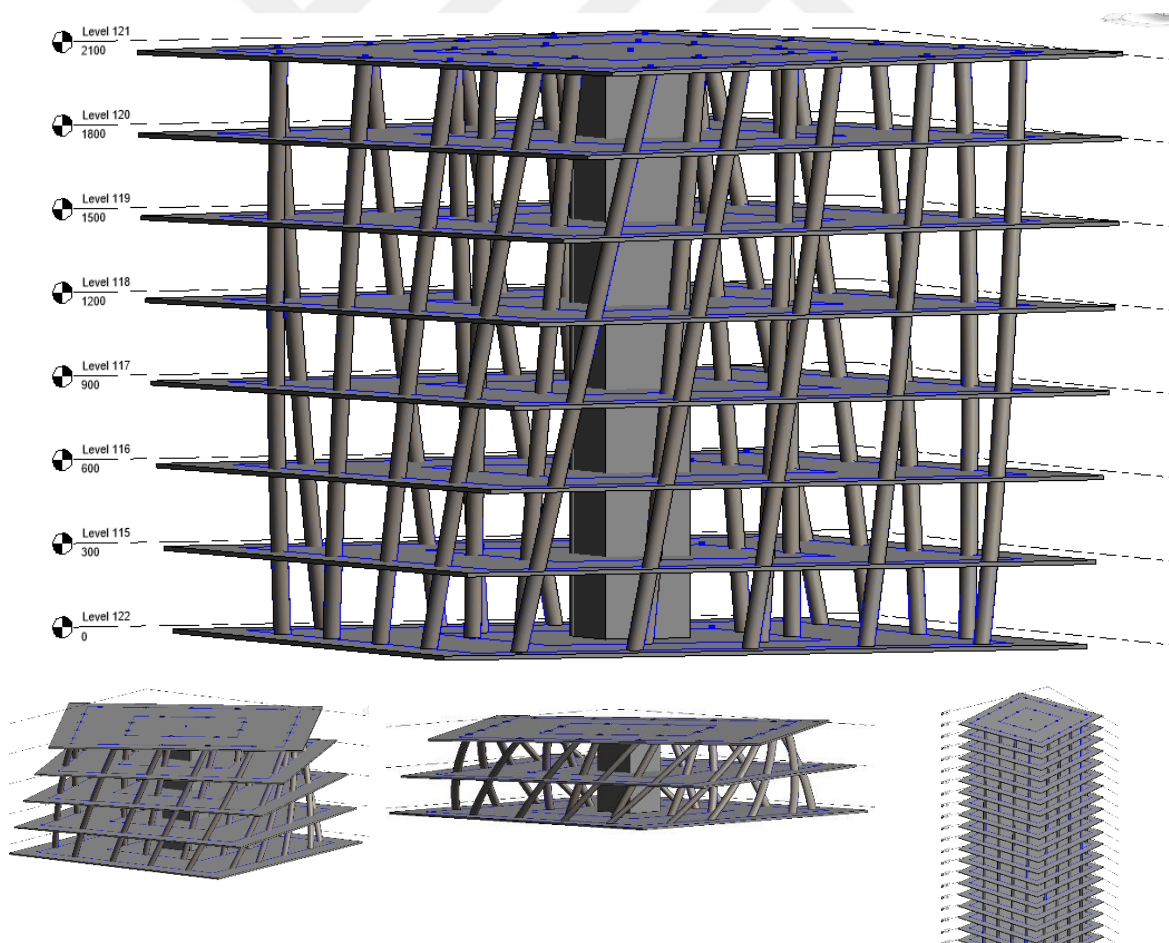


Figure 3. Example of parametric design

2.4. Structural Optimization Types and Methodologies

According to J.E. Gordon (Gordon, 1978), the goal of obtaining the optimal performance from a structure, which is the body of connected parts that is meant to bear loads safely within the engineering context, made the idea of structural optimization rise. Briefly, structural optimization is to make the product as good as it can be within its boundaries of function and the chosen concept. However, the term "best", "good", and "fittest" requires further clarification. One possible specification may involve minimizing the weight of the structure to get a lightweight design. Alternatively, creating a structure that is too stiff or extremely resistant to buckling or instability may also count as the "best" structure if there are no limitations. At this point, an important aspect has to be remembered. While looking for the best option, boundary conditions or constraints are vital because if there is not any constraint about the material used in the structure, and if the aim is to create a stiff structure result would be a structure that is infinitely stiff since there is no limitation about material used. But this design option is not realistic and it cannot be a solution for optimization problems. So, it is important to remember to set correct constraints which evaluate the performance of the structure. Constraints typically can be counted as stresses, displacements, weight of the structure, or geometric parameters, etc. in structural the optimization concept. These quantities can be regarded as measures of "fittest" design and can serve as objective functions which is explained in section 2.4.7. To formulate a structural optimization problem, one must select an objective function to maximize or minimize while employing other measures as constraints (Christensen & Klarbring, 2008). In Section 2.4.8, the mathematical formulation of such functions has been investigated. In the subsequent sections, the perspective momentarily has shifted to the type of optimization approaches.

2.4.1. Single-Objective Optimization

Single-objective optimization (SOO) is the optimization method that tries to discover the "fittest" solution or the value that results in a single objective function being minimized or maximized, as can be understood from the name, even though real-world decision-making problems often involve multiple goals which could be like maximizing reliability, minimizing weight, minimizing stresses, minimizing deviations, minimizing cost, etc.

(Cardoso et al., 2017; Savic, 2002). For example, imagine a truss design created by 3 elements. Choosing the best section to have minimum weight is a single-objective optimization problem (SOOP). But the designer might want to have minimum weight while wanting a stiff truss. Then it becomes a multi-objective optimization problem (MOOP) which will be explained in the following section. But SOO can also find solutions for that kind of problems by combining all these objectives into one single objective function (Savic, 2002), (Deb & Deb, 2014). To combine all objectives and solve MOOP with SOO methods, the constrained problem should be converted into an unconstrained one as Rajeev et al suggested (Rajeev & Krishnamoorthy, 1992). Defining an objective function that represents a score of performance or quality that needs to be maximized or minimized by adding some extra values on weight by taking into consideration the violence made on constraints is the most common way to do that (Zielinski & Laur, 2006). Therefore, this approach is really useful for simple problems. It offers useful information for designers but it does not provide many options for balancing several goals, as multi-objective optimization (MOO) does (Savic, 2002). For this reason, in some cases using MOO which is explained in the following section can be more suitable for decision makers.

2.4.2. Multi-Objective Optimization

MOO is another optimization method that tries to find the “fittest” solution for problems that have more than one objective function, as can be understood from the name, that are to be minimized or maximized. It aims to discover a variety of solutions called Pareto-optimal front which represent alternatives between the different objectives, where achieving one goal could negatively impact another (URL-5, 2023). MOOP has tremendous practical importance because it is well-compatible with real-world optimization examples that have diverse competing goals. Traditionally, these problems were approached by converting multiple objectives into a single objective, as explained above and demonstrated in this thesis. However, sometimes this causes challenges because some problems are not suitable for solving with SOO techniques, instead, they need a set of trade-off optimal solutions because finding multiple Pareto-optimal solutions becomes crucial in some cases as they provide many alternative choices between objectives for users (Deb & Deb, 2014).

To make the subject clear, an example similar to Kalyanmoy Deb's example (Deb & Deb, 2014) has been described in the following section. The example demonstrates a decision-making on buying a smartphone. Smartphones can be purchased for costs spanning from one-two hundred to several thousand euros depending on their features. In the example, two extreme hypothetical smartphones, one costing around 200 Euros (solution 1) and another costing approximately 2500 Euros (solution 2) are considered, as illustrated in Figure 4.

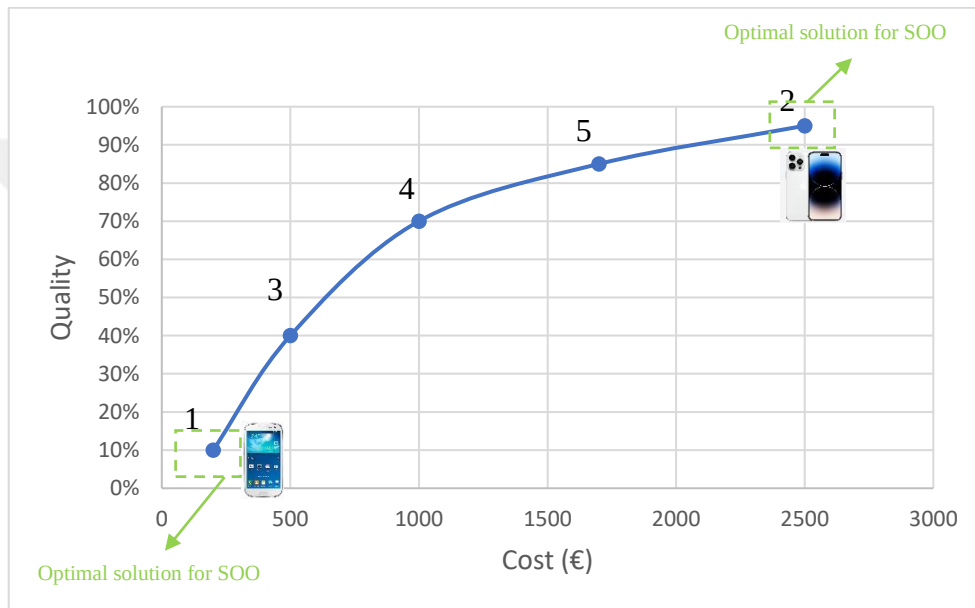


Figure 4. A hypothetical decision-making chart (adapted from (Deb & Deb, 2014))

If the main goal of this decision-making process is to spend less money, answer 1 would be the best option. If the main objective of all customers is to have the cheapest smartphone, there would be only the cheapest smartphones on the market to sell because no phone maker would have manufactured any pricey smartphones. Luckily, buying a smartphone process does not have a single goal for every customer. Because, with a few exceptions, it is believed that a cheap smartphone will have fewer features such as low RAM, the poorest camera, a bad battery, etc. The hypothetical Quality-Cost chart in Figure 4 shows that the cheapest phone has a 10% feature level. For wealthy purchasers who value quality and want to have a good smartphone, Option 2 with a hypothetical maximum quality level of 95%, is the better option. This designed two-objective optimization problem does not have to be handled in the same way as the two independent optimization problems whose outcomes are the two extreme solutions, outlined in Figure 4, because

there are many other solutions exist which can be seen as solutions 3,4, and 5 with varying prices, qualities, and performance tiers. Between any of such solutions, improvement can only be achieved at the expense of the decreasing other objective. Therefore, each of these trade-off possibilities is the optimum response for a MOOP. Such trade-off solutions are often referred to as Pareto-optimal solutions and they typically result in a line known as the Pareto-optimal front (Deb & Deb, 2014).

2.4.3. Brute Force Method

The well-known problem-solving method "brute-force search" or "exhaustive search," works with "generate and test" logic to test all possible solutions. It systematically checks every candidate if it fulfills the problem's statement or not until the stopping criteria are fulfilled or the best solution is found (URL-6, 2023). That is why the "brute-force search" approach is very time-consuming. Because of time reasons, other versions have existed that include certain heuristics and optimizations to eliminate possibilities without exploring them all. This kind of search guarantees to find the feasible solution but is most suitable for simpler and smaller problems. It can serve as a benchmark for comparison of problems that don't exist in the literature, as it does not require specialized domain knowledge. However, the brute force approach has several disadvantages, such as its poor performance and speed in discovering feasible alternatives and its lack of creativity when compared to other algorithms (URL-7, 2023). Because the brute-force only computes the predetermined set of combinations; it will not explore potential improvements. One of the main differences between GA and the brute force method is GA Optimization employs several specific techniques like the natural selection of the initial solution list and the use of cross-over, mutation, deletion, etc. operators to find the best solution (Vermeulen, 2018). That is why GA is used during the analysis process to speed things up and produce more accurate results in this thesis.

2.4.4. Shape Optimization

Shape optimization, one of the well-known structural optimization branches, aims to find an optimum shape that improves the performance of the body while satisfying constraints and boundary conditions as in all optimization problems (URL-8, 2023). In

shape optimization theory, the outer boundary of the structure or in other words, the surface node coordinates of the structure are the design variables. To solve optimization problems and reach desired outcomes which can be counted as reducing stresses, minimizing structure weight, increasing structural stability, and so on, they need to be modified (Mei & Wang, 2021),(Bakhtiary et al., 1996). It should be kept in mind that shape optimization does not change the connectivity of the structure or form new boundaries instead, it focuses on modifying the existing geometry within specified constraints (Christensen & Klarbring, 2008). In Figure 5, only the height of the truss has been optimized while other parameters are staying the same which visually presents a two-dimensional shape optimization example for trusses.

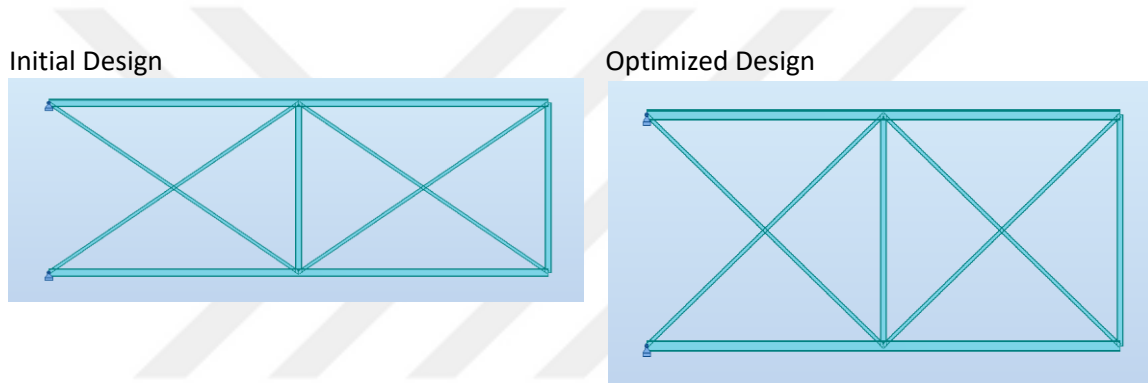


Figure 5. Shape optimization

2.4.5. Size Optimization

Sizing optimization which is also known as cross-sectional optimization, is another optimization branch that concentrates on cross-sectional areas of structures or structural elements (Mei & Wang, 2021) to find the suitable thickness distribution to fulfill the required performance or objectives (Christensen & Klarbring, 2008). Because it is easy to make calculations for size optimization, it is proven that these approaches are useful in dealing with real word problems, and they are considered as state-of-the-art in the field (Bakhtiary et al., 1996) Therefore, they are implemented in several structural analysis software and optimization tools to optimize and find the best section properties for structures. Overall, it is important to use size optimization in the structural optimization process because it allows designers to optimize the dimension of structural elements to get better performance, lighter weight, low cost, etc. (Bakhtiary et al., 1996). To make a clear

understanding about the subject, an example depicted in Figure 6 shows a truss structure sizing optimization problem, highlighting the changed section parameters to improve the design.

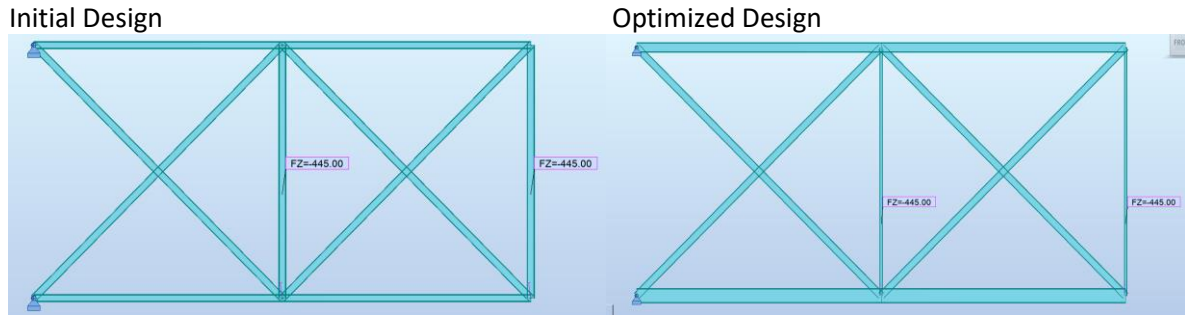


Figure 6. Size optimization

2.4.6. Topology Optimization

In order to maximize system performance, minimize weight, or maximize the strength of the structure, topology optimization (TO), another optimization technique, changes the material's position, subtracts or adds material, as depicted on an arched beam in Figure 7, while satisfying boundary conditions and constraints.

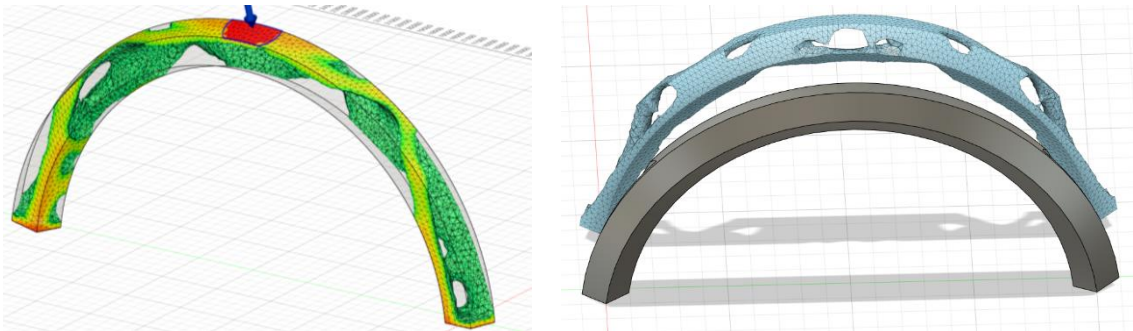


Figure 7. Topology optimization example for a curved column

The things that are different between TO and size-shape optimization can be confusing because there are some grey borders between them. A simple explanation can be made as; unlike other optimization types, TO has the ability to generate any shape within the design space, instead of being restricted to preset configurations (Mei & Wang, 2021). A truss structure can be given as an example of that. In such case, bars can be removed

from the truss structure and a different structure from the original one can be formed, as shown in Figure 8, by setting the cross-sectional areas as design variables and enabling them to take zero as a value. As a result, it is possible to say that the topology of the truss changes since the connection of the nodes is varied (Christensen & Klarbring, 2008), as shown in Figure 8.

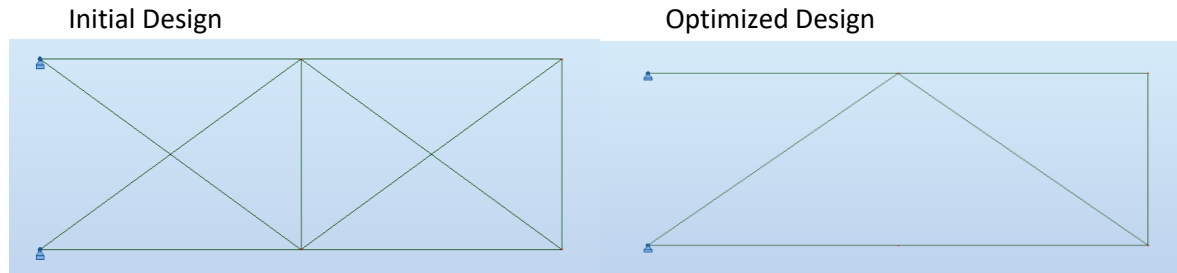


Figure 8. Topology optimization

Nowadays, the technique is growing popular because of its efficiency in real word problems. One of the real-world examples is the Qatar National Convention Centre (QNCC) located in Doha which is presented in Figure 9 (Photo usage permission secured from the official website:(URL-9, 2023)). It stands among the large-scale AEC projects that are generated through the utilization of generative tools and TO to maximize structural performance (Białkowski, 2016).



Figure 9. QNCC (URL-9, 2023)

Another good real-world example of TO applied in the design phase during the structural optimization is coming from China. The amount of material used in the Shenzhen CITIC Financial Center in Shenzhen has been decreased, while ensuring the structure's capacity to withstand external loads, after performing TO (Sun, 2020).

2.4.7. Objective Function

In an optimization problem, even though there can be multiple objectives, the commonly used objective is only the weight of the structure while other objectives are considered as constraints (Savic, 2002), (Deb & Deb, 2014). In a design problem with multiple objectives, the objective function W is expressed as shown in Formula 1.

$$W=f(c_m, c_b, c_k) \quad (1)$$

In this context, c_m represents the weights of the structural elements in the system, c_b represents the cost of joint points of the elements, and c_k represents the material properties (Rasheed, 1998). As mentioned before, optimization problems either have to be solved with MOOP techniques or they need to turn into unconstrained problems and this process is called as constraint handling. Plenty of constraint-handling techniques have been developed over the past several decades. Four categories have been used to group the most common constraint-handling techniques currently in use as Wang et al. suggested in his paper (Wang et al., 2020). Since only approaches based on the penalty function technique have been used in this thesis, the other techniques except the one used will not be explained in the following sections but those who are interested can find related papers in the bibliography attached in the following list.

1. Approaches based on penalty function (Liu et al., 2016; Tessema & Yen, 2009; Yeniay, 2005); include the death penalty, static penalty, dynamic penalty, adaptive penalty, exact penalty, the self-adaptive fitness formulation, stochastic ranking, etc.
2. Approaches based on treating constraints and objective function separately (Deb, 2000; Runarsson & Xin Yao, 2000; Takahama & Sakai, 2010).
3. Approaches based on multi-objective optimization (Cai & Wang, 2006; Jiao et al., 2013; Wang et al., 2007).
4. Approaches based on hybrid techniques (Mani & Patvardhan, 2009; Peng et al., 2018; Wang et al., 2008).

It should be noted that there are also alternative classification methodologies for constraint-handling operations in GAs. Only the most well-known had been shown there.

2.4.8. Penalty Function

As previously stated, the penalty function converts a constrained problem into an unconstrained one. There are two ways to do that. The first method involves adding some extra weight which is determined by violations made as shown below:

$$\text{Eval}(x) = \begin{cases} f(x), & \text{if } x \in F \\ f(x) + p(x), & \text{Otherwise} \end{cases} \quad (2)$$

In the formula, $p(x)$ denotes a penalty term. If there is a violation of constraints the value will be positive and it will increase the overall weight. Otherwise, $p(x)$ will take the value zero and all formula will be equal to the objective function which is $f(x)$. $\text{Eval}(x)$ represents the overarching objective function arising from this transformation, functioning as an assessment metric within genetic algorithms, as discussed in previous section (Yeniay, 2005).

The other option is to use the multiplicative form:

$$\text{Eval}(x) = \begin{cases} f(x), & \text{if } x \in F \\ f(x)p(x), & \text{Otherwise} \end{cases} \quad (3)$$

In the context of minimization problems, when there is no violation, the value of constraints $p(x)$ is one; otherwise, it exceeds one and gets higher values which increases the value of the objective function. It's noteworthy that within the genetic algorithms community, the additive penalty type has received considerably more focus than the multiplicative penalty type. That is why in this thesis additive penalty type has been utilized. Further information about the types of penalty functions can be found in the referenced article (Yeniay, 2005).

Because GA is ideal for unconstrained optimization problems and the optimization problem in this thesis involves a constrained one, it is necessary to convert it into an unconstrained one before starting the optimization process (Yeniay, 2005). The transformation is made by calculating the violation made on the constraints. These violations are penalized by the penalty term, which affects the objective function, that directs the search for workable solutions (Rajeev & Krishnamoorthy, 1992).

Rajeev and Krishnamoorthy (Rajeev & Krishnamoorthy, 1992) proposed a formulation based on the violations of normalized constraints in the paper named “Discrete Optimization of Structures Using Genetic Algorithms” and it was found to work very well. Therefore, in this thesis same methods have been used to calculate objective function. In Eq. (4), the basic formulation of the proposed methodology is presented:

$$\emptyset(x) = f(x)(1 + KC) \quad (4)$$

In the formula, $f(x)$ is the objective function and K is a coefficient that is determined by the type of problem. For this thesis a value of 10 for K was chosen based on the findings in the referenced (Rajeev & Krishnamoorthy, 1992) as it is found to work well for truss optimization problems. In situations where the design fully satisfies the constraints, the coefficient is set to a higher value, and heavier penalty values are assigned to solutions that have only a slight infringement on the constraints. This approach ensures that the constraints are not violated while the search for solutions continues. In engineering designs, minor deviations from the constraints are often acceptable (Rajeev & Krishnamoorthy, 1992). In such cases, when the main objective is to minimize the weight of the design, a smaller coefficient can be used. This allows for the exploration of solutions with lower weight, without including individuals that have extreme levels of constraint violation (Rajeev & Krishnamoorthy, 1992). This approach helps strike a balance between not violating the constraints and achieving a lighter design. C is the violation coefficient which is calculated based on the violation of constraints. The constraints are formulated in normalized form as demonstrated by the following formulas (Rajeev & Krishnamoorthy, 1992):

$$g_i(x) = \frac{\rho_j}{\rho_a} - 1 \leq 0; \quad g_i(x) = \frac{u_j}{u_a} - 1 \leq 0 \quad (5)$$

Here ρ_j is the stress value on the element and ρ_a is the allowable stress. Similarly, u_j is the displacement on the nodes and u_a is the allowable displacement. The violation coefficient C is determined in the upcoming way: if $g_i(x) > 0$, then $c_i = g_i(x)$; or if $g_i(x) \leq 0$, then $c_i = 0$ (Rajeev & Krishnamoorthy, 1992).

$$C = \sum_{j=1}^m (c_j) \quad (6)$$

where m = the number of constraints. After making all the calculations the problem turns into an unconstrained optimization problem and the new objective function becomes $\emptyset(x)$ (Rajeev & Krishnamoorthy, 1992).

2.5. Meta-Heuristics Algorithms

Metaheuristics, which gives a set of proposals or tactics for developing heuristic optimization algorithms, are updated versions of heuristic optimization algorithms on the aspect of accuracy in finding optimal solutions and success in handling complex problems (Sörensen & Glover, 2013). Heuristic algorithms are optimization methods that try to solve problems faster when compared with traditional methods although it is not certain that they will provide optimal solutions. In the dictionary, heuristic means discovery or problem-solving with the method of trial and error. They are especially helpful when the problem is computationally difficult, the search space is enormous, or it is hard to identify or assess the best solution (URL-10, 2023). The greedy algorithm can be given as an example of the heuristic algorithms. On the other hand, meta-heuristics which are named all algorithms that have stochastic components, can be considered an enhanced version of heuristic algorithms and mostly are known as nature-inspired algorithms (Rajwar et al., 2023), (Blum & Roli, 2003). The reason why they are named nature-inspired algorithms is they mimic the behavior of nature such as evolution (GA), animal behavior (Particle Swarm-Ant colony Optimization), or temperature change of materials (Simulated Annealing) (Glover & Sörensen, 2015). Particularly for complicated problems or large problem examples, metaheuristics are usually capable of offering an improved balance between solution quality and computing time (Blum & Roli, 2003).

In the following sections, three examples of the most well-known metaheuristic algorithms, according to the papers (He et al., 2018; Salleh et al., 2018; URL-10, 2023), will be explained briefly to better understand the concept of metaheuristics. GA, one of the explained algorithms, has been used during the thesis.

2.5.1. Genetic Optimization

English naturalist Charles Darwin transformed our understanding of life through his exploration and concept of natural selection in his famous book, “On the Origin of Species”. He introduced Natural Selection or Survival of the Fittest as the process by which organisms adapt and evolve. Individuals with advantageous traits for their environment have better survival and reproduction chances, passing those traits to future generations while those that are weak and unfavorable are destroyed by nature (Darwin, 1964). Briefly, the process known as evolution forces creatures to adapt in order to live in nature by keeping the traits that make an individual strong, and better suited to the environment to compete with other creatures and nature as well as the elimination of traits that make them weaker. These traits are established by chromosomes, which are made up of groups of units called genes. Not only do the fittest individuals survive across succeeding generations, but their fittest genes are also passed on to their offspring during the sexual recombination process known as crossover (Coello Coello, 2002). The algorithms that use these logics and mimic similar processes are called as Evolutionary Algorithms (EA).

GAs, which are a specific type of EA, use the "survival of the fittest" idea and integrate evolution operators into the algorithm by mimicking them to create a robust search mechanism (Rajeev & Krishnamoorthy, 1992). Similarly, Goldberg describes GAs as a mixture of Darwin's theory with a structured yet random technique of exchanging information that results in a systematic search strategy that can be used for a variety of issues (Goldberg, 1989). John Holland of the University of Michigan was the first who propose GAs in the late 1900s, which is now known as the Golden Age of Automation (Goldberg, 1989). GAs are simple from a computational point of view but powerful in their quest for improvement. Additionally, they have demonstrated to be highly flexible and capable of dealing with dynamic or changing environments, making them useful tools in a variety of disciplines, including engineering, economics, and biology (Goldberg, 1989). As previously mentioned, GAs construct their strength from the operators they employ. Reproduction, deletion, dominance, crossover, duplication, mutation, intrachromosomal translocation, inversion, migration, etc., are among the genetic operators as stated by Rajeev et al. (Rajeev & Krishnamoorthy, 1992). The number of these operators is specified in the performance expectations and type of the problem. If needed, GAs have the potential

to be enhanced by employing an increasing number of them (Rajeev & Krishnamoorthy, 1992) but the present thesis is centered around a simple GA that employs reproduction, mutation, and crossover operators because of the scope of the study as explained in the introduction section. The basic components that GAs generally use to solve a problem are listed below (Coello Coello, 2002).

1. Initialization: It is the operator that generates potential solutions called initial population which is normally executed randomly, but the designer can also determine them in the beginning. In this thesis, the initial population was created randomly by using Python. Each solution, which is often encoded as a string or binary sequence known as a chromosome, is represented as an individual in the population. A number of these solutions are called population size and the level of complexity of the problem and the computational capacity of the computer usually define the population size (Coello Coello, 2002).

2. Evaluation: This operator plays the role of the nature in real world's evolution, rating the "fitness" of solutions. This step is detailly explained in Sections 2.4.8 but usually, individuals with lower fitness values represent better solutions in GAs.

3. Selection: This operator chooses the parents that will pass the crossover process and produce the next generation. Various selection strategies such as tournament selection or roulette wheel selection can be employed in GAs (Coello Coello, 2002). In this thesis, the selection procedure works with choosing an "elite" group that has an "n" number of individuals based on fitness values. Then randomly two parents are picked from this elite list to reproduce. The number of solutions that go to the elite list and reproduction rate can be adjusted on the script based on specific needs.

4. Reproduction: In this operator, offspring's genes are determined by evolutionary operators in order to create new individuals. Among reproduction operators, crossover and mutation take precedence (Coello Coello, 2002).

5. Crossover: This operator exchanges genetic information between two parents to create new offspring. It works in three phases in the single-site crossover. The reproduction operator first looks for a match between two individual strings for pairing. Then, along the length of the string, a random crossover point is picked, and values are exchanged between the two strings based on that crossover point. For example, if two selected individuals in a matched couple are $A = 55555$ and $B = 33333$. If the crossover point is chosen as 3, the new strings after the crossover would be $A' = 55533$ and $B' = 33355$ (Rajeev &

Krishnamoorthy, 1992). This process results in offspring that inherit characteristics from both parents. The crossover technique implemented for this thesis will be explained in detail in section 3.4.3.

6. Mutation: This operator causes random changes in an individual's genes in order to create population variety. It is often accomplished by randomly flipping bits or changing values on the chromosome. The mutation prevents the algorithm from reaching inefficient answers too rapidly or focusing only on one direction. That's why Within a basic GA, Goldberg also includes a mutation operator (Goldberg, 1989), so along with reproduction and crossover operators, a mutation operator is also used in this thesis.

7. Replacement: The selection process for individuals to make room for the new offspring is based on the replacement strategy. Elitism, which keeps the best individuals, or generational replacement, which involves replacing the entire population, are common replacement tactics (Coello Coello, 2002).

8. Termination: To stop the iterations, a stopping condition needs to be defined and fulfilled. Otherwise, the algorithm keeps iterating through the phases of GA operations. Thus, it is necessary to set a stopping criterion, which can be based on the number of generations, reaching a certain fitness level, or exceeding a time limit. When the termination condition is met, the algorithm comes to an end and the best individual or collection of good solutions is chosen as the output (Coello Coello, 2002).

2.5.2. Ant Colony Optimization

Ant Colony Optimization (ACO) is a meta-heuristic algorithm that simulates some of the basic abilities of ants like pheromone communication, path selection, and memory to solve stochastic optimization problems. The algorithm begins with an ant randomly choosing a path to reach a food or an objective. While ants travel during that path it leaves some pheromones on the path for other ants. Subsequently, the ant returns to its initial location, finalizing a tour. The paths chosen by the ant form a possible solution to the problem. This process is repeated by more ants many times and at the end, the better path with more pheromones will be followed by more ants. That's how the algorithm gets closer to the solution. The cycle continues until a stopping condition is met, determined by comparing the best solution from the previous cycle with the best global solution. If this

comparison reveals that the algorithm is no longer enhancing the solution, the stopping criteria are reached and the cycle ends (Camp & Bichon, 2004).

2.5.3. Particle Swarm Optimization Algorithm

Another nature-inspired stochastic optimization technique that mimics the social behavior of birds or fish groups is called the PSO algorithm which was introduced first by James Kennedy and Russell Eberhart in 1995 (Kennedy & Eberhart, 1995). In these groups, individuals with the potential best solution share their information with other individuals. By doing that, the information is being updated constantly by group members to reach the best hunt (Kennedy & Eberhart, 1995),(URL-11, 2023). The algorithm is known for its simplicity, user-friendly deployment, and efficient convergence; thus, it has been implemented in various fields like engineering, economics, and bioinformatics (Han, Ning, et al., 2019). However, it can suffer from premature convergence or getting trapped in local optima. Therefore Han et al. stated that using different variants of the algorithm or combining it with other optimization techniques could give better results in optimization problems (Han, Ning, et al., 2019).

2.6. Life Cycle Assessment

LCA is a powerful analytical instrument that evaluates the ecological consequences of a product across its entire life cycle, from resource extraction to waste management, including disposal and recycling (URL-12, 2023). LCA gives an opportunity to the AEC specialist who wishes to build an environment-friendly and sustainable design by performing a study on the project and providing a detailed report about the materials. The report shows the scenarios about if used materials are changed what would be the impact of that action like if it reduces the environmental impact, becomes more efficient, durable, or economical (URL-12, 2023). Additionally, clients, architects, and engineers might want to design their projects to be Leadership in Energy and Environmental Design (LEED) certified which can be obtained by LCA report (URL-13, 2023). Therefore, it plays a critical role in sustainable development and decision-making processes, enabling stakeholders to make informed choices by considering the broader environmental consequences of their actions. These distinguishing characteristics of LCA prevent the

postponement of environmental problems from one stage to another and, in fact, offer solutions to them (Finnveden & Potting, 2014). Therefore, in this thesis, the LCA plug-in for Revit known as "Tally" (version 2022.04.08.01) has been utilized to show the advantages of using BIM tools in the optimization process. Detailed information can be found in section 2.7.2.

2.7. Tools

In this section, a list of the tools such as used software, plug-ins, and Dynamo packages along with their version numbers are given to provide technical data. Also, a brief explanation has been made about the software and their capabilities. Lastly, an overview of Python codes used throughout this thesis is given.

2.7.1. Autodesk Revit

Revit is created for AEC professionals to model objects, buildings, and systems in 3D with precision, and convenience (URL-14, 2023). It allows user to make instant modifications on the project which improve project administration and it brings together interdisciplinary project teams for increased efficiency, and cooperation, which increase the performance of teams in the office or on the job site (URL-14, 2023). Its capabilities support the BIM principles and allow for the integration of models from many disciplines into a single platform, allowing for collision management through add-ons. Therefore, Revit is used in this thesis as the main BIM tool to store information about models and make relationships between other used software. Dynamo has been run through Revit even though it can be run as a sandbox separate from Revit. The best result after optimization has been imported Revit to get a 3D view along with other project details. LCA analysis has been performed through Revit add-ins (Tally) and cost analysis has been made with the help of Revit schedules. While doing all of them Autodesk Revit 2024 version is used in this thesis.

2.7.2. Tally plug-in for Revit

Tally, an LCA software that enables users to accurately calculate the environmental effects of material choices immediately inside their Autodesk Revit models, is a big advance in the direction of sustainability in the sector. With Tally, AEC specialists can assess the embodied environmental impact of entire buildings, allowing them to benchmark their impact at every stage of the design process. The program makes it simple to compare multiple design options as well as allows users to compare different sets of construction components side by side (URL-13, 2023). Moreover, Tally can make LCA for components of various materials and assemblies by using information from manufacturer Environmental Product Declarations (EPDs) which allows designers to answer important questions such as whether it is more responsible to use long-lasting materials or ones with smaller initial impacts, and what trade-offs are involved in material selection. Users also can gain insights into the recyclability and reclamation potential of building components, focusing their attention on the material decisions that matter most (URL-13, 2023).

Tally, which is fully integrated with Revit, connects modeled BIM parts to real material amounts in assemblies, accelerating the design process. Also, the app has been approved for LEED v4 and refined by over 400 architects and engineers, making it a significant tool for attaining the Building Life Cycle Impact Reduction credit (URL-13, 2023). Due to these reasons, Tally has been utilized along with Revit to show readers the advantages of using BIM environment and tools while making structural optimization. Since Tally was not compatible with Revit 2024 during the thesis process, Revit 2023 has been utilized for only the LCA report. The Tally version used on Revit is 2022.04.08.01.

2.7.3. Autodesk Robot Structural Analysis

RSA is a FEM software that is under the ownership of Autodesk. This shared ownership of Dynamo, Revit, and RSA enables good interaction between these software applications (Vermeulen, 2018). RSA is specialized structural load analysis software that ensures compliance with building codes and facilitates the exchange of data with Revit through BIM-integrated workflows (Vermeulen, 2018), (URL-15, 2023). By utilizing RSA, researchers can develop resilient and constructible designs that are precise, coordinated, and interconnected with BIM (URL-15, 2023). Some of these features can be

also reached by using Dynamo. These are the reasons why Autodesk RSA Professional 2024 is utilized throughout this thesis to perform FEM analysis for created trusses. As explained before, due to FEM analysis and RSA, stress, displacement, and weight information which are necessary for objective function are obtained.

2.7.4. Dynamo

Autodesk's Dynamo is a visual programming tool that allows designers to create scripts without the need for textual programming knowledge. It offers a solution to the rising need of designers and engineers by allowing them to develop visual logic, enabling parametric designs and task automation. Also, the software can show geometries that enable designers to examine their work and make visual changes (URL-2, 2023). Dynamo speeds up the design and problem-solving processes by enabling data management such as documentation, fabrication, coordination, simulation, and analysis in the BIM environment (Vermeulen, 2018). Because Dynamo has a “Design Script” functionality that Revit does not, it offers a user-friendly interface for creating customized computational design workflows, and also by the help of the node-based system, users can visually connect different designs and analysis components which enables the creation of complex algorithms and the automation of tasks within the BIM environment (URL-16, 2023). The Dynamo graphs, which are a key tool within the methodology, are used for this thesis to perform optimization, collecting data, and structural translation. The used Dynamo version was 2.17 during the thesis.

2.7.5. Visual Programming

There are two common techniques to write computer programs: visual programming (VP) and textual programming. The one in which graphical elements and visual syntax such as blocks, nodes, or diagrams are used to create program logic and flow is called visual programming. Since it requires no significant syntactic knowledge, it can be quite user-friendly and suitable for beginners.

Textual programming, on the other hand, involves creating lines of code in a specific programming language, which necessitates mastery of syntax rules. Both approaches have their strengths and weaknesses depending on the project's requirements like textual

programming provides more flexibility and is appropriate for more challenging and complex problems (Noone & Mooney, 2018; Williams & Buehler, 1999). To explain the subject clearly, a demonstration of the two types of technique for programming have been presented in Figure 10 and Figure 11 both resulting in the same ellipse seen at the right side of the picture.

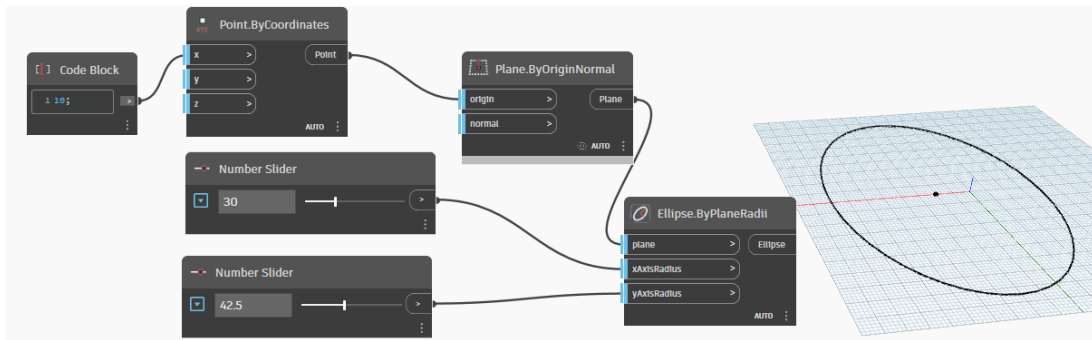


Figure 10. Example of visual programming

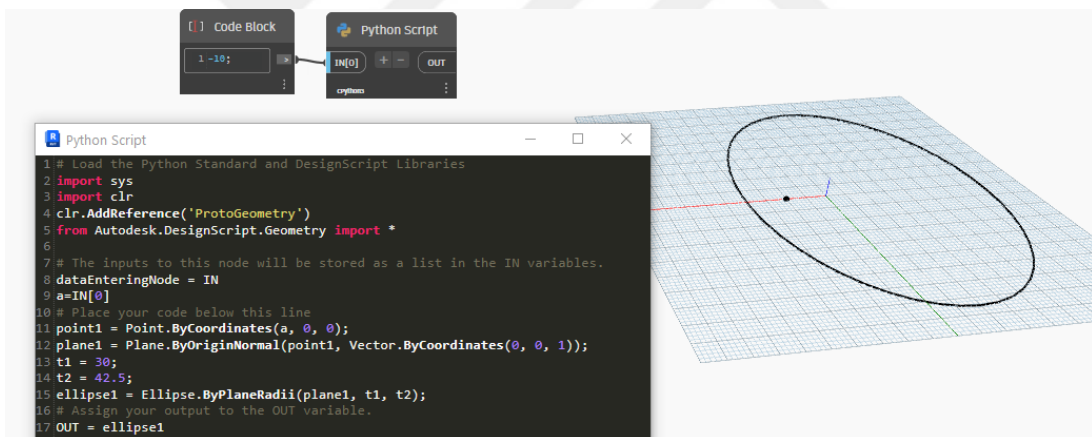


Figure 11. Example of textual programming

By using Dynamo and VP, parametric structures can be created and they can be updated easily. In addition, after a good design for the structure has been determined, it can be imported to Revit. For a better understanding of Dynamo, Revit, and parametric modeling, a basic example has been presented in this section to show those features. In the example, a parametric structure (The Absolute Tower) is modeled in Dynamo by using VP qualifications which is shown in Figure 12. In the following sections, the same structure will be modeled by textual programming to show differences. The structure is fully parametric so the user can use sliders to easily control and update the characteristics of the building like the height, width, length, and number of floors of the building.

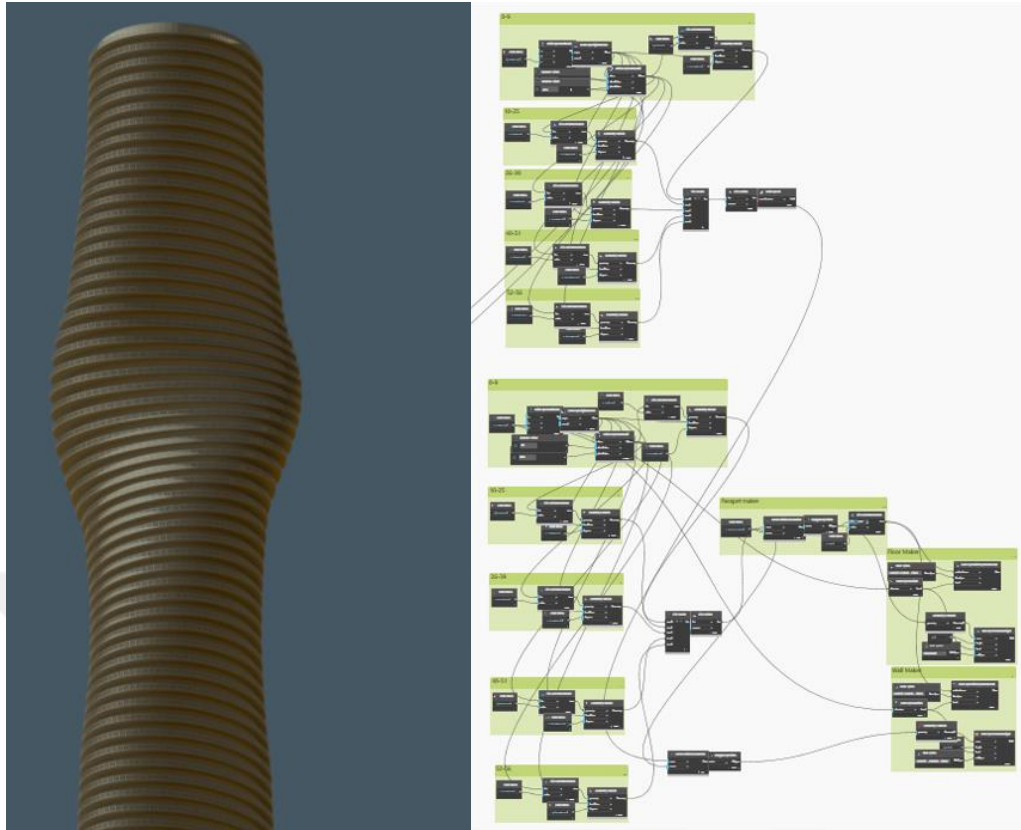


Figure 12. Example of created building on Revit by Dynamo

To generate The Absolute Tower in Revit using Dynamo, several steps are undertaken. Firstly, a series of ellipses (57 in total), representing the number of floors, are formed as previously shown in Figure 10. Subsequently, these ellipses are rotated at specific angles information that is obtained by the original project. Upon completion of these processes, the model in Dynamo takes shape, as illustrated in Figure 13.

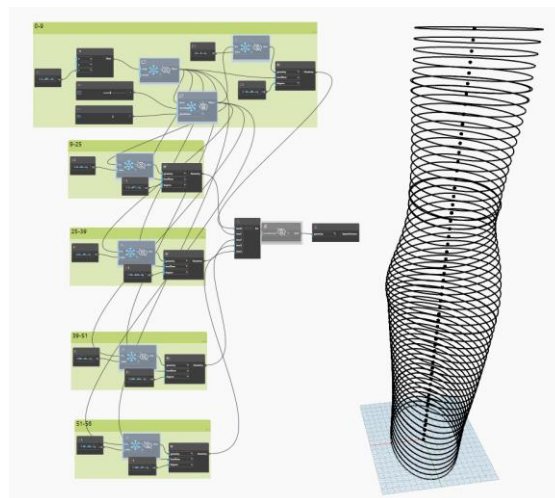


Figure 13. Rotated ellipses for the Absolute Tower

Afterward, additional elements such as floors, walls, balconies, and glass features are incorporated into the model. Figure 14 demonstrates the inclusion of wall details.

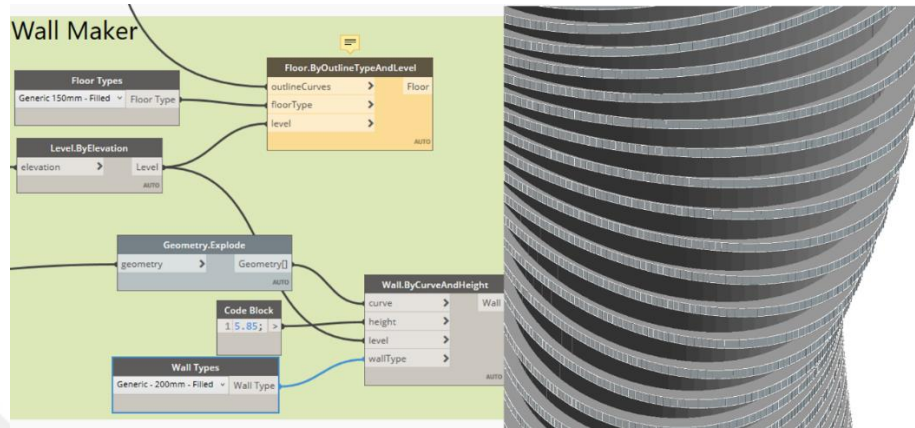


Figure 14. Example of wall maker nodes

The resulting model can be imported to Revit and further modifications and adjustments can be performed within the Revit. The structure after importing Revit is presented in Figure 15. Other than parametric modeling Dynamo and VP can also help with repetitive tasks like capturing Revit families and elements, updating them on specific needs and writing back to these elements on the Revit project, or creating an optimization script as has been prepared in this thesis.

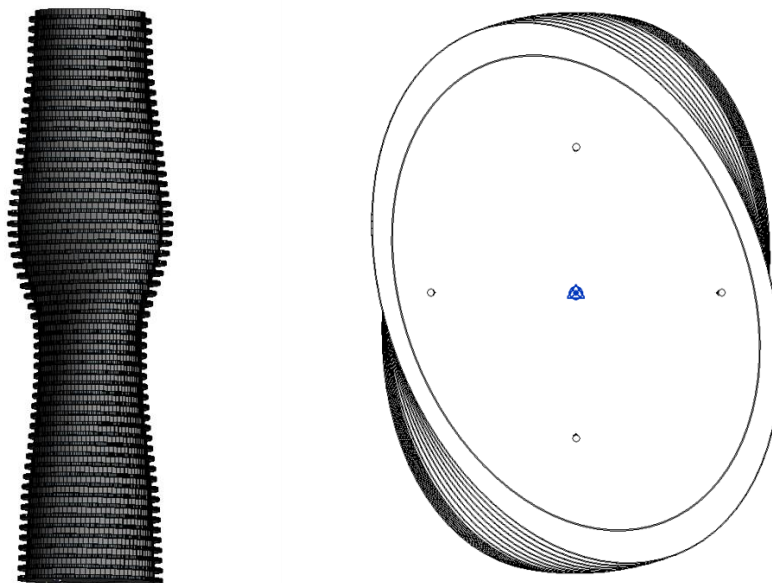


Figure 15. The final version of the model on Revit

2.7.6. Utilized Dynamo Packages

Structural Analysis for Dynamo: The "Structural Analysis for Dynamo" package is a useful tool for connecting Robot Structural Analysis to Dynamo and enabling smooth workflows between parametric modeling and structural analysis. This package has played a central role in facilitating the interaction between Dynamo and Robot Structural Analysis Professional throughout the course of this thesis. The version of the package used is v.2.17.0.3472. Detailed information about nodes that have been utilized is given in section 3.2.

CPython3: CPython3 allows the designer to use Python capabilities inside Dynamo and connects textual programming with VP. It enables users to utilize Python libraries for various purposes, including data visualization, vector and matrix creation, machine learning, and so on. This integration has greatly expanded the capabilities of Dynamo, unlocking new possibilities for those who work with it. In this thesis, CPython3 has been used for running Python codes (see Table 1) prepared for Dynamo to perform structural optimization.

Custom Nodes: Dynamo enables the user to turn their scripts into one node which is called as "Custom Node" that contains all nodes in the script. Custom nodes can be saved in ".dyf" format in the user's library and easily reused for different projects. They are recognized by their yellow background as demonstrated in Figure 16. Using custom nodes in the script has many advantages like users can share the programs that they created with each other or Dynamo community which is usually helpful for VP tasks. Other than sharing purposes, it offers many advantages in terms of speeding the process, cleaning the graphic, increasing adaptability, and enabling the script to be used inside a loop. Moreover, any modifications made are automatically mirrored in all instances of that custom node within the graph by modifying the base custom node, ensuring automatic adaption to workflow or design changes (URL-17, 2023).

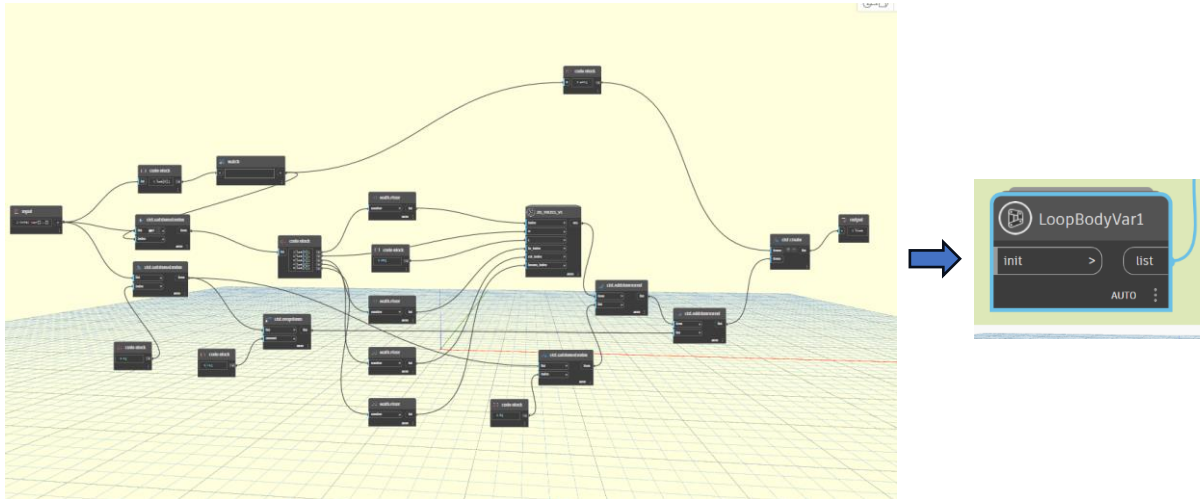


Figure 16. Example of a custom node

Creating custom nodes can be done directly on Dynamo by following the “File-New-Custom node” tabs or directly using shortcut buttons “Ctrl+Shift+N”. Afterward, it is important to define the node's name, description, and category to find them easily for future works. In this thesis, the custom nodes are used in connection with the package “Structural Analysis for Dynamo” and for specifying inputs in an optimization algorithmic loop.

2.7.7. Python

“The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds castles in the air, from air, creating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures. Yet the program construct, unlike the poet's words, is real in the sense that it moves and works, producing visible outputs separate from the construct itself. It prints results, draws pictures, produces sounds, moves arms. The magic of myth and legend has come true in our time.”

Fred Brooks, The Mythical Man-Month (Brooks Jr, 1975).

Similar to different human languages, Python is simply one of several programming languages like Java, Lisp, PHP, and Perl (URL-18, 2023). It was developed by Guido van Rossum and first released in 1991 (URL-19, 2023). After that date, it gained popularity and was widely used for real-world applications because of its ease of use, adaptability, and user-friendly syntax (URL-20, 2023; URL-21, 2023). Along with many advantages like it is completely free, easy to learn and read, etc., one of the main advantages of Python

is that there is a vast ecosystem of libraries and frameworks that provide numerous tools and resources that can serve the user's purpose (URL-20, 2023; URL-22, 2023). These are the reasons why Python has been utilized during the thesis and plays a crucial role in implementing various computational tasks and data analysis.

Also, Dynamo enables users to use Python in the VP environment with the CPython3 package as mentioned before. Here, to demonstrate the process and compare VP with textual programming, the same example which is prepared by VP and given in section 2.7.5 has been re-prepared by Python codes and the CPython3 package. Figure 17 illustrates that the script that is prepared with the Python node within Dynamo produces a significantly smaller graph, as most of the processes are handled within Python.

Moreover, an overview of the Python scripts developed for this thesis can be found in Table 1, providing a comprehensive summary of their functionalities and contributions. All Python scripts can be found in Appendix A, Python Scripts section.

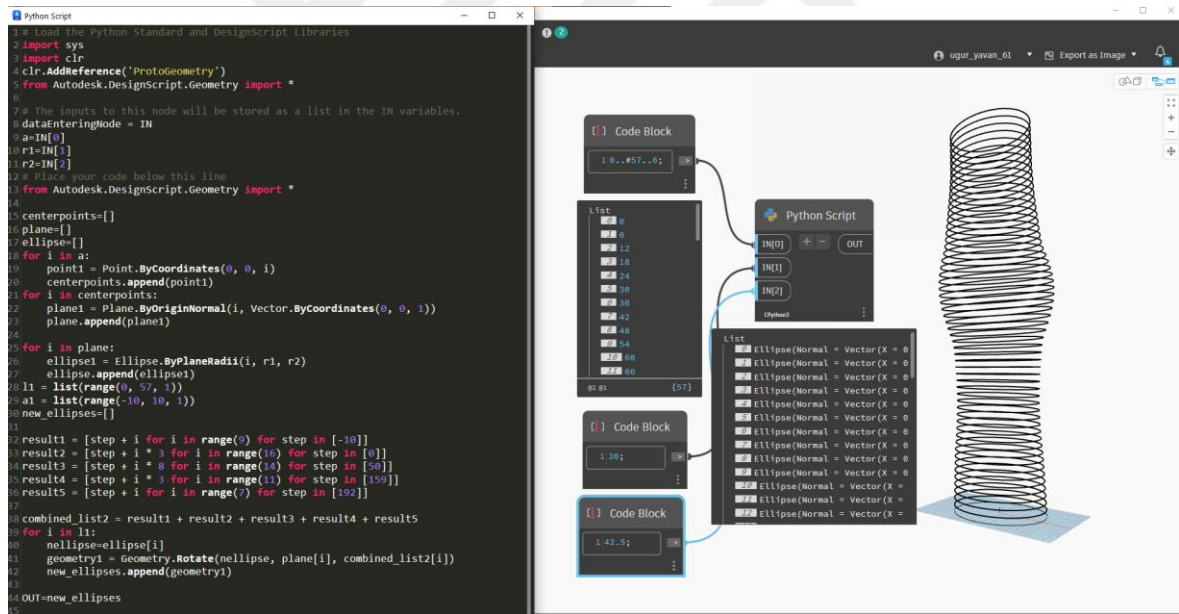


Figure 17. Example of Python usage in Dynamo

Table 1. Overview of Python scripts written throughout the thesis.

Python code's section and figure number	Explanation
Section 3.3.1, Figure 37. Python script used for evaluating and ranking the results	Takes stresses, displacements, the weight of the structure, and boundary conditions and gives a weight score belonging to the structure.
Section 3.4.2, Figure 42. Python script to create an initial population	Takes upper-lower limits, population size, chromosome length, and other boundary conditions and gives a random initial population.
Section 3.4.3, Figure 50. Next population generator	Choses best individuals from given populations and by using these individuals' information it creates new populations by using GA operators.
Section 3.4.4, Figure 53. Offspring list and the selection operator	Takes the offspring, initial population, and their fitness results as lists and then chooses the best of them to create the upcoming population.
Section 3.5.1, Figure 57. Python script that chooses the best individual and gives its parameter	Takes "original_list" as input and gives the individual parameters that have the best fitness values.
Section 4.3, Figure 88. Critical stresses after calculations	The code that calculates critical stresses for buckling regulations.
Section 4.3, Figure 90. Updated weight score calculator for buckling	Updated code for buckling constraints that calculates weight scores for possible solutions

3. METHODOLOGY

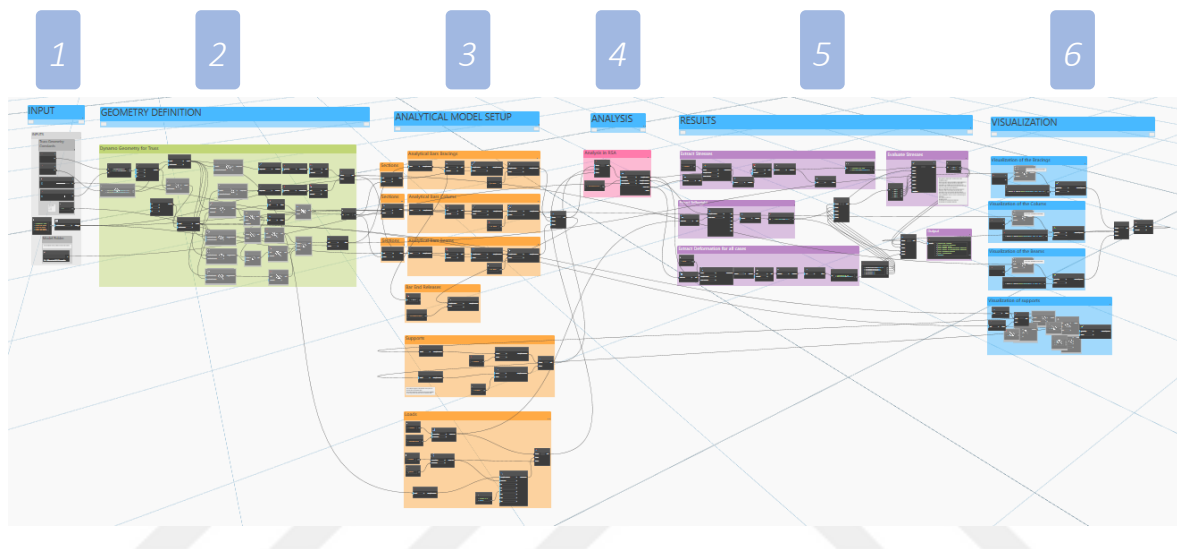
In this section, a comprehensive explanation of creating a model on Dynamo, performing structural analysis utilizing FEM in RSA through Dynamo, retrieving the analysis results on Dynamo, and at the end of these steps optimizing a structure in the early design phase by using a metaheuristic algorithm has been explained on a simple example to help readers understand the necessary steps. Along with them, the advantages of using BIM tools during the optimization process are also shown with examples. Table 2 provides an overview of the structural optimization process developed for Dynamo throughout this thesis.

Table 2. Optimization processes

Optimization Process
1. The user sets Inputs, limits, and stop criterion
1.1. User sets the upper and lower limits of project variations
1.2. User sets the population and chromosome size
1.3. User defines the boundary conditions, material properties, and section inputs
1.4. User sets the stop criterion
2. Start the optimization
2.1. User defines the initial population or creates a random initial population by the script provided
2.1. Dynamo calculates the initial population's fitness values with the help of RSA
2.2. Prepared code chooses the best individuals from the initial population based on fitness values
2.3. New generations are produced from chosen populations
2.4. Dynamo calculates the new generation's fitness values
2.5. Restart the process from step 2.2 if the design criterion is not met
2.6. Stop the iteration if the design criterion is met
3. Importing structure to Revit and utilizing BIM advantages
3.1. Run the Revit Creator node on Dynamo
3.2. Add assembly details to the structure
3.3. Perform LCA and Cost analysis

3.1. Prepare model on Dynamo

The Dynamo script for generating the structural model on either Revit or RSA comprises six key components, as depicted in Figure 18. In this section, a step-by-step explanation (Step 1 and Step 2) of the process for creating a model using VP on Dynamo has been provided.



- Step 1: Input parameters, such as sections, truss geometry, etc.
- Step 2: Generation of the model geometry in Dynamo
- Step 3: Generation of the structural model in RSA
- Step 4: FEM analysis with the help of RSA
- Step 5: Collecting results from RSA and calculation weight score
- Step 6: Colorized visualization of the wire model created on Dynamo

Figure 18. General overview of the Dynamo script

3.1.1. Step 1: Input Parameters

Figure 19 illustrates the use of a number slider, integer slider, or code block nodes to specify various parameters such as truss height, truss length, the number of vertical struts, sections and their indexes, the truss type, and the folder for storing analysis results. These values are all expressed in millimeters (mm), aligning with the Project Unit setting in RSA.

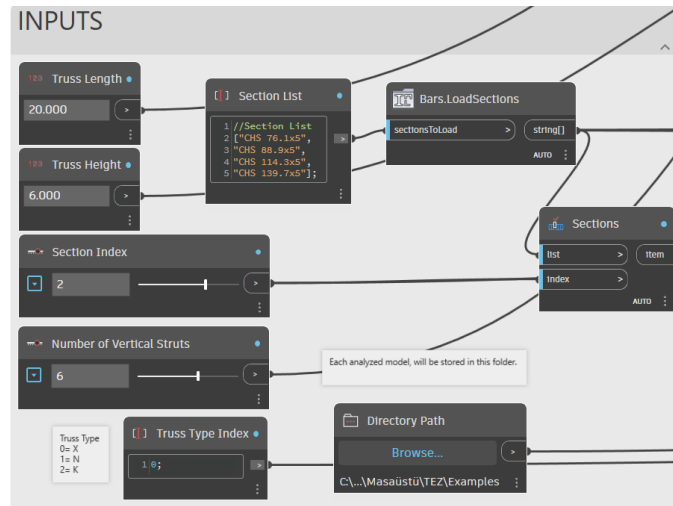


Figure 19. Input parameters for the truss

3.1.2. Step 2: Generation of the Model Geometry in Dynamo

In this section, a detailed explanation and demonstration of the nodes used to generate outline curves for the truss structure are provided. Figure 20 shows the visual representation of these outline curves that will create truss elements in the following sections. Also, the adjustment of the truss type by changing the index value is clearly indicated in the same figure.

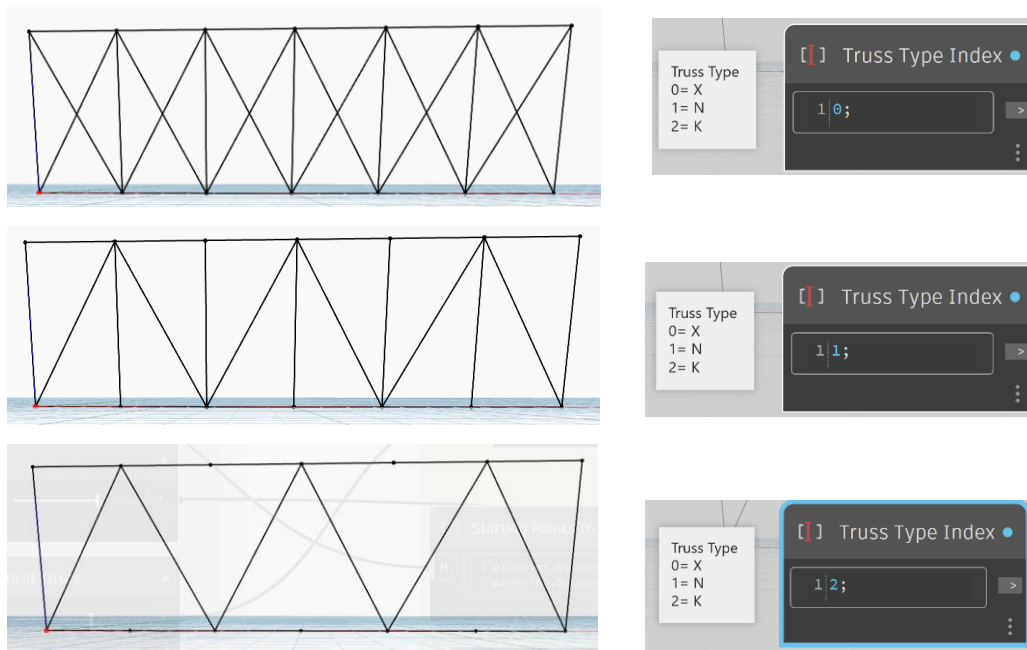


Figure 20. Geometries explained in this section

To create the truss structure, the starting points need to be defined. This can be achieved by either entering the code text “Point.ByCoordinates(0,0,0)” in the code block or utilizing the “Point.ByCoordinates” node, as demonstrated in Figure 21.

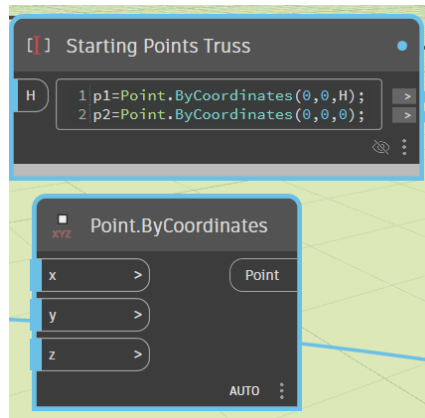


Figure 21. Usage of the Dynamo nodes

In the next step, two vectors were added to the previously created points using the “Vector.ByCoordinates” Then as many points as the number of vertical struts were generated along the vectors. These points were then connected using the “PolyCurve.ByPoints” node, forming the beams of the truss structure. After the beams were created, three different types of custom nodes were developed for the bracings. These custom nodes, how to use custom nodes explained on page 33, were designed to provide adaptability to the bracings. To provide an example of the nodes inside the “x bracing” custom node, the prepared graphic is given in Figure 22 while Figure 23 shows the utilization of these custom nodes for the bracings.

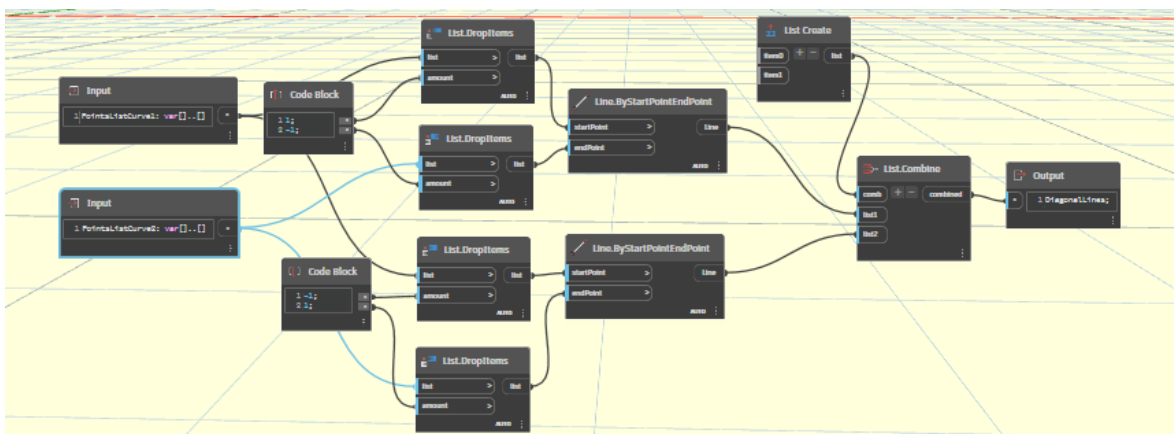


Figure 22. Nodes inside the "x bracing" custom node

In Figure 23, lines were created for different bracings and then collected into a single node using the “List Create” node. Subsequently, the “List.GetItemAtIndex” node is used to get lines corresponding with the type of bracing. The node shown in Figure 24 will be connected to the “List.GetItemAtIndex” node from Figure 23 to change the type of truss by choosing the list that will create bracings.

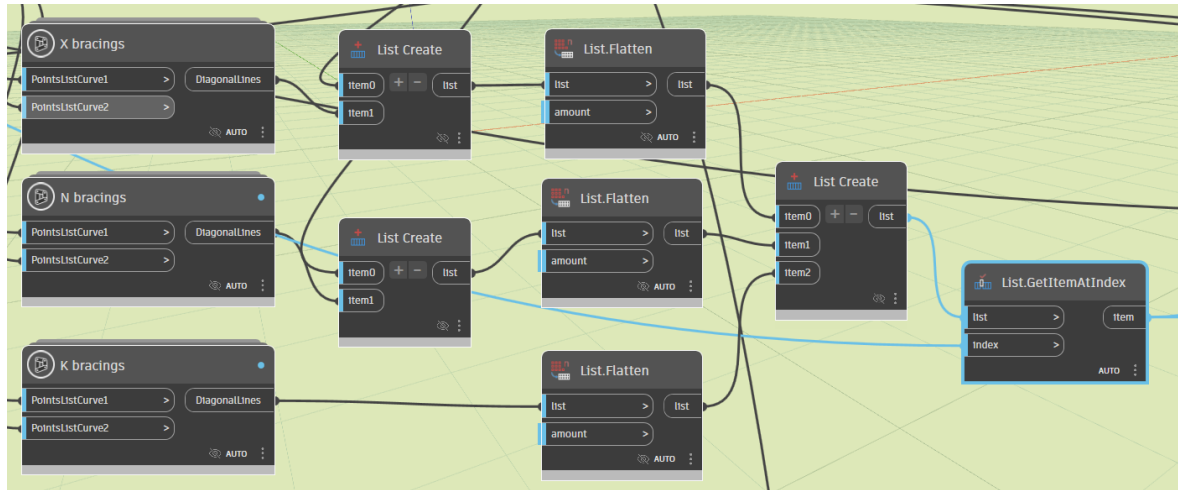


Figure 23. Creating the adaptive bracings

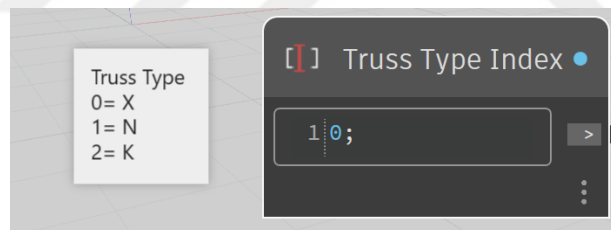


Figure 24. Truss type input

After all, the nodes responsible for creating the geometry lines of the truss can be seen in Figure 25. This section has to be updated whenever different trusses want to be optimized. In the Results section, different truss examples and corresponding nodes to create these trusses are provided.

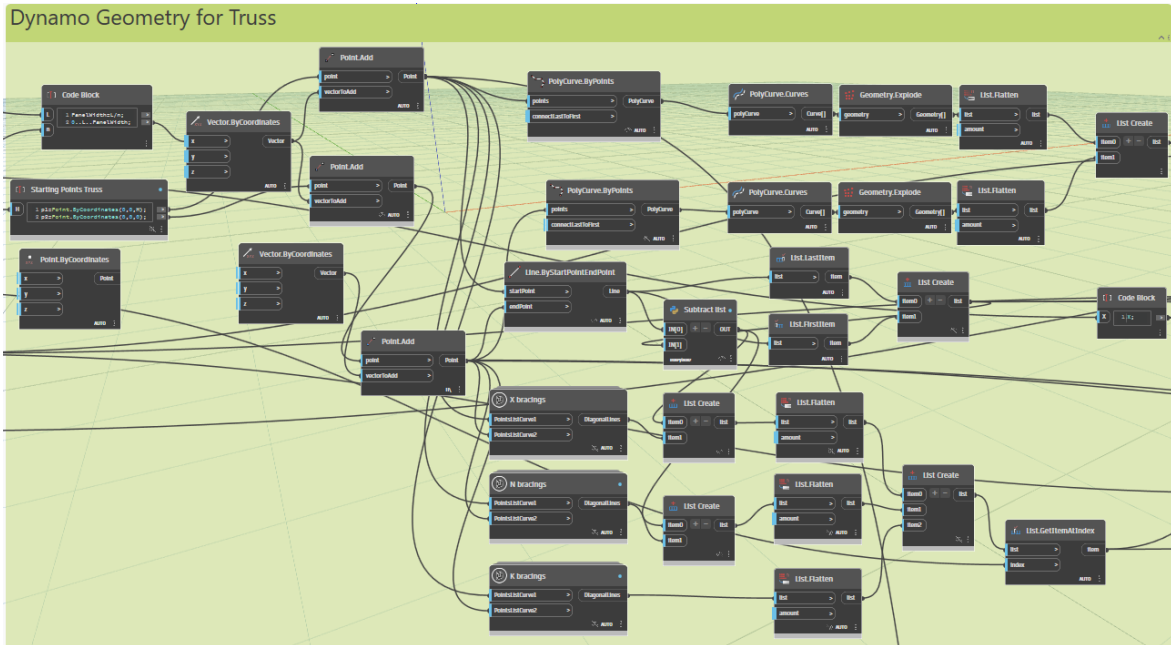


Figure 25. Nodes to create the geometry lines of the truss

3.2. Perform Calculation and Retrieve Results

In this section, a step-by-step explanation of the process for performing a structural analysis (Step 3, Step 4, and Step 5) using the FEM in RSA by Dynamo has been provided. To establish a connection between Dynamo and RSA, the Structural Analysis for Dynamo package (refer to the section “Structural Analysis for Dynamo” for more details on used Dynamo packages) is required. This package enables the creation of geometry and the assignment of simulation criteria, such as supports, loads, and sections, based on the geometrical input from Dynamo within RSA.

3.2.1. Step 3: Analytical Model Setup

Creating the Analytical Members and Assigning the Sections: As for this part, the lines created in the previous section need to be connected with the "AnalyticalBar.ByLines" node from the Structural Analysis for the Dynamo package. This node takes the lines and creates the bars in RSA. Once the bars are created, their sections need to be assigned using the "AnalyticalBar.SetSectionByName" node. This node requires two inputs: one connected to the bars created from lines using "AnalyticalBar.ByLines"

and the other being the name of the section available in the open RSA template. To load the available sections, the "Bars.LoadSections" node is used in Dynamo. Then, with the "List.GetItemAtIndex" node, the desired section for a specific bar can be selected and connected to the "AnalyticalBar.SetSectionByName" node, as shown in Figure 26.

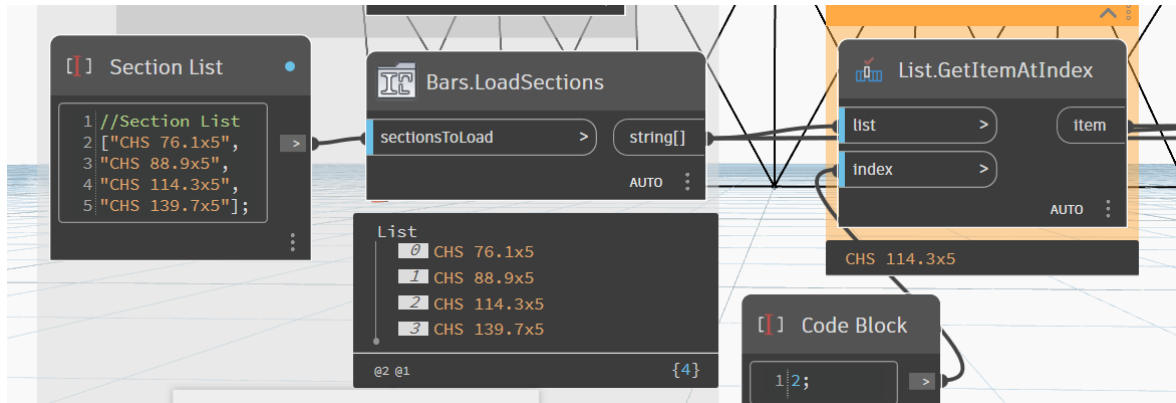


Figure 26. Assigning the sections on Dynamo

The final step in creating the bars is assigning the material to them using the "AnalyticalBar.SetMaterialByName" node. It is important to note that the material should be available in RSA, and the node should be given the exact same string as the material name in the RSA template. The complete script for these steps is illustrated in Figure 27.

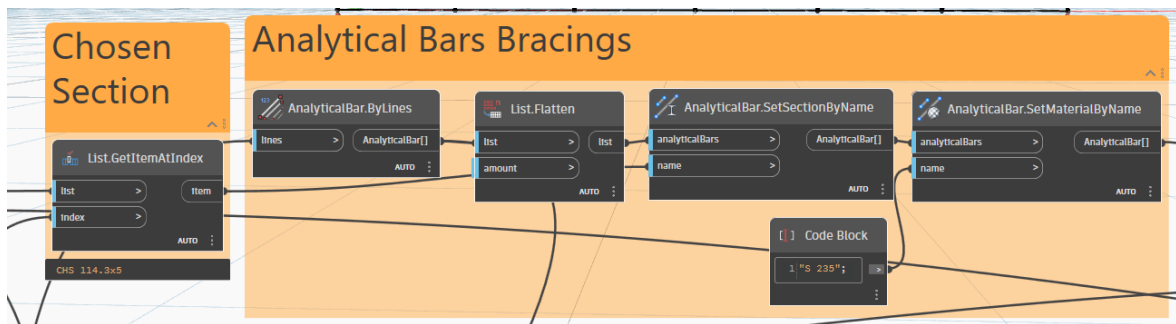


Figure 27. Creating the bars, sections, and materials on RSA

Defining Supports and Bar End Releases: The subsequent step involves the definition of bar end releases and supports. This is accomplished by utilizing the "AnalyticalBar.SetReleaseByName" node as shown in Figure 28.

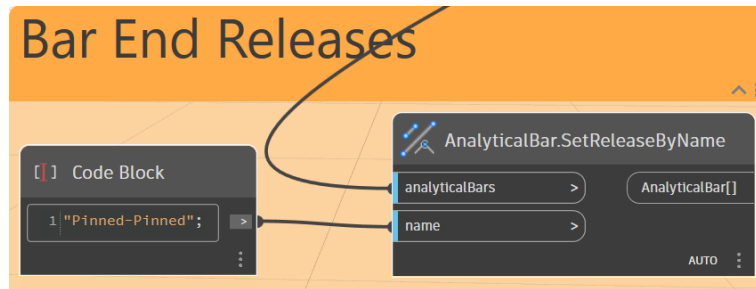


Figure 28. Assigning the bar-end releases

It is necessary to provide the name of the connection as a string and ensure that it is properly defined in the RSA template. To check it in RSA supports or releases should be clicked under the geometry menu as shown in Figure 29. Since the objective of work optimization of trusses, the “Pinned-Pinned” connection was used for this model.

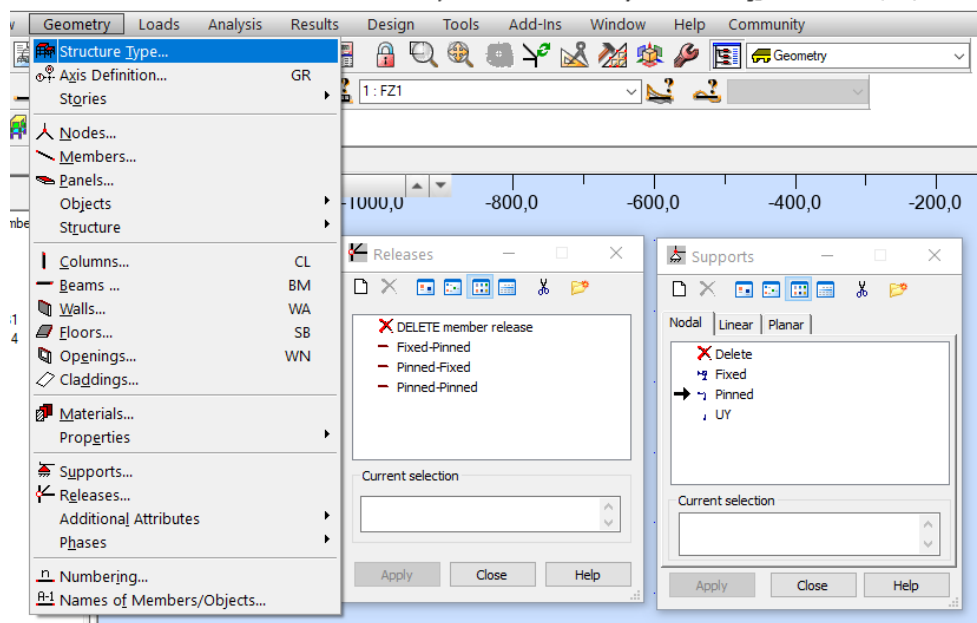


Figure 29. Definition of the Supports and Releases in RSA

To establish support for the structure, the "AnalyticalNode.SetSupportByName" node was utilized. This node requires two inputs: the desired location of the support (point) and the type of support. The support type should be defined as a string and it should exist in the RSA template before assigning it. For this study, "Pinned" supports were utilized, as depicted in Figure 30.

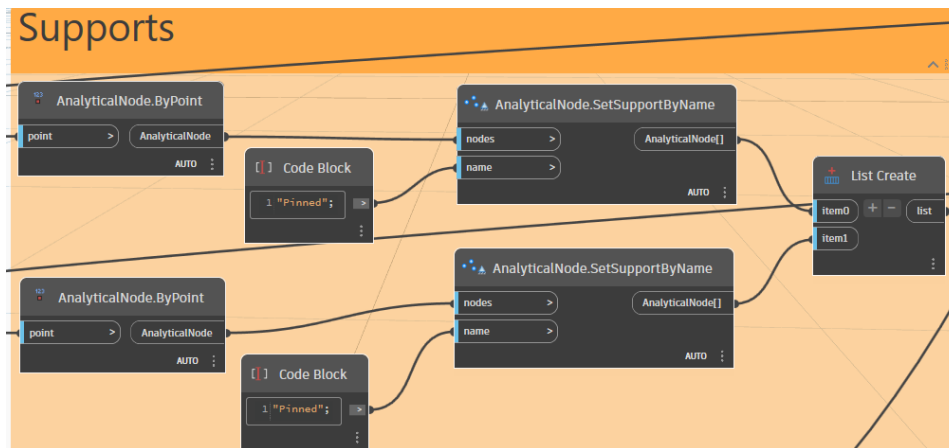


Figure 30. Defining the supports on Dynamo

Applying the Loads: The last step of this section is defining the loads. To do that, the “LoadCase.ByNature” node has been utilized. Two load type have been applied during the process and there are two input port that accepts strings of this node. Case Nature which is “Dead” for this scenario and case name which is “selfweight” for the first load type as shown in Figure 31. For the second load type the parts that need to be loaded in this structure have to be chosen and connected to the “analyticalNodes” port in the “NodalLoad.ByNodes” node. After defining the load case with the same steps as the self-weight case, the point load’s magnitude has to be defined by a code block or integer slider. The input of this node for f_x , f_y , and f_z is expressed in N as shown in Figure 31.

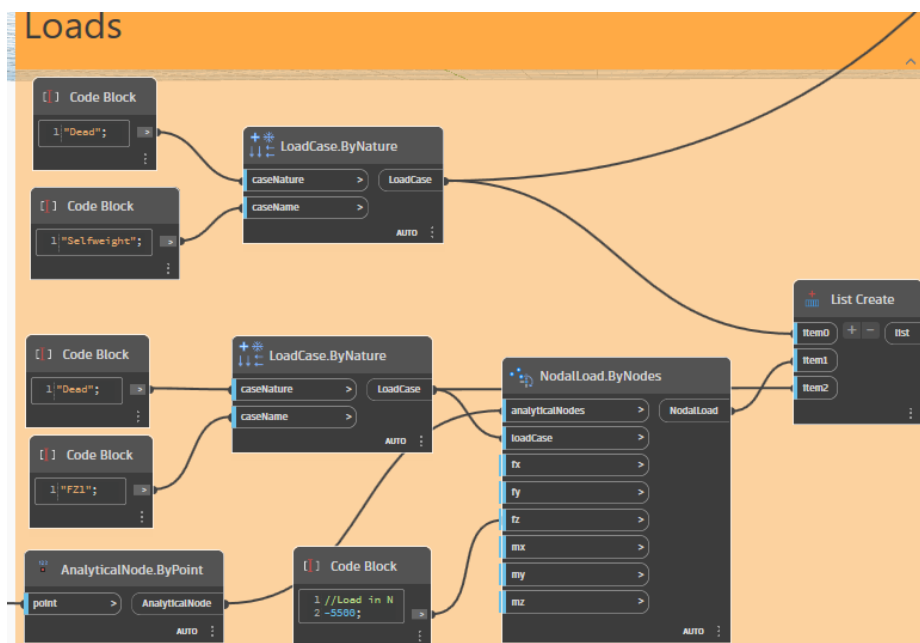


Figure 31. Definition of the loads for RSA

3.2.2. Step 4: FEM Analysis with the Help of RSA

Dynamo drives the analysis process in RSA by using the “Analysis.CalculateWithSave” node. The information (analytical model, support and releases, load cases) that has been created in the previous section has to be connected to this node but before connecting it is important to first create a flattened list that contains all elements like in Figure 32. This node performs the analysis and saves the model at a specified location. To verify the results in RSA, “Results - Diagrams for Bars” path can be followed (Vermeulen, 2018; Vermeulen & Eng, 2015).

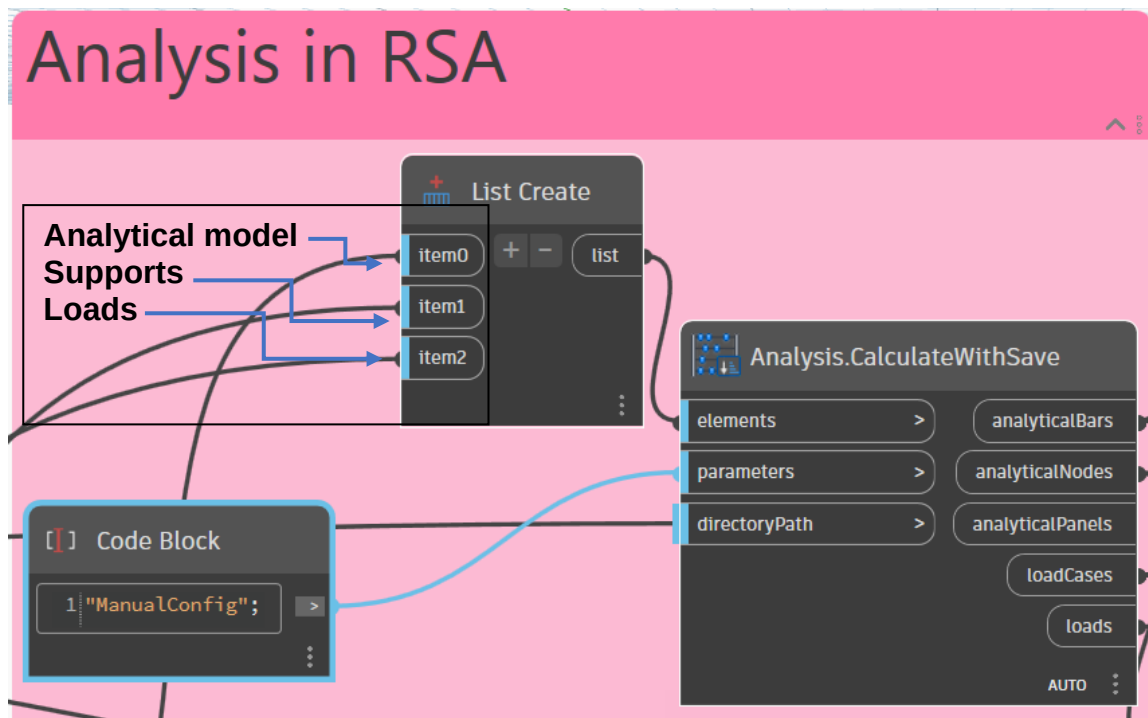


Figure 32. RSA analysis on Dynamo

3.3. Evaluate the Results

3.3.1. Step 5: Collecting Results from RSA and Calculation of Weight Score

Stresses and displacement are needed for the calculation of the weight score. To get the maximal values of S_{max} which is the maximal stress in each bar, the “BarStress.GetMaxValuesList” node is utilized. The “analyticalBars” input in this node is

connected with the “analyticalBars” output from the “Analysis.CalculateWithSave” node. Also, it is important to connect the right load case with the “loadCaseIds” input port. This node does not separate the stresses as compression or tension even though it is necessary for buckling constraints. So, the distinction between tension and compression is made by examining whether the values are positive or negative. Positive values represent tension, while negative values indicate compression. The “List.FilterByBoolMask” node has been utilized for separation processes, as shown in Figure 33.

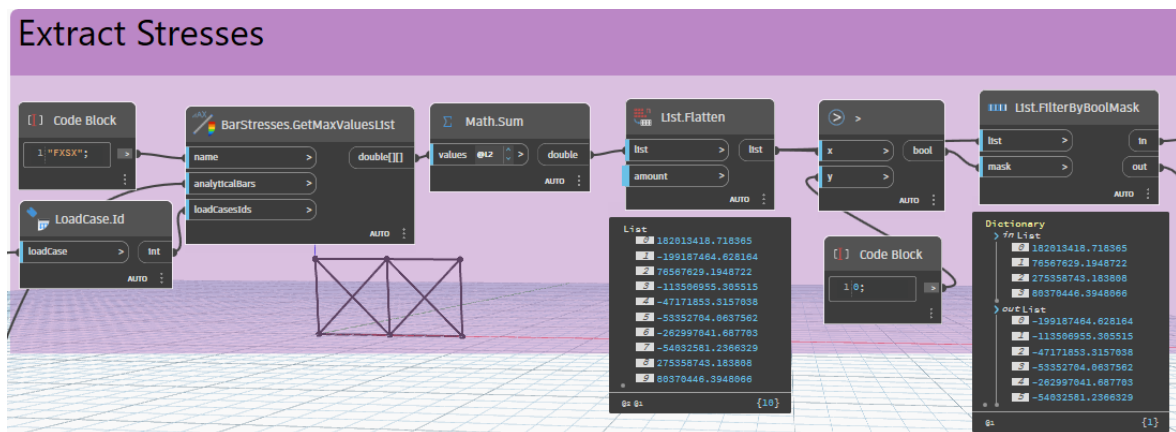


Figure 33. Extracting the stresses from the analysis

The results for the stresses need to be converted from newtons per square meter (N/m^2) to newtons per square millimeter (N/mm^2), which has been achieved by dividing the stress results by 1,000,000 as depicted in Figure 34.

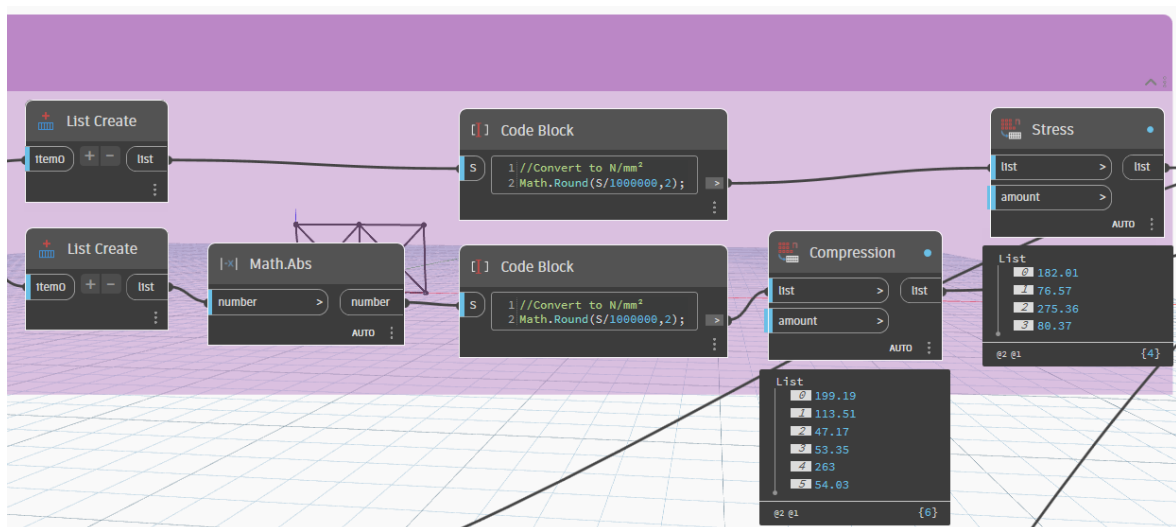


Figure 34. Changing units and grouping the stresses

To get the maximal values of displacement on the joints, the “BarDisplacement.GetMaxValuesList” node is utilized with the same steps explained in the previous section as can be seen in Figure 35. The obtained displacements are in meters (m) and have been converted to millimeters (mm) by multiplying them by 1000.



Figure 35. Extracting the deformations from the analysis

The last necessary parameter for the weight score is to get the “selfweight” value of the structure. Figure 36 shows using the “NodeReactions.GetValues” node and extracting the “FZ” results, which represent the vertical reaction forces for all the nodes that have support assigned to them from the “Analysis.CalculateAndSavenode” node, specifically for the load case labeled as “Selfweight”. The result obtained for self-weight is in newtons (N) and needs to be converted to kilograms (kg). Using a code block, the self-weight is converted to kg by dividing it by 9.81.

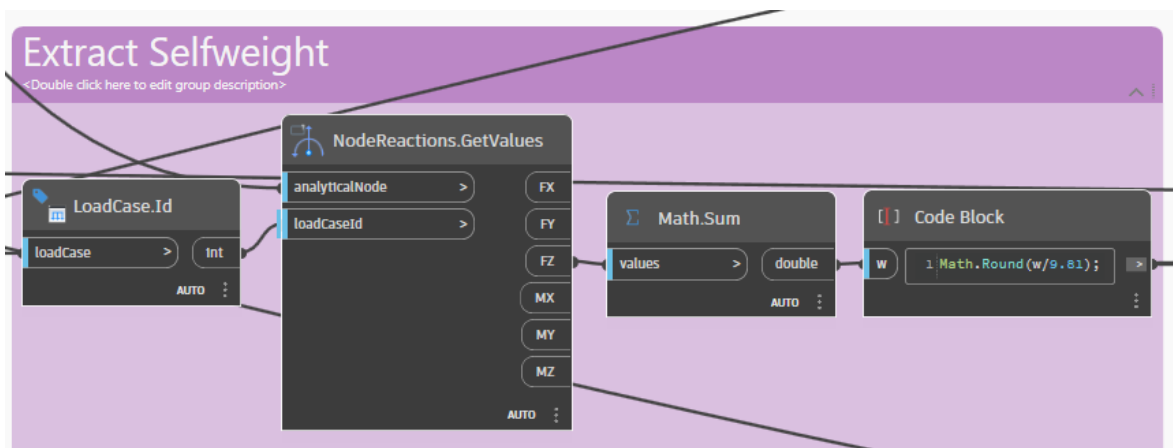


Figure 36. Extracting the self-weight from the analysis

Evaluating and assigning a weight score to each design option is crucial in order to rank the fitness of designs while using the SOO method as explained in Section 2.4.1.

That's why the goal of the last part of the script is to evaluate the bar stresses and apply "penalty weight values" to bars that fall outside a given stress ratio and displacement range. The stress ratio is calculated as the ratio of stress to allowable stress ($R = \rho_j / \rho_a$ or $R = u_j / u_a$), considering both tensile-compression stresses and displacements. Similarly, the displacement ratio is calculated using the same logic. If there is a violation of the boundary condition, the penalty involves adding additional fictive weight to the actual structure weight. The closer the "Weight score" approaches the "Actual weight", the more optimal the design option is considered.

Before calculating the weight score, the allowable stress value for steel has to be entered into the code block. The allowable stress is defined by the object's safety factor as well as the yield strength, which is the stress at which an object will be irreversibly damaged if exceeded. The yield strength depends on the material's mechanical properties. For example, the number in "S235" refers to the material's yield strength which means the deformation will be elastic while its maximum capacity that will carry the load without break is somewhere between 360-510 MPa. In the example provided for explaining methodology, the "S235" type of steel used, and for allowable stress, it is divided by a material safety factor which can be obtained from steel design regulations of countries. It is important to note that the example provided in the methodology does not follow any regulations because it is created to show and explain the methodology to readers.

Penalty weight values are applied to members that exceed the boundary conditions which are the ratios (for stresses it is the ratio of obtained stress on material to allowable stress) greater than 1. The ratios less than 1 are considered safe and do not receive additional penalty weight. For members with stress ratios greater than 1, penalty values are applied to ensure they do not exceed allowable stress limits. The formula used is " $WS = W(1+KC)$ ", where WS is the penalty function or weight score, W is the actual weight, K is the coefficient, and C is the sum of all violations. A detailed explanation of the penalty function can be found in Section 2.4.8.

This process is implemented to the VP script with the help of Python script and using a for-loop that iterates over each bar inside it. The Python script responsible for these calculations is shown in Figure 37. The script can be found in text format in Appendix A, Python Scripts section. Also, an explanation of the script inputs can be found in Table 3.

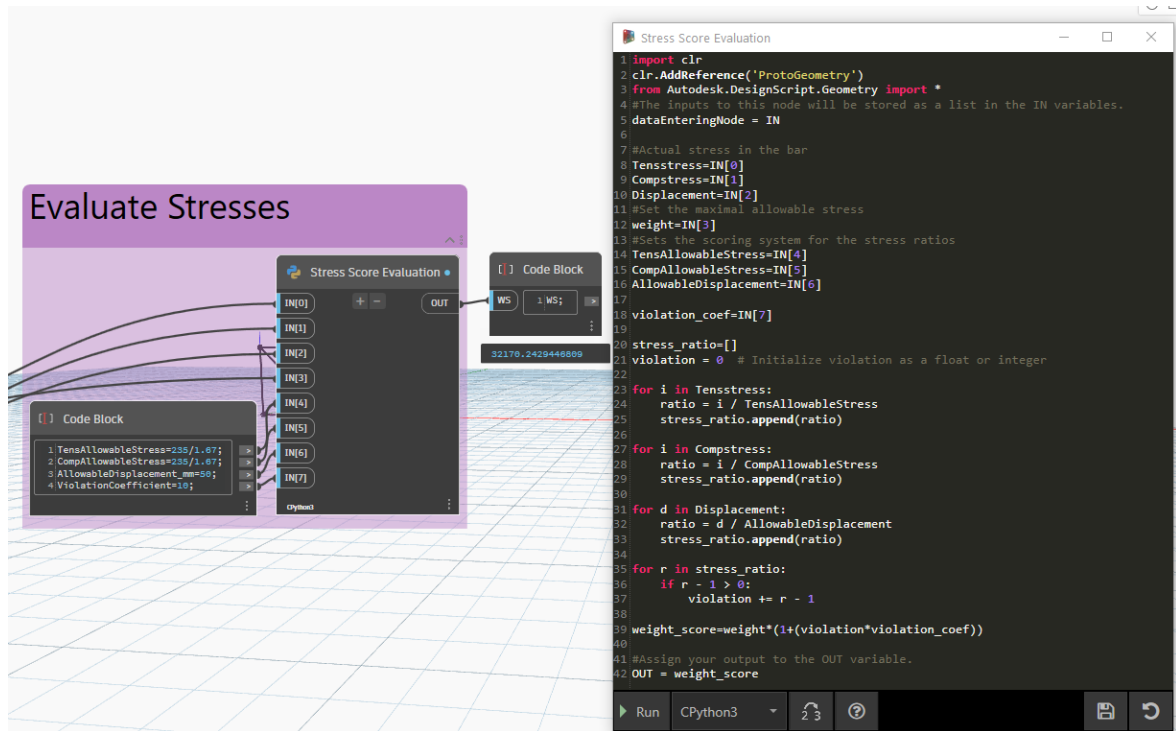


Figure 37. Python script used for evaluating and ranking the results

Table 3. Python inputs and explanations for the weight score calculation

Python input	Explanation
IN[0]	The list of tensile stresses
IN[1]	The list of compression stresses
IN[2]	The list of displacement at the nodes
IN[3]	The weight of the structure
IN[4]	The allowable stress for tensile stress
IN[5]	The allowable stress for compression stress
IN[6]	The allowable displacement
IN[7]	Violation coefficient
	Out = Weight Score

3.3.2. Step 6: Colorized Visualization of the Wire Model Created on Dynamo

This step is not necessary for optimization, but if desired, the created truss and supports can be visualized on Dynamo using a wire model and the nodes shown in Figure 38. The figure only illustrates the visualization of bracings to save space from the image. The other groups are also created with the same method in the main script.

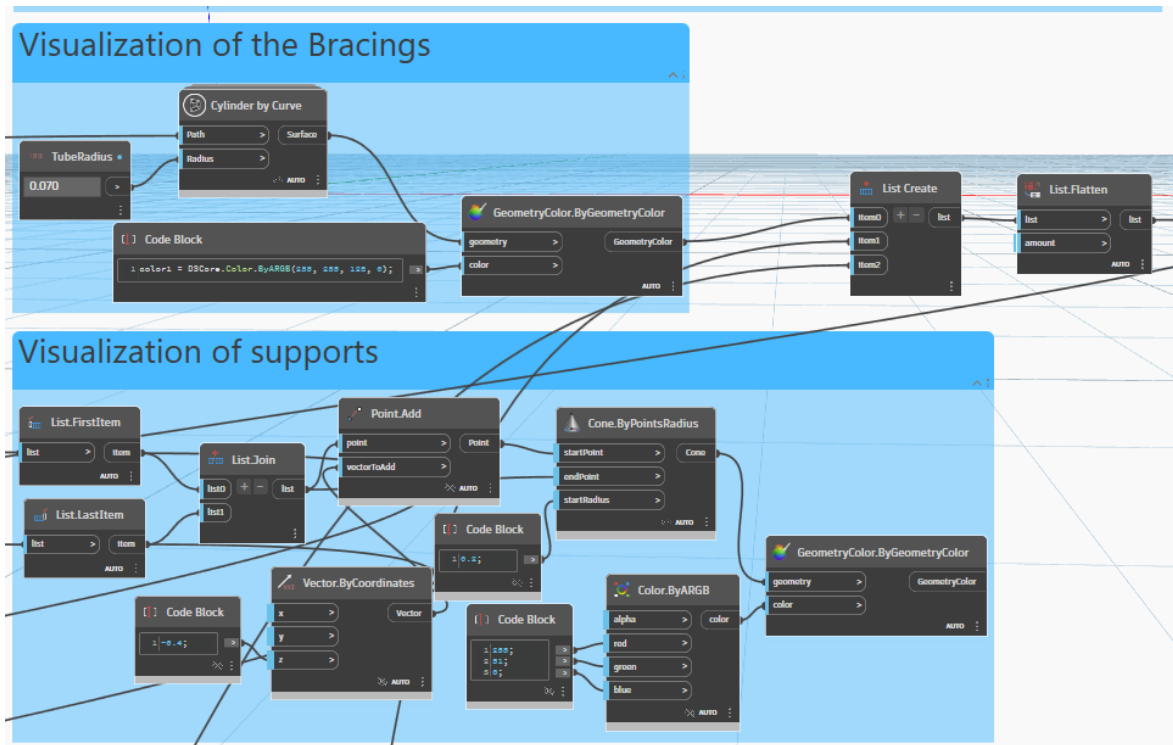


Figure 38. Necessary nodes to visualize the model on Dynamo

Circular sections have been used for visualization, and the truss elements have been divided into groups, each represented by a different color as depicted in Figure 39. It is good to remind that the grouping that has been made in this section is just for explanation reasons. Different grouping systems can be made for different needs. In this section, the example structure has been divided into three groups. These groups can be also assigned with different sections during optimization, as mentioned earlier. The “GeometryColor.ByGeometryColor” node has been used to assign different colors to the geometry. To create circular sections from the wire model, the “Circle.ByCenterPointRadiusNormal” node has been utilized.

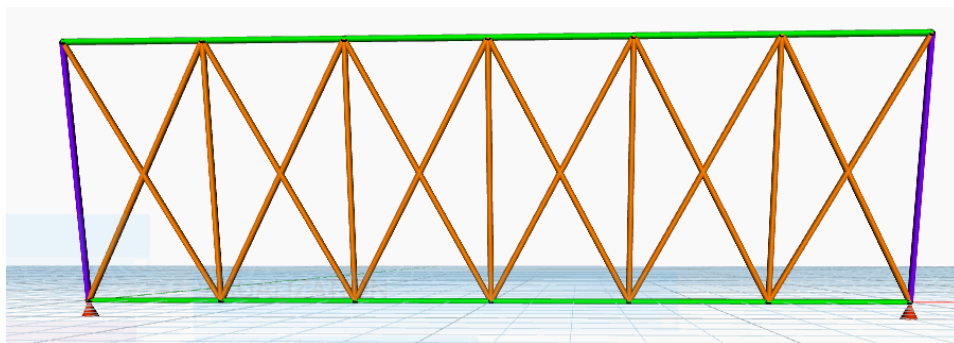


Figure 39. The truss after visualization

3.4. Structural Optimization

As mentioned in the literature review, there are different structural optimization types and the proposed methodology is capable of performing all explained optimization types. To perform shape optimization, the length, height, or in the case of a 3D truss, the width of the truss can be set as a design parameter. That's how the algorithm will try to find the best parameters for those inputs. Similarly, for size optimization, sections, and for topology optimization, the type of the truss can be set as a design variable and optimized. The feature that allows users to do that is parametric modelling because as soon as an input is updated new model is automatically created and FEA can be performed for the new model. While defining the design variables and indirectly type of structural optimization, the core node which will be explained in the next title plays a critical role. But before that, an example of this subject has been presented in Figure 40 for better understanding. The input values in the figure are defined respectively as sections of 3 groups of elements, number of vertical struts, height, and type of truss. As soon as these values are updated, the corresponding weight score will be calculated and added to a list as can be seen in Figure 40. With the aid of parametric modeling, all parameters, including support locations and bar end releases, can be optimized using the Brute Force method, which is thoughtfully explained in section 2.4.3, based on the sections explained information obtained so far. The next sections will explain how to integrate metaheuristic algorithms into the process.

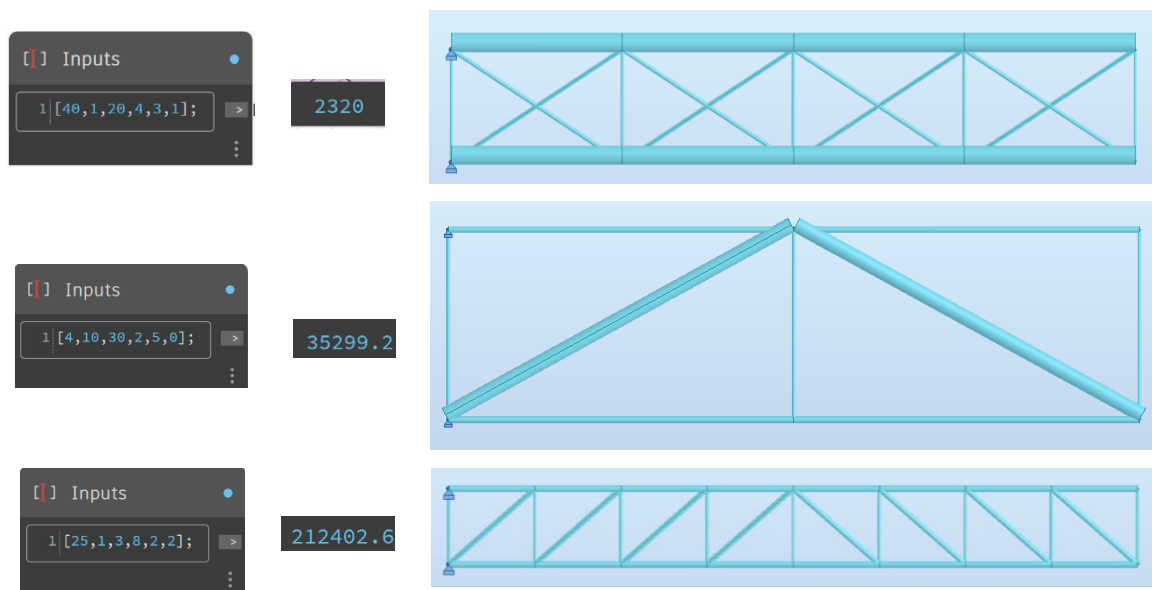


Figure 40. Visualization of trusses created based on inputs

3.4.1. Core Node

To perform optimization, the steps explained in the previous section need to be performed multiple times. To do that Custom node which is explained in section 2.7.4, and the “LoopWhile” node have been used. By creating a custom node, all the nodes in Figure 18 turned into one node as can be seen in Figure 41. This node is the core node of all analysis which takes the parameter that wanted to be optimized with its import port and gives the weight score as output. It will perform the FEM analysis, take the results, evaluate them, and create a weight score over and over. The core node requires multiple inputs and works as an ordinary node in dynamo reducing the size of the graph significantly. Multiple evaluations can be achieved with the core node by enabling users to input static parameters and retaining customizable options.

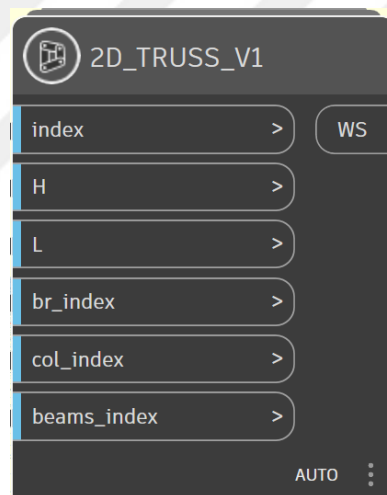


Figure 41. The core node of optimization

3.4.2. Creation of Initial Population

The initial population either can be created randomly or predefined sets of solutions can be given to the Dynamo. In this thesis, a Python script has been prepared to create a random initial population by defining lower-upper limits of solution sets, population size, and chromosome length. There are four input ports for this Python code which are explained in Table 4. The script can be found in text format in Appendix A, Python Scripts section.

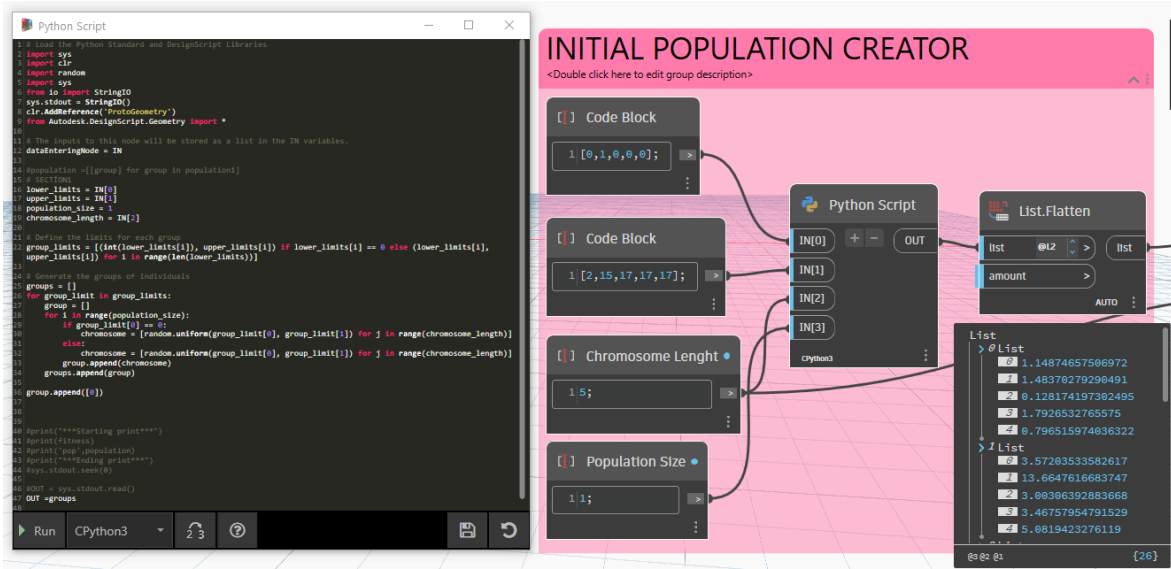


Figure 42. Python script to create an initial population

Table 4. Python inputs and explanations for the initial population creator

Python input	Explanation
IN[0]	The lower limits list of the initial sets
IN[1]	The upper limits list of the initial sets
IN[2]	Chromosome length which is the number of created solution set
IN[3]	Population size: If the magnitude is 1, lower-upper limits define population size but it can be arranged manually too

After creating initial populations, the outputs are connected to the “LoopWhile” node to create a loop for calculating the fitness results of the initial population. Figure 43 illustrates the nodes necessary for this operation. There are two custom nodes in the figure which are “LoopCheck” and “LoopBodyVar1”.

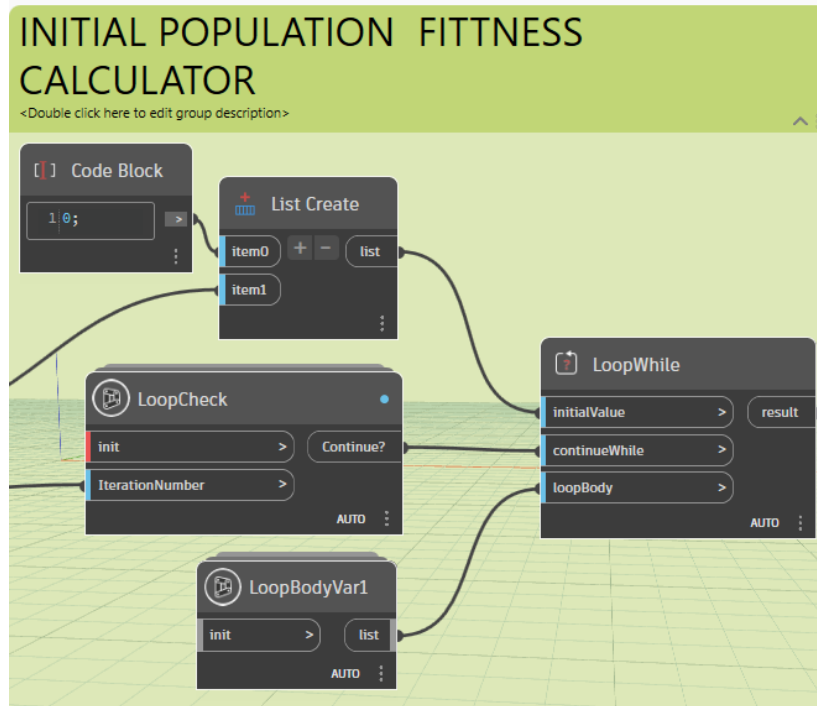


Figure 43. Initial population's fitness calculator

The "LoopCheck" node allows defining the stopping criteria by taking the iteration number and comparing it with the input number. Once the iteration number exceeds the input number, it sends a stop signal to the "LoopWhile" node, and the loop ends. Figure 44 illustrates which nodes created the "LoopCheck" custom node.

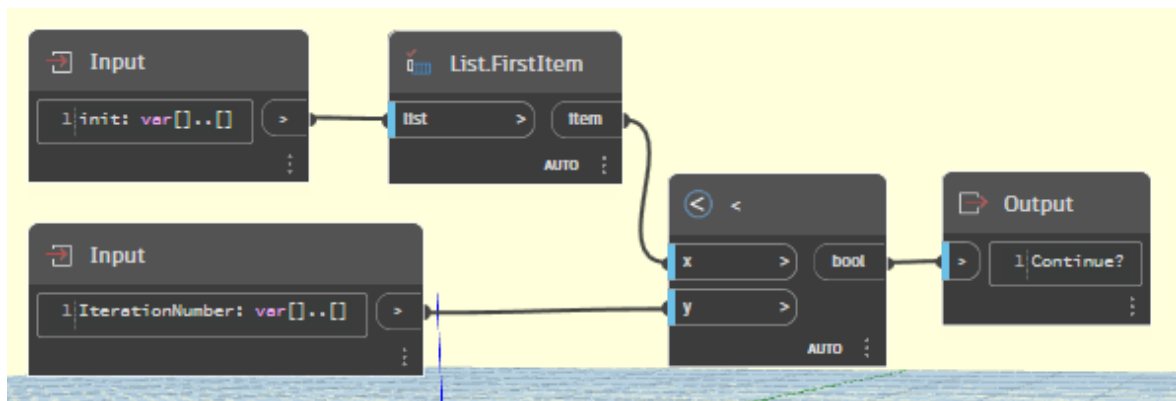


Figure 44. The "LoopCheck" node and explanation

Subsequently, every individual in the initial population list has to be sorted based on their fitness value. That is why the "LoopBodyVar1" custom node is important because it contains the core node inside it which makes all the calculations as can be depicted in Figure 45.

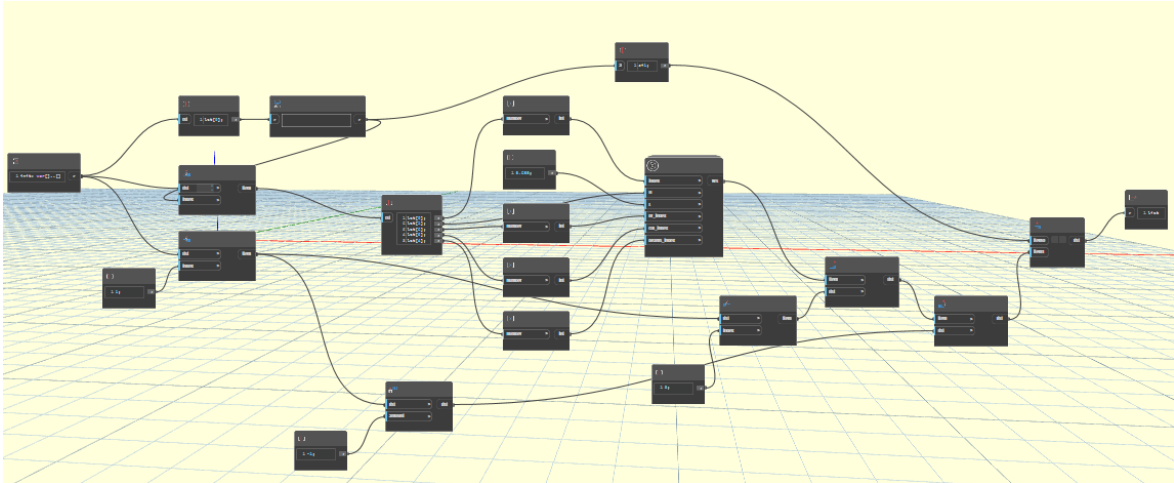


Figure 45. Content of the “LoopBodyVar1” custom node

Here created initial population is chosen based on the iteration number with the help of the “List.GetItemAtIndex” node by connecting the iteration number to the index port which can be seen in Figure 46 and then each solution for different parameters is connected to the core node’s input port after rounding and selection operation which have been shown in Figure 45. Then the weight scores are restored with the information of the solution set and iteration number. In each iteration, the previous iteration’s number is incremented by 1, as depicted in Figure 46. This information is passed to the “LoopCheck” node to meet the stopping criteria.

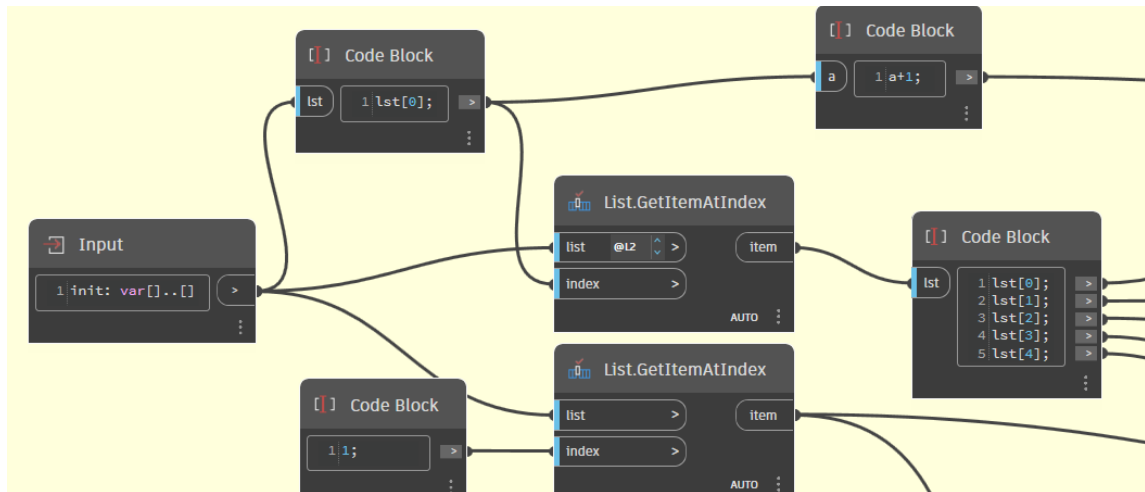


Figure 46. Iteration-Based Input Retrieval

3.4.3. Creating The Parents and Getting the Results

Figure 47 illustrates initial population's fitness values have been obtained. The parents that will create the succeeding generation of the population are selected by using these obtained fitness values and binary tournament selection, which is a selection type where two individuals are chosen at random from the initial population. Then they are compared based on their fitness function values. The individual with the better fitness value is selected as the parent while the other individual is not selected and its genes are eliminated. The chosen parents create the next generation with evolutionary operators. This process is repeated until a goal is met or a certain number of generations have been produced.

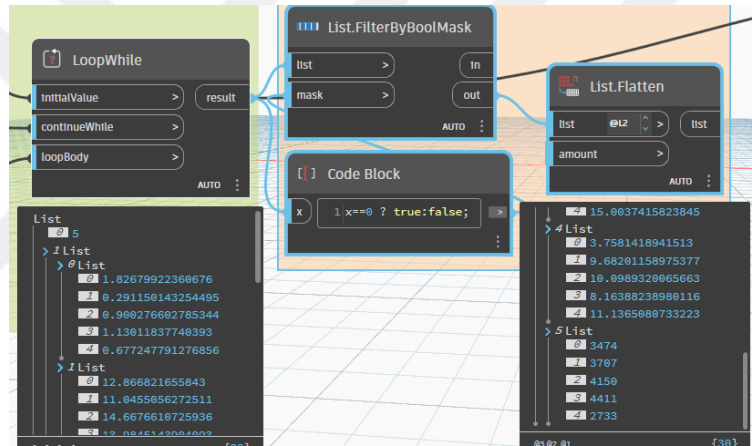


Figure 47. Fitness result of the initial population

Figure 48 shows the nodes that are necessary for GA operators. Except for the “LoopBodyVar2” node other nodes have been explained in previous sections.

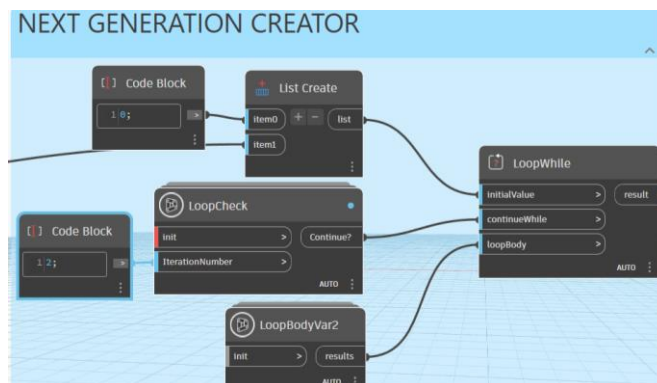


Figure 48. Nodes to run the GA operators

The “LoopBodyVar2” custom node contains two Python scripts and the core node of the script as can be seen from Figure 49. First Python code (see Figure 50) uses GA operators to create new populations based on initial populations and their fitness results by using crossover, mutation, etc. Second Python code on the other hand performs like a selection operator of GA and chooses the best combinations from parents and children.

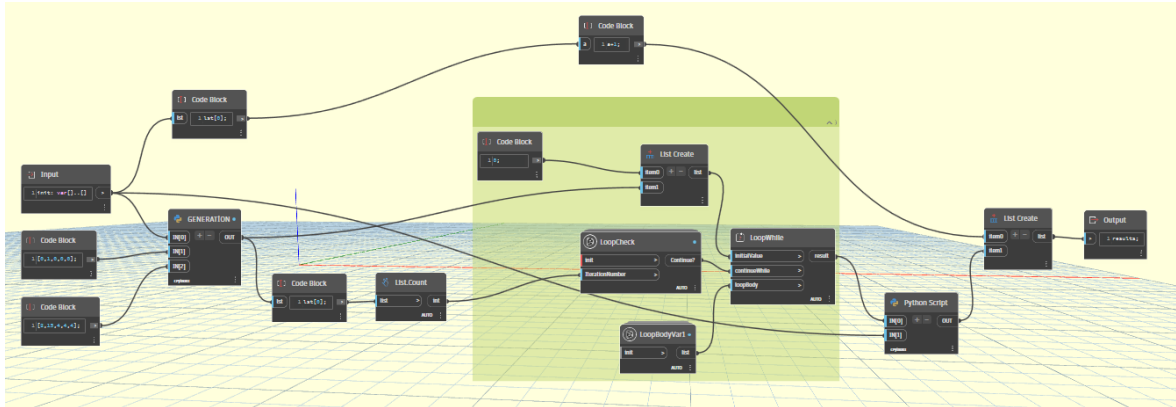


Figure 49. Nodes that “LoopBodyVar2” contains

The python script in Figure 50 retrieves information about the initial population and creates an elite list based on that information. From this created elite list, random individuals are selected using the Python code, and new individuals are generated using the crossover method by using those chosen individuals. In the upcoming section, a brief explanation of utilized GA operators has been given.

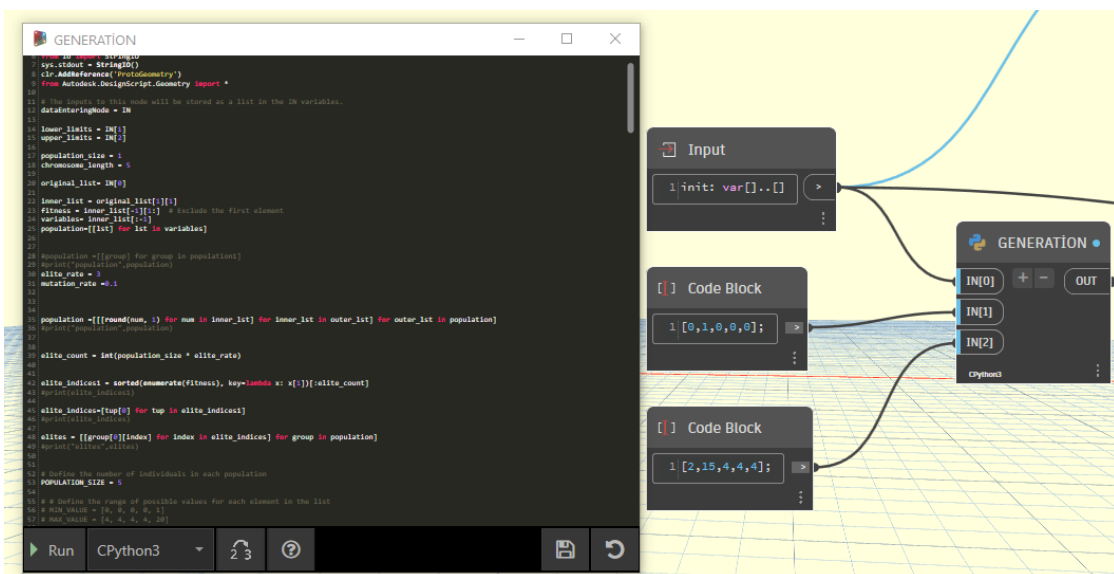


Figure 50. Next population generator

Utilized crossover method: In this thesis, the simple crossover method has not been used because if the numbers are similar to each other, their binary representations will also be similar, and eventually the created individuals will be the same as parents. For example, if the chosen two numbers are 10 and 12, their binary representations will be 1010 and 1100. When a random crossover point is chosen, three out of the four options will result in the same numbers again. In this scenario, changing the second digit's number will be the only way to create a different individual from the parent. Therefore, the created crossover method examines the crossover points that will generate different individuals with the code explained in the section below and randomly selects one of them as the crossover point.

Utilized Selection method: In this thesis, the selection procedure runs with a Python code, which will be explained in the upcoming section, works by choosing an “elite” group that has an adjustable number of individuals based on fitness values. The code takes the initial population and offspring lists and compares them based on their fitness value. In the end, it chooses for example 5 of them to be parents of the new generation.

Utilized deletion method: The deletion method is used to eliminate identical individuals and the individuals exceed lower or upper limits after the crossover process. In certain cases, even if the parents are different, the same new generation can be produced, leading to the search always taking place in the same space. To prevent this, only unique individuals are selected for fitness value calculation.

Utilized Mutation Method: In some cases, after the deletion process, there may not be enough individuals for weight score calculation. In such situations, the mutation operator generates random individuals within predefined lower and upper limits for the optimization process.

These operators are performed by utilizing the Python code given in Figure 51 which is the node named “Generation” in the “LoopBodyVar2” custom node. All the Python codes created, are provided in Appendix A, Python Scripts section. It is important to note that these scripts were developed to evaluate and test the proposed Dynamo script, and due to time limitations, extensive testing was not conducted. In future research, exploring various algorithms like ant colony optimization or particle swarm optimization, as discussed in the literature review section, can be accomplished by making modifications to the Python code and utilizing different operators.

Point by point explanation about marked sections of the code can be found below:

1. **Dynamo inputs:** There are three input ports for the Python code that execute these

1. **Dynamo inputs:** There are three input ports for the Python code that execute these operators, which need to be specified input from Dynamo, as can be seen in Table 5.

Table 5. Python inputs and explanations for the GA generator

Python input	Explanation
IN[0]	The list of initial population and fitness results that belong to the population which is obtained by the initial population creator node group is shown in Figure 47.
IN[1]	The lower limits list represents the lower bounds or limits for each parameter in the sets of the next generation.
IN[2]	The upper limits list represents the upper bounds or limits for each parameter in the sets of the next generation.

2. Elite list creator: Operator that chooses elites from a given population and creates a new list. Also, it contains settings for elit operators like elite rate.

3. Decimal to binary convention: This part defines a function called “binary_to_decimal” that converts decimal numbers to their binary representations. The first step that the code does is checking if the number is negative by looking at the first character. If the first character is “-”, then it marks the number as a negative. After marking the number code deletes the “-” symbol from the binary string. Next step is dividing the number integer and decimal parts to convert them into binary numbers separately. After each of them are converted to the binary representation, the code connects two number with a dot (.) and if the number is negative, puts the negative sign that is deleted in the beginning.

4. Binary to decimal convention: This part of the code defines a function called “binary_to_decimal” that converts a binary number in string format to its decimal representation. The first thing that the code does is check if the number is negative by looking at the first character. If the number is negative, it sets a flag as “True” which indicates the final decimal value will be negative. Then it separates strings from the dot into fraction and integer parts. Then the code converts strings to their decimal representations by using the float() function. After getting two numbers for fraction and integer parts, the code connects them together with a dot and if the number was negative at the beginning the code multiplies the number with -1 to make it negative.

5. Finding different strings for crossover and padding operator: This section of code defines a function called “find_differing_positions” which takes two numbers (num1 and num2) as input and returns a list of positions where the binary representations of the

numbers differ from each other. Also, it adds zeros to number to make the length of the binary representations equal for both numbers. An explanation of how the code works is given below:

The function begins by converting the inputs (num1 and num2) to strings using the `str()` function. This allows for easier manipulation of the binary representations. Then the code finds the length of the longer number between num1_str and num2_str using the `max()` function to ensure that both numbers have the same length. If they are not the same length, the shorter number is padded with zeros using the “`zfill()`” method. After making the length of the numbers same by adding zero at the beginning of binary representations, the code initializes an empty list called “`differing_positions`” to store the positions where the binary representations differ from each other. Then, a loop that compares numbers in the same position in the string is used to iterate over each position from 0 to `max_length - 1`. If the bits at position `i` are different, the index belonging to that position is appended to the “`differing_positions`” list. After comparing all positions, the `differing_positions` list is returned as the output of the function.

6. Crossover operator: This part of the code defines a function called “`binary_crossover`” that performs crossover between two binary-coded individuals. How the code works is the function takes two individuals (individual1 and individual2) as input for the beginning. Then two empty lists, “`offspring1` and `offspring2`”, are initialized to store results after the crossover. Then a loop iterates over each gene in the population. For each corresponding gene from individual1 and individual2, decimal to binary conversion has been made using the function called “`decimal_to_binary`”. After, the binary representation of each gene is split into integer and fractional parts. The differing positions (indices) in the integer and fractional parts are determined using the function called “`find_differing_positions`”. If there are no differing positions, a default index of 0 is added as a crossover point. Then crossover is performed on the integer part by randomly selecting a differing position (index) from the `diff_index_int` list. Then the corresponding bits at the chosen index in the integer parts of individual1 and individual2 are swapped as a crossover operation. The same steps are performed on the fractional part for crossover. At the end of these operations, the modified integer and fractional parts are combined to form new binary genes. Then, the new binary genes are converted back to decimal representation using a function called “`binary_to_decimal`”. The new decimal genes are added to the `offspring1` and `offspring2` lists, respectively. After iterating over all genes, the `offspring1`

and offspring2 lists containing the crossover results are returned as the output of the function.

7. Offspring generator: This part of the code generates offspring populations based on a set of elite individuals by using the functions explained so far. Firstly, the code creates an empty list called offspring. Then two parents are randomly chosen from the elite list using the `random.sample()` function which selects two distinct individuals from the list. Subsequently, the crossover is performed between the selected parents using the “binary_crossover” function, resulting in new offsprings which are appended to the population list. A new list called `unique_population` is initialized to store only unique individuals from the flattened population list as referred deletion method. A loop iterates over each sublist in the population list to check if the sublist is not already present in the `unique_population` list. Then the `unique_population` list is filtered to include only individuals that fall within the range defined by `lower_limits[i]` and `upper_limits[i]`.

8. Mutation applier: This part of the code edits the list generated in the previous section. If the length of the created list in the previous section (`unique_population`) is greater than `chromosome_length`, there are some individuals in the population that need to be removed. Inside the while loop, the code identifies the individuals in `unique_population` that are farthest from the best result. The farthest element is then removed from the `unique_population` list using the `remove()` method. If the length of `unique_population` is less than `chromosome_length`, it means that there is a need to add more individuals to the list. Inside the second while loop, the code appends new individuals which are generated using the `random.uniform()` function within the range defined by `lower_limits[i]` and `upper_limits[i]` to `unique_population` list. If the `unique_population` list contains an individual “0”, it means that a zero value was generated during the process. In such a case, the code finds the index of “0” using the `index()` method and replaces it with a randomly generated individual within the appropriate range. Then the `unique_population` is appended to the offspring list. After some list management and flattening, the final version of the list is assigned as a result which is “OUT”.

After these steps, the Generation node creates new populations by taking into consideration lower and upper bounds as can be seen in Figure 52. Later, with these results, a new generation will be created and they will be sent to the RSA to obtain their fitness result as explained in section 3.3.

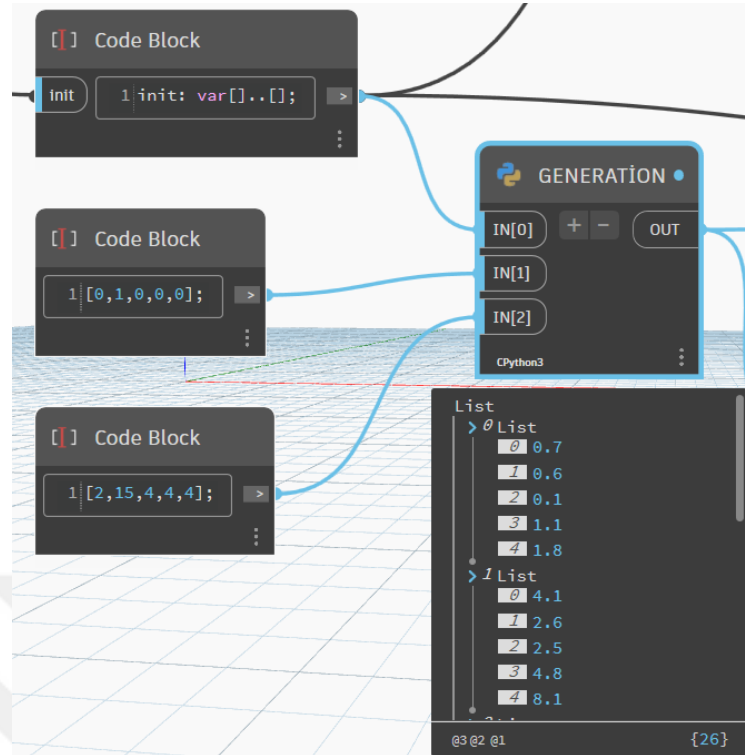


Figure 52. The results of the generation node

3.4.4. Selection Operation

After the previous step, it is necessary to choose the best individuals that will create upcoming offspring from the initial population and old offspring list. For this step, a Phyton code has been created and used as a node named “Selection” as depicted in Figure 53. This node takes the offspring, initial population, and their fitness results as lists and then chooses the best of them to create the upcoming population. The number of individuals that will be chosen here can be adjustable towards the need of the optimization. The Python script has 2 input ports which need to be specified as input from Dynamo. The explanations of these inputs can be seen in Table 6. The text version of this Phyton script can be found in section 9.1.4.

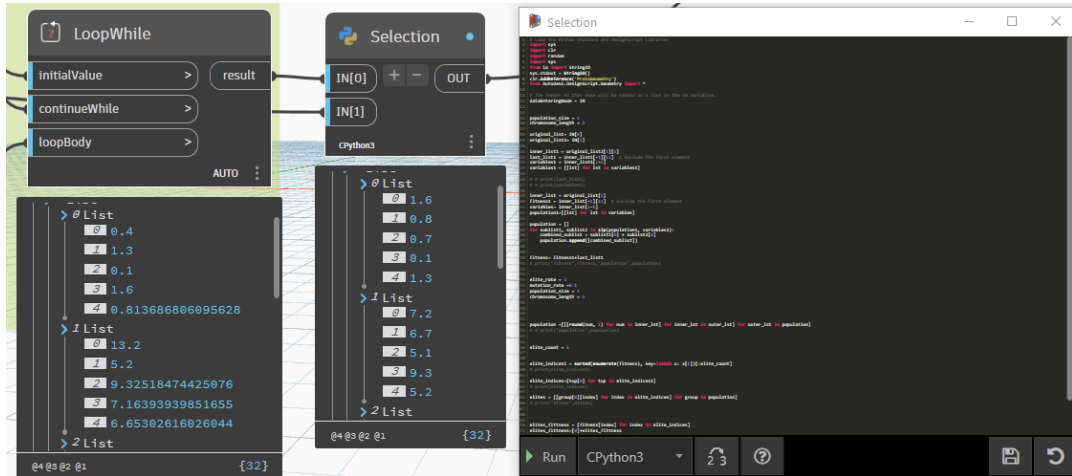


Figure 53. Offspring list and the selection operator

Table 6. Python inputs and explanations for the Selection node

Python input	Explanation
IN[0]	The list of offspring and their fitness result which obtained from the “LoopWhile” node in the “LoopBodyVar2” custom node.
IN[1]	The list of the initial population and their fitness result which obtained from the “LoopWhile” node in the “LoopBodyVar1” custom node.

After getting the best result from the initial population and offsprings, these results become the new initial population and the same process will be repeated in a loop until the stopping criteria is fulfilled. In the end, the results can be seen and read in the nodes in Figure 54.

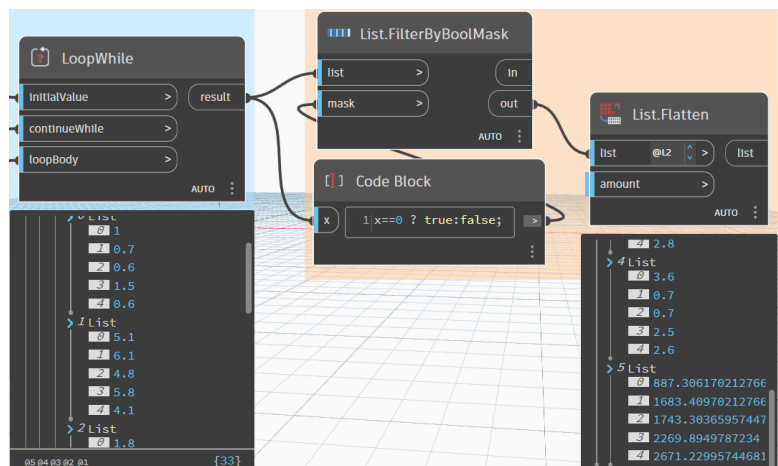


Figure 54. Final results of the optimization

3.5. Revit Importer

In this section advantages of using BIM are given with some examples. To use Revit's features, the optimized model has been exported to Revit by Dynamo script. After creating a model on Revit cost analysis and LCA have been performed and some connection details have been assigned to the model.

3.5.1. Importing the Best Result to Revit

After the optimization process, the proposed script imports the structure into Revit for additional calculations and adjustments to leverage the advantages of BIM. For this task, the “RevitCreator” custom node which can be seen in Figure 55 has been prepared.

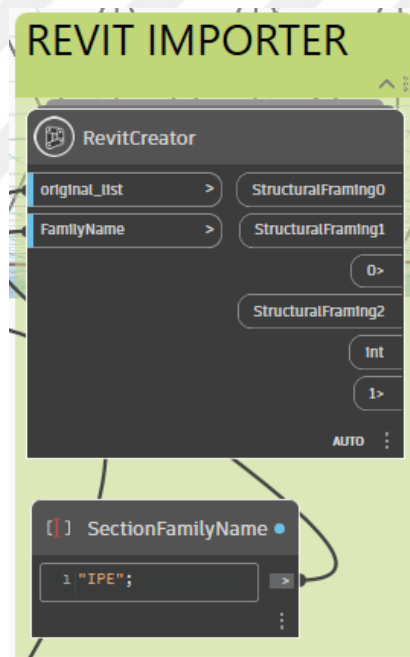


Figure 55. Custom node for importing the structure to Revit

This custom node has 2 input ports. The “LoopWhile” node’s results should be connected to the “original_list” input. The “FamilyName” input is the Family name of the section that is used during the optimization. On the Revit document, section families have to be imported before running this script. To import sections “load Autodesk family” from the insert menu can be used. The “type properties” menu can be used to learn the exact family names of the sections as depicted in Figure 56.

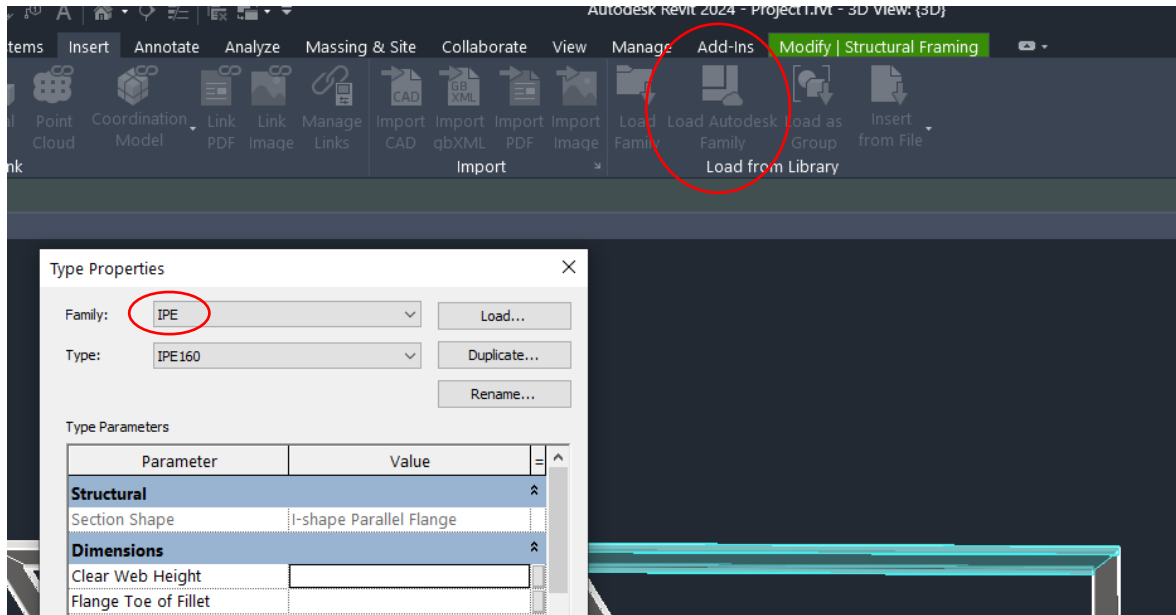


Figure 56. Learning the family name of the cross-sections

Inside the custom node, there is a Python code prepared to choose the best result to import on Revit by its fitness value. The Python code takes “original_list” as input and gives the individual parameters that have the best fitness values. The text format of the code in Figure 57 can be found in section 9.1.5.

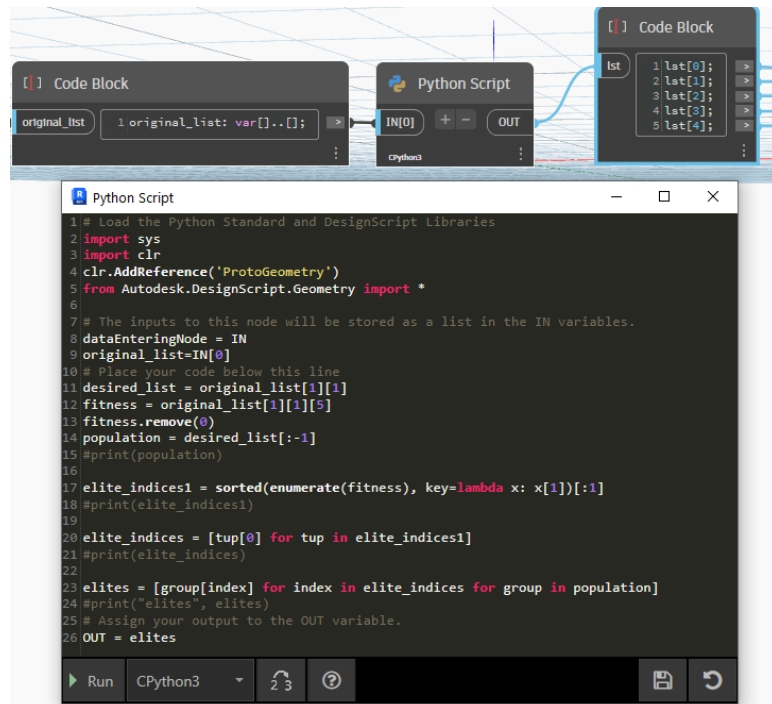


Figure 57. Python script that chooses the best individual and gives its parameter

After these steps, the node does the same thing that is explained in section 3.1.2 with the same nodes. The remaining important step is choosing the right section by the string that is connected to the custom node. To do that, nodes in Figure 58 have been utilized.

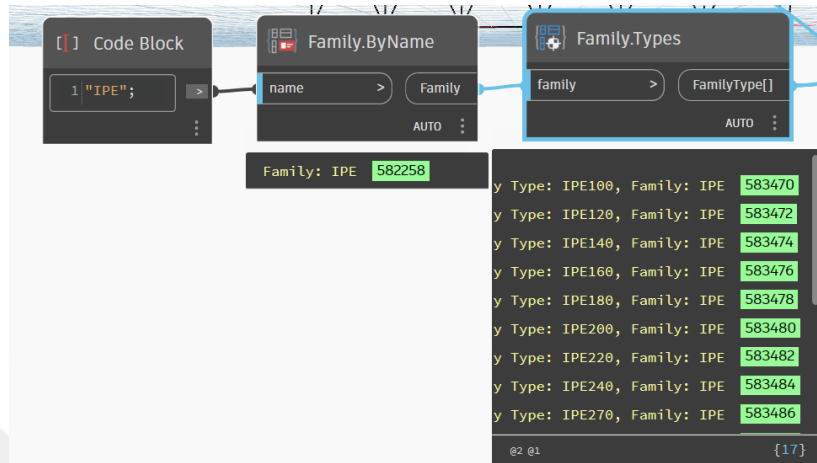


Figure 58. Calling the families by the string

From this family list, by using the “List.GetItemAtIndex” node and the section index information, the right sections are selected and connected to the “StructuralFraming.BeamByCurve” node as shown in Figure 59 which will create the structure on Revit.

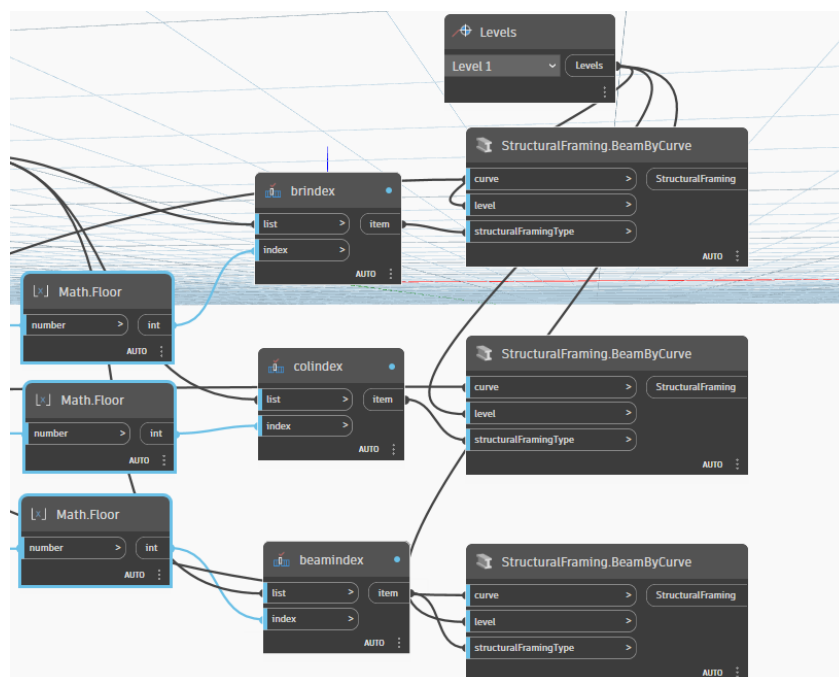


Figure 59. Creating the elements on Revit

After completing these steps and running the script, the optimized structure will be shown on the Revit as depicted in Figure 60.

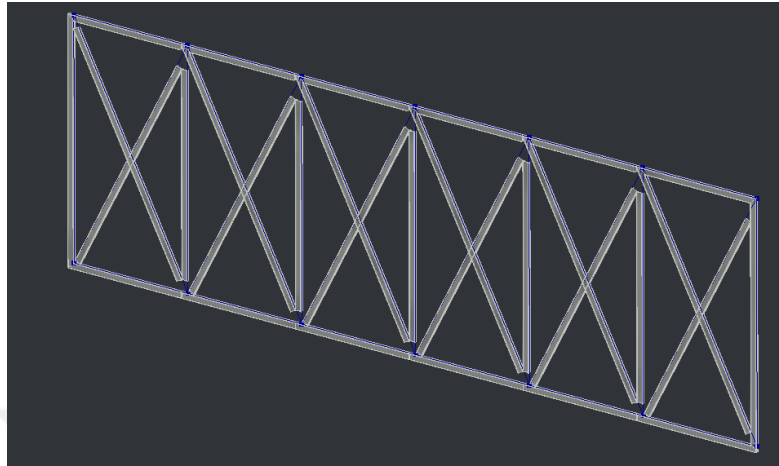


Figure 60. The Revit model of optimized structure

3.5.2. Assembling the Connection Details

The structure after importing the Revit is shown in Figure 61. Revit allows users to design connections and prepare more detailed models. This can be done in two ways. One, choosing the connection points manually on Revit, under the “structure tab”, on the “assembling steel connections” menu. Another way is doing all these steps parametrically with the help of Dynamo. In this thesis, to show the capabilities of BIM and Dynamo, this process is performed on Dynamo with the help of the nodes depicted in Figure 62.

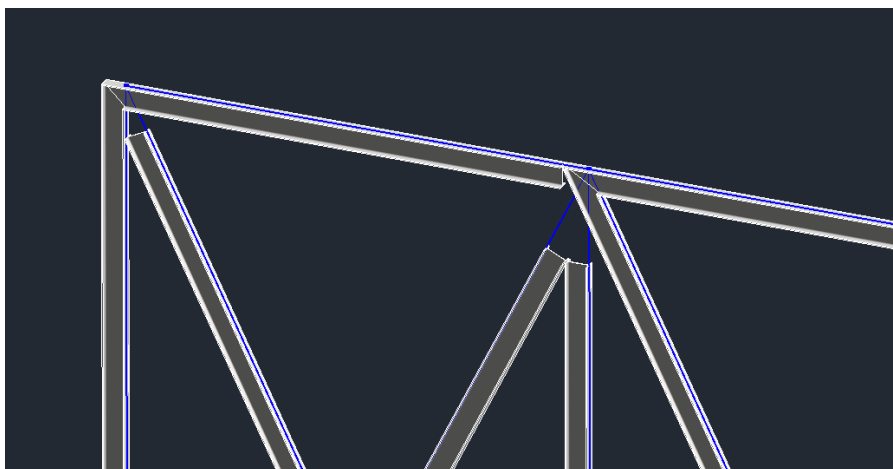


Figure 61. Truss model after importing to the Revit

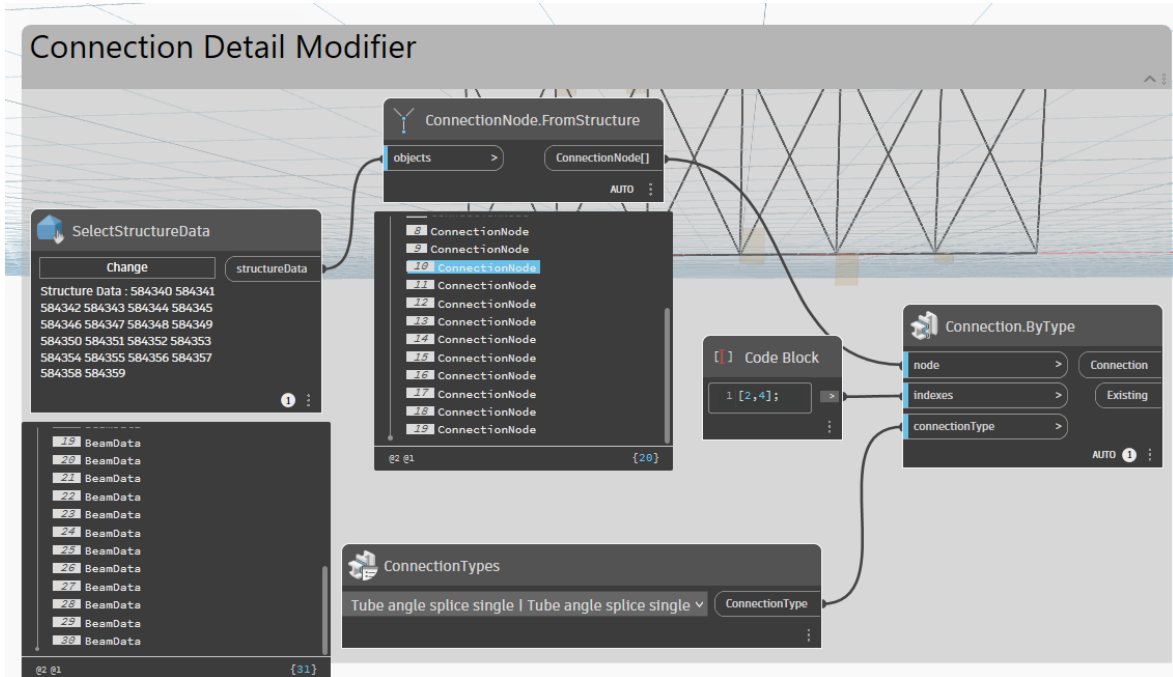


Figure 62. Nodes for adding the connection details

The initial step is to select the elements from the structure in Revit to obtain their connection points by utilizing the “SelectStructureData” node. Once the elements are chosen in Revit, the “ConnectionNode.FromStructure” node will extract the connection points from the structure. For the next step, it is essential to determine the appropriate connection type using the “ConnectionTypes” node. After all steps explained above are established, all the data should be connected to the “Connection.ByType” node which will facilitate the creation of connections for structure in Revit, as shown in Figure 63.

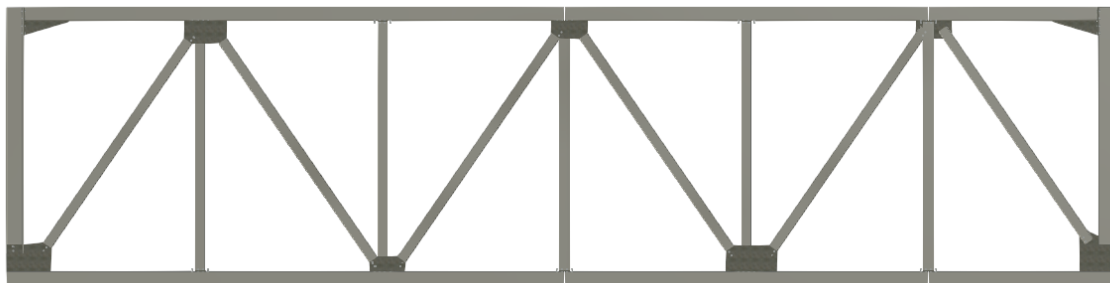


Figure 63. (a) The structure after adding the connection details

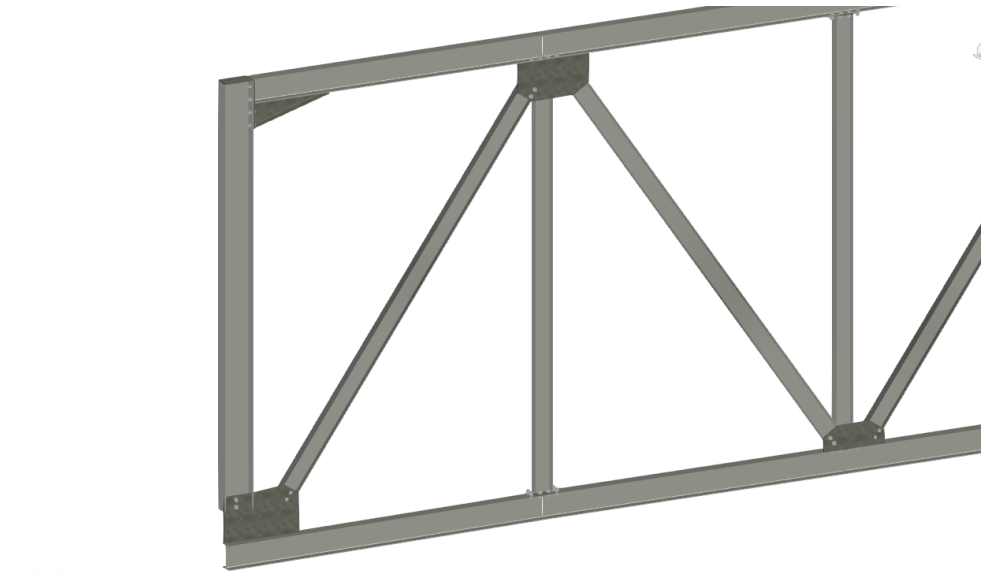


Figure 63. (b) The structure after adding the connection details

3.5.3. Perform LCA and Cost Analysis

This section will demonstrate other advantages, such as obtaining cost and LCA analysis reports using BIM. It will also explain how to generate these reports and provide essential information about both subjects.

3.5.3.1. LCA

After downloading and installing Tally from the Autodesk website, it can be accessed through the Add-Ins Tab, as illustrated in Figure 64. Upon running the app, a window resembling the one depicted in Figure 65 will appear, requiring the selection of the object of study. Once the type of study is selected, all categories, phases, work sets, and linked models that need to be included should be checked, as shown in Figure 66.

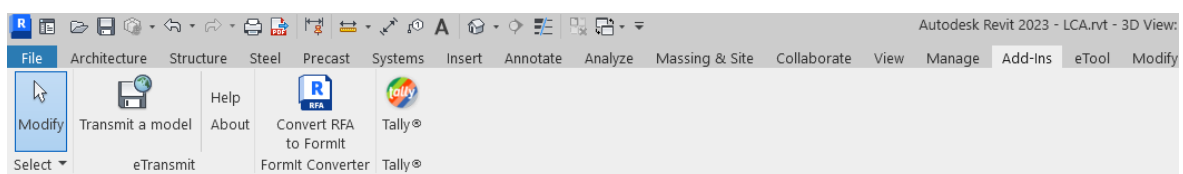


Figure 64. Tally add-ins in the Revit

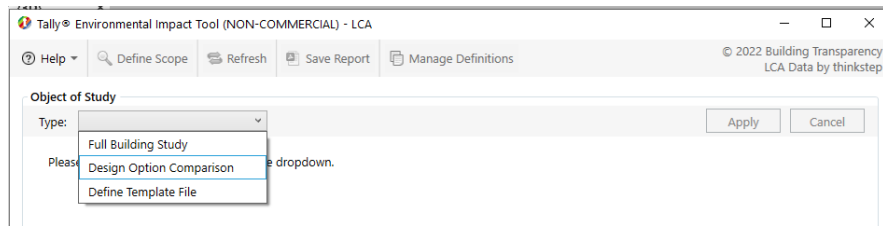


Figure 65. Type of the study for the LCA

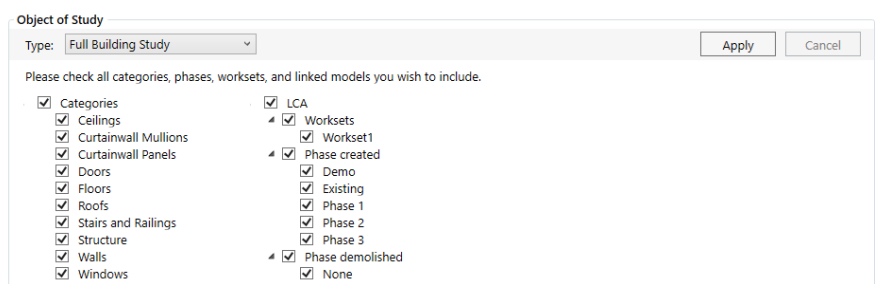


Figure 66. The object of study

Subsequently, all elements that are used in the project are displayed in the app, as depicted in Figure 67. All utilized elements must be defined by right-clicking and selecting "edit definition". Also, it is possible to isolate the chosen element on Revit to see where it is located and what are its properties on Revit. After defining the definitions, the icon next to the elements will turn green. When all icons are green, the app is ready for analysis, and a detailed report can be obtained.

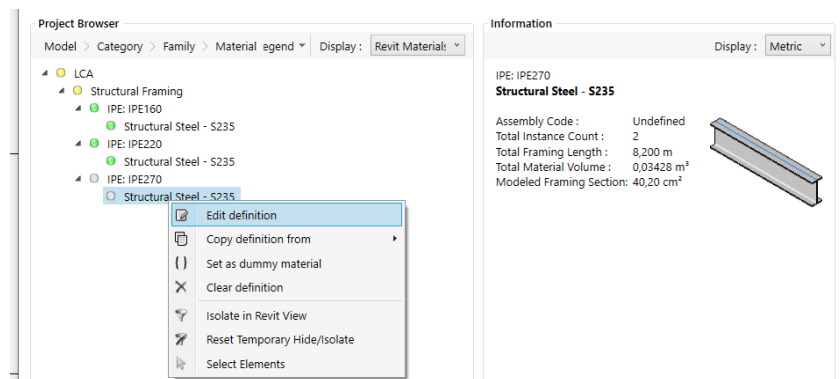


Figure 67. The project browser

Selecting the correct definitions is critical as all calculations will be based on this information. Tally provides a database for the elements that have been utilized. After the appropriate database is chosen, it needs to be clicked "apply" to reach the last section as shown in Figure 68.

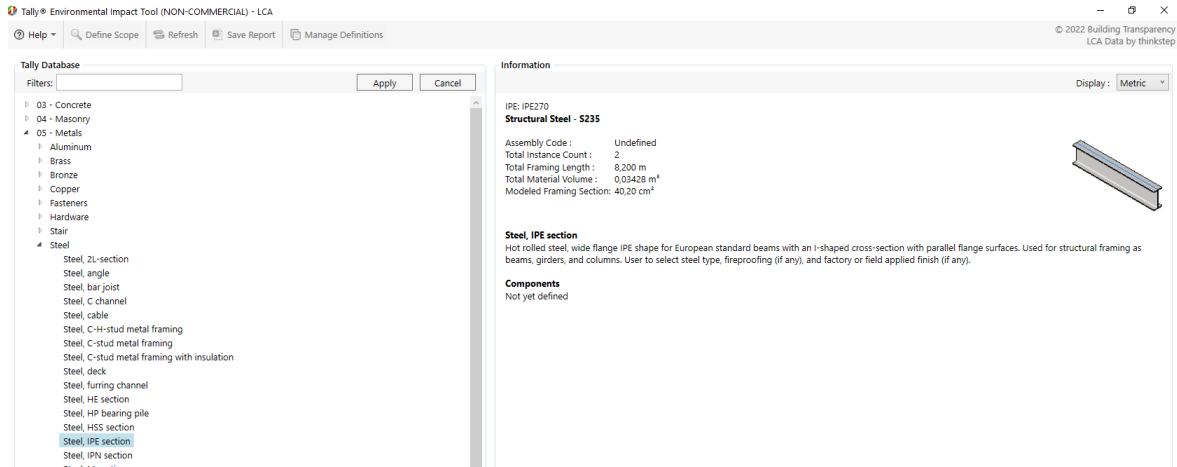


Figure 68. Tally database

The last step is defining components and quantity takeoffs. In this section, it is necessary to specify the steel type, fireproofing, and finish details, as shown in Figure 69. After seeing all the icons are green, a report can be saved to the selected folder.

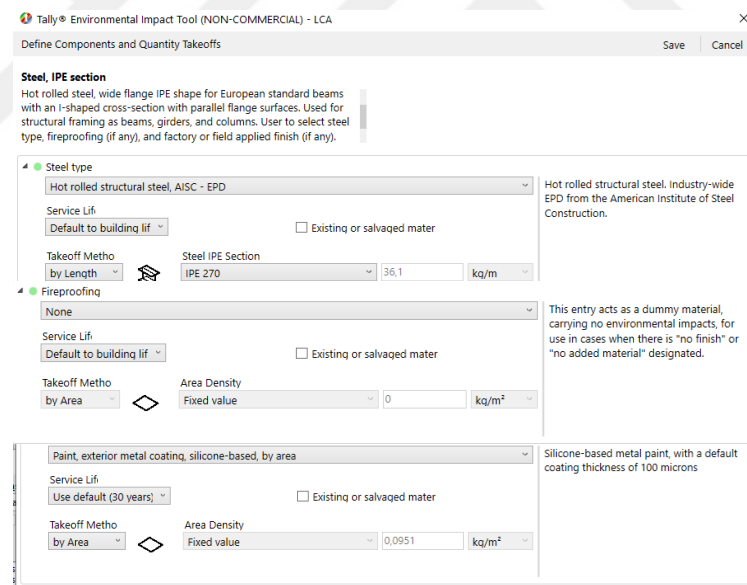


Figure 69. Defining the components and quantity takeoffs

Once the material in a present scope of assessment is completely defined, the results can be obtained by clicking on the “save report” button. Figure 70 shows the table of contents of the created LCA report. This action brings up a window that allows the selection of information to be included in the report. Tally offers several preform contribution assessments organized by life cycle stage, CSI division, and rapid category. Moreover, it provides an itemized bill of materials for generating custom reports directly

from Excel if needed. After entering basic project information, selecting a file name, and clicking "Save," Tally generates a PDF containing all the selected contribution assessments.

Table of Contents	
Report Summary	1
LCA Results	
Results per Life Cycle Stage	2
Results per Life Cycle Stage, itemized by Division	3
Results per Division	4
Results per Division, itemized by Tally Entry	5
Results per Division, itemized by Material	6
Results per Revit Category	7
Results per Revit Category, itemized by Family	8
Results per Building Element	9
Appendix	
Calculation Methodology - Life Cycle Assessment Methods	10
Calculation Methodology - Life Cycle Stages	11
Calculation Methodology - Environmental Impact Categories	12
LCI Data	13

Figure 70. Table of Contents of the LCA

Tally utilizes contribution assessments to demonstrate the connection between products or components that are used in construction and their ecological effects throughout different classifications, for instance, ozone depletion, acidification, and global warming potential. As seen in Figure 71, these assessments are displayed as a number of stacked bar charts with mass and impact category details.

Results per Life Cycle Stage

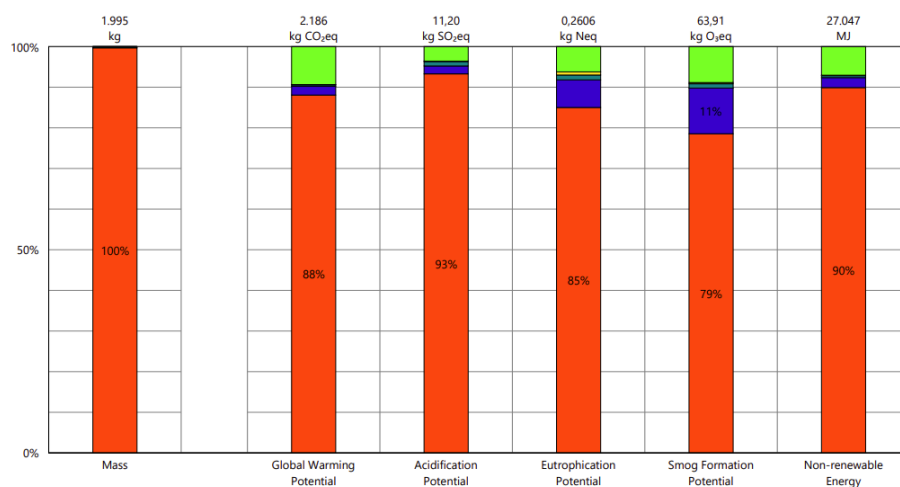


Figure 71. Bar graphs showing mass and impact category breakdowns

For a focused view of global warming potential and embodied energy, pie charts (see Figure 72) are provided on the facing page. When examining a whole building contribution assessment sorted by the CSI division, it becomes evident that steel constitutes 100% of the building's total mass, since the analyzed structure is a simple truss and 88% of its carbon footprint. On the other hand, transportation contributes 2% of the carbon footprint. In order to study impacts based on where materials occur within the building, a contribution assessment by Revit category is available, which can be itemized by family type or by material. This view is especially valuable during the design process as it allows designers to focus on specific assemblies and components in the model and make comparisons between elements that serve similar functions. For a more detailed understanding and LCA analysis, more complex models created by more than one material should be examined.

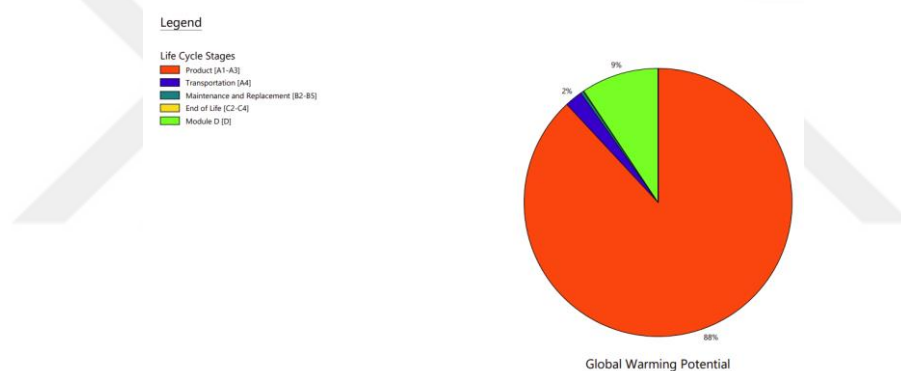


Figure 72. Pie chart of the global warming potential

The report includes an appendix (Figure 73) containing a description of Tally's calculation methodology and a glossary of terminology. Additionally, there is a detailed list of all materials used in the assessment, each containing data about the material's lifecycle inventory entry scope and source, along with the list of all Revit families where it was applied in the model. The full version of the report can be found in Appendix B, LCA Report section.

PRODUCT [A1-A3]

Materials and components are listed in alphabetical order along with a list of all Revit families and Tally entries in which they occur. The masses given here refer to the quantity of each material used over the building's life-cycle, which includes both Product [A1-A3] and Use [B2-B5] stages.

Additional provided data describing scope boundaries for each life cycle stage may be useful for interpretation of the impacts associated with the specific material or component. Each material or component is listed with its service life, or period of time after installation it is expected to meet the service requirements prior to replacement or repair. This value is indicated in parentheses next to the mass of the material associated with the listed Revit family. Values for transportation distance or service life shown with an asterisk (*) indicate user-defined changes to default values. Values for service life shown with a dagger (†) indicate materials identified by the modeler as existing or salvaged.

Hot rolled structural steel, AISC - EPD	1.983,2 kg
Used in the following Revit families:	
IPE	1.983,2 kg (60 yrs)
Used in the following Tally entries:	
Steel, IPE section	
Description:	
Hot rolled structural steel. Industry-wide EPD from the American Institute of Steel Construction.	
Life Cycle Inventory:	
For information and quantities, see EPD	
Product Scope:	
Cradle to gate	
Transportation Distance:	
By truck: 431 km	
End-of-Life Scope:	
98% Recovered	

Figure 73. A section from the appendix of the report

3.5.3.2. Cost Analysis

Tally also generates a bill of materials saved as an Excel file with Pivot Tables corresponding to the same classifications found in the PDF, allowing users to customize their own reporting formats and focus on specific design questions. By adding the cost information to this file and multiplying it with the mass, the total cost of the structure can be found. But in this thesis, the Revit schedules/Quantities section has been used to calculate the cost. In the project browser or “View tab/Create panel/Schedules drop-down/Schedule/Quantities” path new schedule can be created. Then it is necessary to adjust schedule properties. Fields are an important section of properties because in there the parameters that are wanted to be shown in the final bill of quantities table can be chosen and put in order as shown in Figure 74.

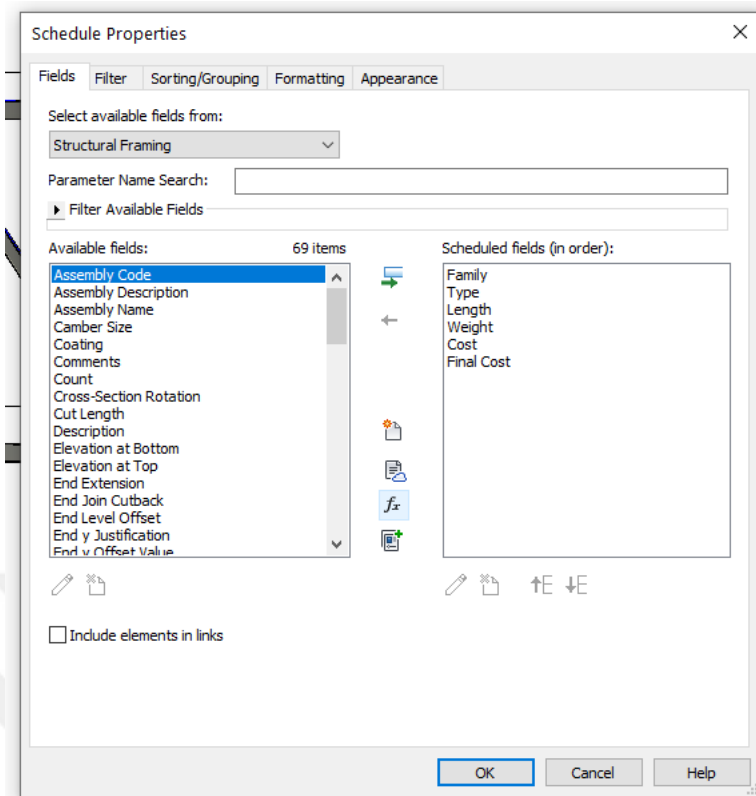


Figure 74. Schedule properties

Also, calculation parameters can be added to the field. As an example, the final cost parameter has been added to calculate the total cost of the building. To calculate the cost of the steel, material weight and the unit kg cost of steel have to be multiplied and this “fx” symbol shown in Figure 74 allows us to add this kind of calculation in the schedules as depicted in Figure 75.

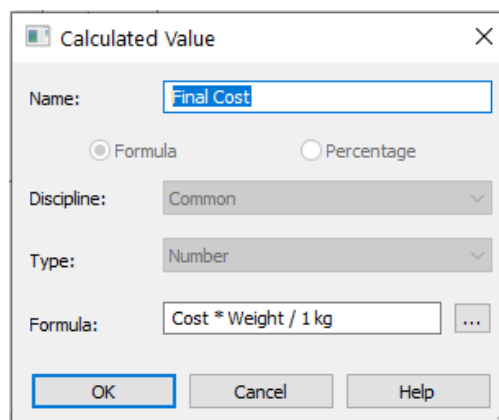


Figure 75. Adding calculation parameter

After organizing all the properties, the bill of quantities table can be generated, as depicted in Figure 76. This table offers the flexibility to be customized and adapted according to the specific requirements of the project, allowing for a more intricate representation and also it can be exported in Excel format if needed.

<Structural Beams & Bracing Quantities>					
A	B	C	D	E	F
Structural Member	Type	Length	Weight	Cost	Final Cost
IPE					
IPE	IPE160	4.100	62.10 kg	0.51	31.672861
IPE	IPE160	4.100	62.10 kg	0.51	31.672861
IPE	IPE160	4.100	62.10 kg	0.51	31.672861
IPE	IPE160	4.100	62.10 kg	0.51	31.672861
IPE	IPE160	4.100	62.10 kg	0.51	31.672861
IPE	IPE160	4.984	66.47 kg	0.51	33.899802
IPE	IPE160	4.984	72.35 kg	0.51	36.896217
IPE	IPE160	4.984	68.92 kg	0.51	35.148309
IPE	IPE160	4.984	70.88 kg	0.51	36.147114
IPE	IPE160	4.984	69.02 kg	0.51	35.198249
IPE	IPE160	4.984	64.71 kg	0.51	33.000878
IPE160: 11		50.402	722.85 kg		368.654873
IPE	IPE220	2.833	73.09 kg	0.58	42.393526
IPE	IPE220	2.833	75.56 kg	0.58	43.826875
IPE	IPE220	2.833	75.56 kg	0.58	43.826875
IPE	IPE220	2.833	75.56 kg	0.58	43.826875
IPE	IPE220	2.833	75.56 kg	0.58	43.826875
IPE	IPE220	2.833	70.15 kg	0.58	40.689003
IPE	IPE220	2.833	73.49 kg	0.58	42.625977
IPE	IPE220	2.833	75.56 kg	0.58	43.826875
IPE	IPE220	2.833	75.56 kg	0.58	43.826875
IPE	IPE220	2.833	75.56 kg	0.58	43.826875
IPE	IPE220	2.833	75.56 kg	0.58	43.826875
IPE	IPE220	2.833	77.77 kg	0.58	45.105283
IPE220: 12		34.000	899.02 kg		521.428789
IPE	IPE270	4.100	135.28 kg	0.58	78.464505
IPE	IPE270	4.100	140.44 kg	0.58	81.453631
IPE270: 2		8.200	275.72 kg		159.918135
IPE: 25		92.602	1897.59 kg		1050.001797
Grand total: 25		92.602	1897.59 kg		1050.001797

Figure 76. The bill of quantities table in the Revit

4. EXAMPLES AND RESULTS

This section presents the numerical outcomes for several test problems. A comparison is made between the solution obtained using the genetic algorithm by Dynamo which is created in this thesis and solutions derived from various contemporary algorithms found in the open literature. Table 7 provides an overview of the size and characteristics of the individual problems investigated in this section. It includes information on the number of elements and nodes, as well as details on the initial population and the number of iterations for each problem. These examples are tested with the proposed framework using a computer with features listed in Table 8. The average time spent on 100 analyses with the proposed framework and the computer with features listed in Table 8 is approximately 2 hours. It is assumed that this time will be reduced with a computer having better hardware properties.

Table 7. Information about the design problems

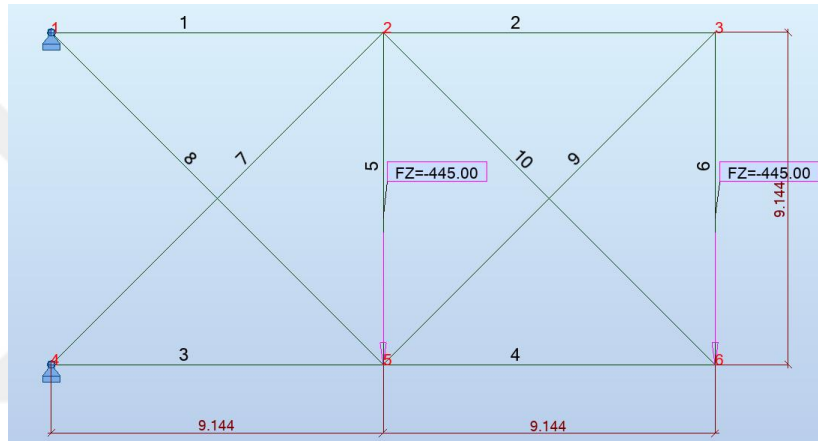
No	Problem name	Buckling	No. of elements	No. of nodes	Initial pop. number	Max iteration number	No. of separate Runs
1	10-bar 2D	No	10	6	20	20	4
2	36-bar 3D	No	36	12	20	80	10
3	120-bar 3D	Yes	120	37	20	20	4

Table 8. Device specifications

Processor	: Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz 2.60 GHz
Installed RAM	: 16,0 GB (15,9 GB usable)
System type	: 64-bit operating system, x64-based processor
Windows edition	: Windows 10 Home Single Language

4.1. 10-Bar 2D Truss

The 10-bar truss example is a standardized test case in the field of structural optimization for those who want to evaluate and validate the effectiveness of proposed optimization techniques, and it is frequently used as a test problem by researchers (Camp & Bichon, 2004; Cao et al., 2017; Goldberg & Samtani, 1986; Groenwold et al., 1999; Rajeev & Krishnamoorthy, 1992). The geometry, support requirements, material properties, and boundary conditions for this 2D hanging truss with loading conditions are given in Figure 77.



Material: 6.89×10^4 MPa
 Loads: P1, P2= 444822.16 N
 Density: 2.770 kg/m^3
 Displacement limit: 50.8mm

Figure 77. 10-bar truss

There are several optimization problems for the 10-bar truss that use distinct cross-sectional area ranges and design constraints. In this thesis, the example and design properties have been adopted from the study prepared by Rajeev et al. (Rajeev & Krishnamoorthy, 1992). That is why constraints are, stresses on member which is limited to 172.25 MPa, and displacements on nodes which is limited to 50.8 mm. A discrete list that contains 41 cross-sectional areas has been prepared and presented in Table 9. These sections and areas have been sourced from the American Institute of Steel Construction (AISC), like in other studies (Arora, 1989; Rajeev & Krishnamoorthy, 1992). The sections imported to RSA from the AISC database (AISC Edition 15.0 American hot rolled shapes) and HSRO (Hollow Structural Round Sections) family were used during the optimization. Detailed information about importing databases and using different section properties can

be found in the referenced Autodesk website (URL-23, 2023). The aim of this example is to find suitable sections for 10 elements in structure from the list that have 41 different cross-sectional areas that do not violate the constraints.

Table 9. Discrete cross-sectional area list and names

Number:	Name of Section	Cross-section area (cm ²)
1	HSRO 4.5x.125	10.32
2	HSRO 3.5x.188	11.74
3	HSRO 3.5x.203	12.71
4	HSRO 5.563x.134	13.68
5	HSRO 6.625x.125	15.3
6	HSRO 4x.237	16.83
7	HSRO 4x.25	17.80
8	HSRO 3.5x.313	18.90
9	HSRO 4.5x.237	19.1
10	HSRO 6x.188	20.52
11	HSRO 4x.313	21.87
12	HSRO 5x.25	22.51
13	HSRO 6.625x.188	22.77
14	HSRO 6.875x.188	23.61
15	HSRO 7x.188	24.06
16	HSRO 7.5x.188	25.8
17	HSRO 5x.258	26.61
18	HSRO 4.5x.337	26.58
19	HSRO 6x.25	27.23
20	HSRO 5x.312	27.74
21	HSRO 4.5x.375	29.35
22	HSRO 6x.28	30.26
23	HSRO 5x.375	32.9
24	HSRO 9.625x.188	33.35
25	HSRO 5.563x.375	36.9
26	HSRO 6.875x.375	46.2
27	HSRO 7.625x.375	51.48
28	HSRO 16x.25	76.19
29	HSRO 9.625x.5	86.45
30	HSRO 10x.5	89.68
31	HSRO 16x.312	92.90
32	HSRO 14x.375	96.77
33	HSRO 10x.625	110.96
34	HSRO 12.75x.5	115.48
35	HSRO 18x.375	125.16
36	HSRO 16x.438	128.39
37	HSRO 20x.375	138.7
38	HSRO 16x.5	146.45
39	HSRO 18x.5	165.16
40	HSRO 16x.625	181.29
41	HSRO 20x.5	183.9

The parameters like population size, generation number, elite rate, mutation rate, etc. are really important for GA to reach the optimum solution earlier as explained before. However, since the aim of this section is to test the proposed methodology and compare it with the literature, a population size of 20 was used for the runs, and calculations were continued for 20 generations, similar to Rajeev et al (Rajeev & Krishnamoorthy, 1992). The results found are shown and compared with the studies taken from the literature are presented in Table 10. The table shows that the solution's weight obtained from this thesis, which is 2838.510 kg, is higher than the solutions obtained from previous studies. However, increasing number of runs and making small changes to the GA algorithm used in the thesis and optimizing settings such as the elite rate, mutation rate, and selection algorithm is expected to enhance its ability to find better solutions. Moreover, since the aim of the study is to demonstrate the usability of the proposed methodology, the solution obtained is acceptable because the study conducted by Camp et al. required an average of approximately 10,000 truss analyses (Camp & Bichon, 2004) to converge the solution by the created algorithm, which is 25 times higher than analysis number that has been made for this thesis. It is also important to consider that, in the study conducted by Cao et al. (Cao et al., 2017), 20 separate runs were required to reach the solution depicted in Table 10, whereas only 4 separate runs were made to reach the result in this thesis. Additionally, different cross-sectional list has been used which allows to assign sections area lower than previous studies.

Table 10. Comparison of 10-bar truss results

	Rajeev et al. 1992	Groenwold et al. 1999	Camp et al. 2004	Cao et al. 2017	This Thesis
Weight (kg)	2549.225	2490.557	2490.557	2299	2838.510
Analysis	400	400	10000	4872	400
A1 (cm²)	216.5	216.5	216.5	197.27	165.16
A2 (cm²)	10.45	10.45	10.45	0.645	29.35
A3 (cm²)	141.94	146.45	146.45	150.89	138.71
A4 (cm²)	100.08	92.90	92.90	95.73	96.77
A5 (cm²)	10.45	10.45	10.45	0.645	25.81
A6 (cm²)	11.61	10.45	10.45	3.29	25.81
A7 (cm²)	128.29	146.45	146.45	136.74	128.39
A8 (cm²)	91.61	51.48	51.48	48.19	110.97
A9 (cm²)	16.92	10.45	10.45	0.645	29.35
A10 (cm²)	128.29	138.7	138.7	138.71	165.16

In addition to the iteration table, weight score change, which is the best weight score found for each generation, is also provided in Figure 78 to show the weight score change by each generation. For the final step, the structure with the optimized sections and the detailed information have been presented in Figure 79.

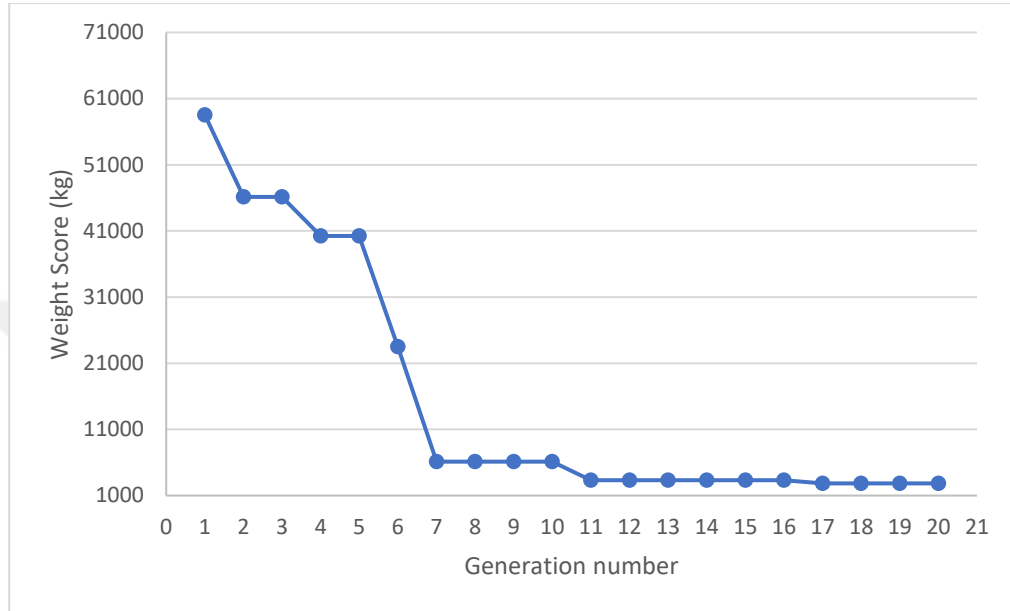


Figure 78. Weight score change for 10-bar truss by generation number

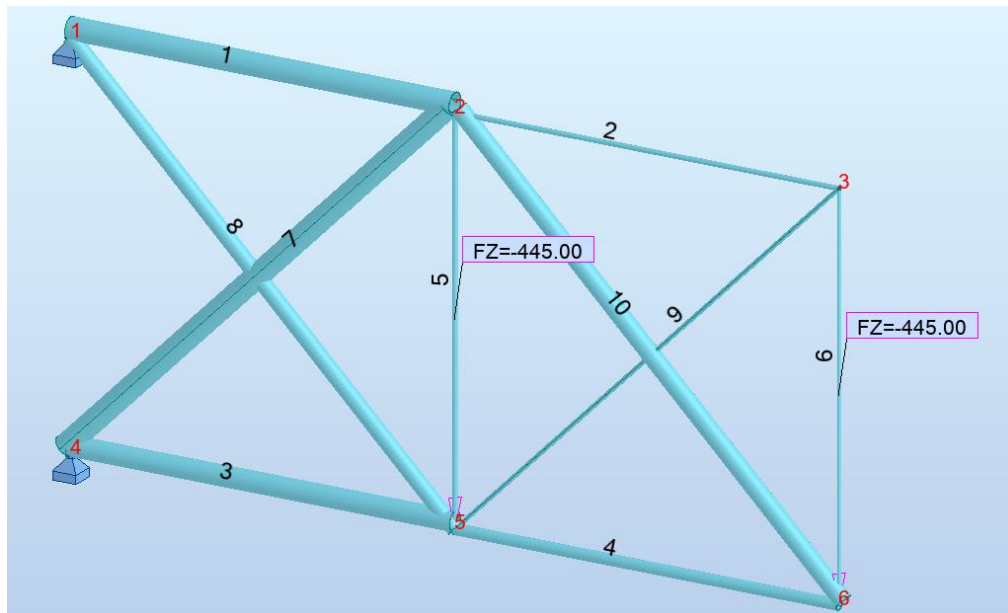
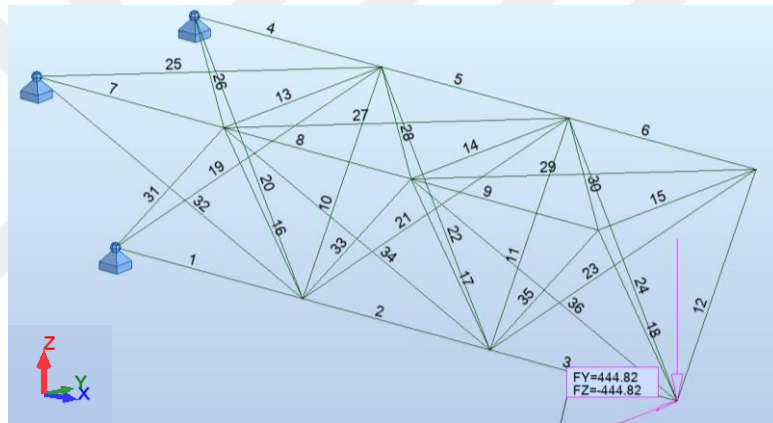


Figure 79. The final version of the 10-bar truss

4.2. 36-Bar 3D Truss

The 36-bar truss problem has emerged as another benchmark frequently used as a test problem among scholars (Groenwold et al., 1999; Ringertz, 1988; Schutte & Groenwold, 2003). The configuration, dimensions, support requirements, material properties, and the loading condition of this three-dimensional truss are shown in Figure 80. The constraints are stresses on member which is limited to 172.25 MPa and displacements are limited to 50.8 mm for node 4 in both negative z and y direction. Moreover, used Dynamo nodes and script to create this truss geometry are given in Figure 81. A copy of a high-resolution “.png” photo of the script has been added to Appendix C, Dynamo Scripts section.



Material: 68947.57 MPa
 Loads: FY, FZ= 444822.16 N
 Density: 27.80 kN/m³
 Length of truss=27.42m
 Height of truss= 7.92m
 Width of top section =4.57m
 Displacement limits: 50.8mm in the Y direction
 50.8mm in the -Z direction

Figure 80. 36-bar truss

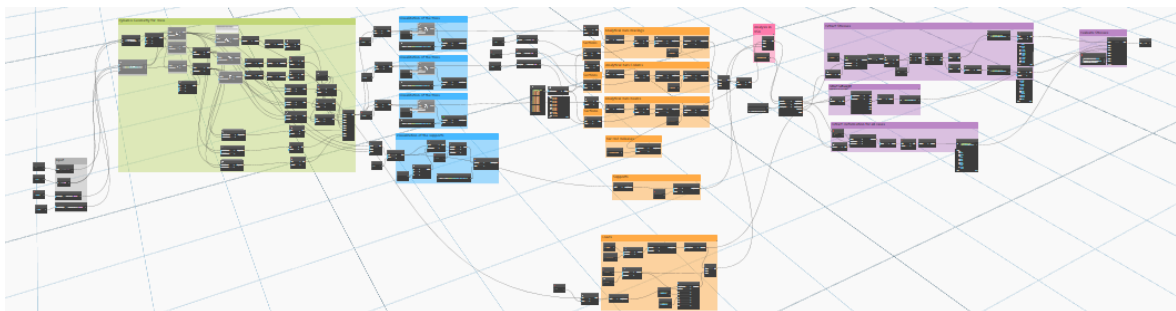


Figure 81. The script for creating the 36-bar truss

Since the authors did not choose the cross-sectional areas from specific regulations in the study taken from the literature, areas that are similar to ones that are used in the literature have been sourced from the AISC for this thesis and they are imported to RSA from AISC database (AISC Edition 15.0 American hot rolled shapes) and RB (Round Bars) family used during the optimization. This discrete list that contains 25 cross-sectional areas has been prepared and presented in Table 11. The aim of this example is to find suitable sections for 36 elements in a structure divided into 21 groups from a list that contains 25 different cross-sectional areas, all of which must not violate the constraints. The grouping of elements which can be found in Table 12 and the configuration of structure are taken from the referenced article (Ringertz, 1988).

Table 11. Discrete cross-sectional area list and names

Number	Name of Section	Cross-section area (cm ²)
1	RB 12	729.66
2	RB 11.9375	722.08
3	RB 11.875	714.54
4	RB 11.8125	707.04
5	RB 11.5625	677.42
6	RB 11.3125	648.45
7	RB 11	613.12
8	RB 10.75	585.56
9	RB 10.375	545.42
10	RB 10.125	519.45
11	RB 9.8125	487.88
12	RB 9.4375	451.31
13	RB 9.125	421.91
14	RB 8.75	387.95
15	RB 8.375	355.41
16	RB 8	324.29
17	RB 7.625	294.60
18	RB 7.125	257.23
19	RB 6.6875	226.61
20	RB 6.1875	193.99
21	RB 5.625	160.33
22	RB 5.0625	129.86
23	RB 4.375	96.99
24	RB 3.5625	64.31
25	RB 2.5625	33.27

To test the proposed methodology and compare it with the literature, a population size of 20 was used for the runs, similar to studies in the literature (Groenwold et al., 1999; Ringertz, 1988). The areas found for 36 elements of truss and their comparison with literature results are depicted in Table 12. The table indicates that the weight of the solution obtained in this thesis, 17913.830 kg, closely aligns with values reported in the literature. However, the maximum iteration number, which is 80, exceeds the numbers found in the literature. This shows that the coded GA encounters difficulties when dealing with optimization problems that involve a relatively high number of design variables.

Table 12. Comparison of 36-bar truss results

Design Variables	Elements	Ringertz 1988	Groenwold et al. 1999	Schutte et al. 2003	This Thesis
Weight (kN)	-	16517.566	16368.334	16181.77	17913.830
A1 (cm ²)	1	258.064	258.064	246.06	257.23
A2 (cm ²)	2	161.290	161.290	161.32	193.99
A3 (cm ²)	3	64.516	32.258	44.27	33.27
A4 (cm ²)	4,7	580.644	612.903	607.20	613.12
A5 (cm ²)	5,8	354.839	354.839	384.65	355.41
A6 (cm ²)	6,9	96.774	96.774	93.44	96.99
A7 (cm ²)	10,16	32.258	32.258	32.26	33.27
A8 (cm ²)	11,17	32.258	32.258	32.26	33.27
A9 (cm ²)	12,18	96.774	96.774	93.54	96.99
A10 (cm ²)	13	32.258	32.258	32.26	33.27
A11 (cm ²)	14	32.258	32.258	32.26	64.31
A12 (cm ²)	15	32.258	32.258	32.26	33.27
A13 (cm ²)	19,31	193.548	193.548	177.28	193.99
A14 (cm ²)	20,32	193.548	193.548	182.08	193.99
A15 (cm ²)	21,33	193.548	161.290	178.72	160.33
A16 (cm ²)	22,34	193.548	193.548	177.95	193.99
A17 (cm ²)	23,35	193.548	193.548	184.17	226.61
A18 (cm ²)	24,36	193.548	193.548	184.72	193.99
A19 (cm ²)	25,26	32.258	32.258	32.26	64.31
A20 (cm ²)	27,28	32.258	32.258	32.26	33.27
A21 (cm ²)	29,30	32.258	32.258	32.26	96.99

Furthermore, Figure 82 displays the weight score history by generation, while Figure 83 presents the final version of the optimized structure with optimum sections found along with detailed information.

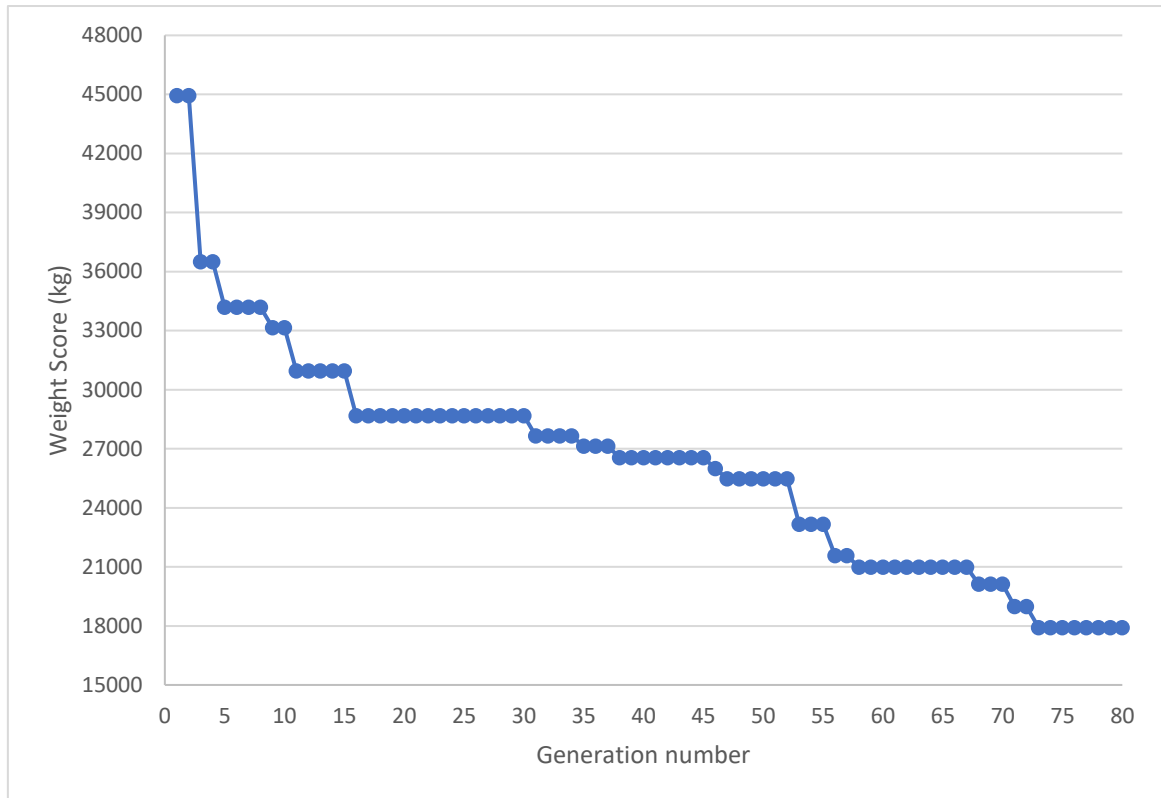
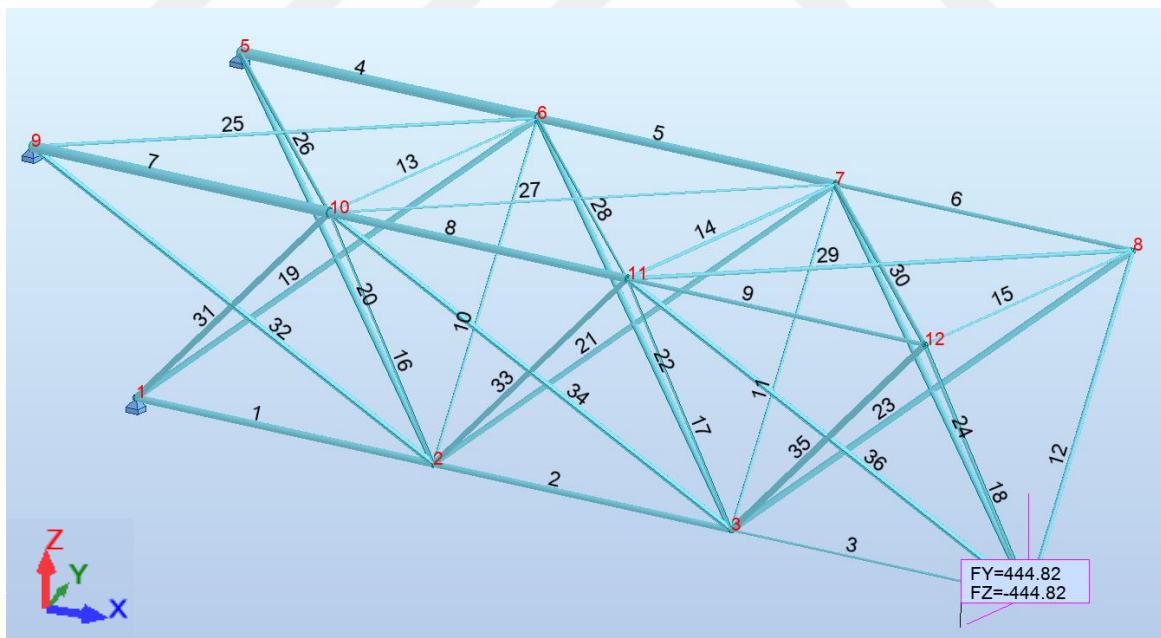


Figure 82. Weight score change for 36-bar truss by generation number

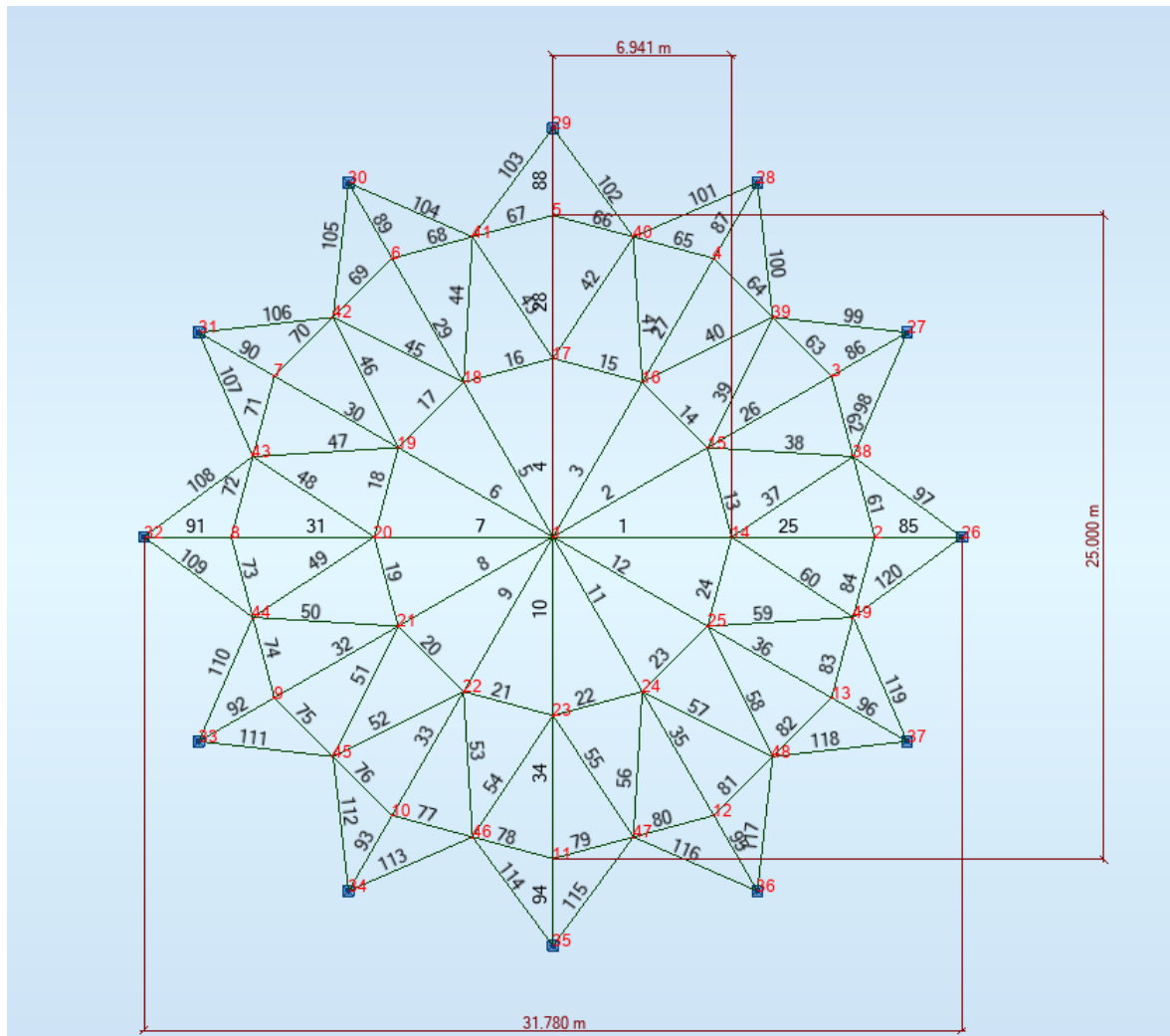


Total weight: 17913.830 (kg)
 Weight score: 17913.830 (kg)
 Maximum deflections on node 4: UZ = -48.25 (mm)
 UY = 50.02 (mm)
 Maximum stress: 51.95 (MPa)

Figure 83. The final version of the 36-bar truss

4.3. 120-Bar Dome Truss

The 120-bar dome truss was first solved by M.P. Saka in 1991. Then other researchers used that structure to test their algorithm (Azizi et al., 2022; Capriles et al., 2006; Ebenau et al., 2005; Kooshkbaghi & Kaveh, 2020; Saka & Ulker, 1991; Tejani et al., 2016). The configuration, dimensions, support requirements, material properties, and the loading condition of this three-dimensional dome truss are shown in Figure 84.



Material: 210000 MPa

Loads: Node 1 = 60 kN

Node 2-14 = 30 kN

Node 15-37 = 10 kN

Density: 7860 kg/m³

Height of top point = 7 m

Height of second circle = 5 m

Height of last circle = 3 m

Displacement limits: 10 mm

Figure 84. 120-bar dome truss configuration

The constraints are stresses on the members which are limited to 240 MPa and displacements are limited to 10 mm for every node in the structure in the negative z direction. The elasticity module is taken as 210000 MPa and the yield stress (σ_y) of the material is 240 MPa. Also, material density is taken as 7860 kg/m³, and loading conditions are taken as 60 kN on the top point, 30 kN on every node at the first circle, and 10 kN on every node at the last circle in the negative -z direction, same with Saka's example as can be seen from Figure 85.

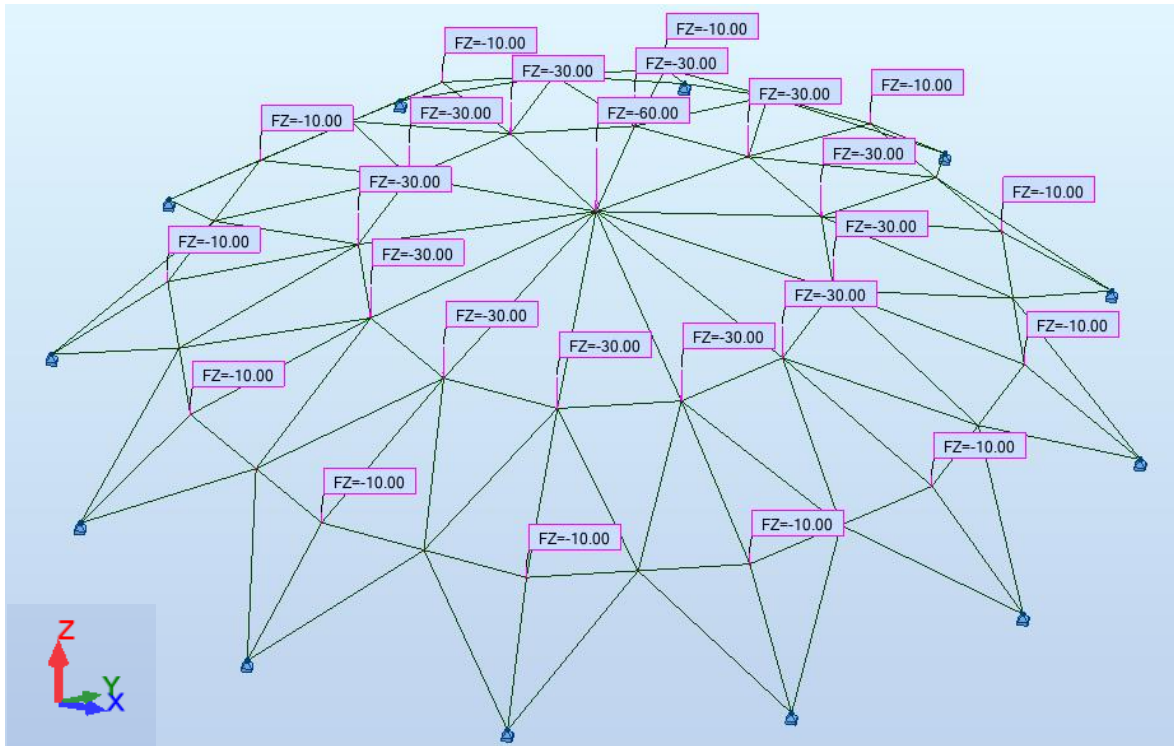


Figure 85. 120-bar dome truss loading conditions

Since the authors did not select cross-sectional areas based on specific regulations in the study used as an example, but rather only indicated the minimum limits, 27 CHS (Circular Hollow Section) sections that comply with the provided limits were chosen from the BS EN 10219-2:2006 standard. The list and corresponding properties are given in Table 13. The example aims to find suitable sections that give minimum weight while satisfying given constraints for 120 elements that are divided into 7 groups from the list that contains 27 different cross-sectional areas. The grouping system of elements is given in Table 14.

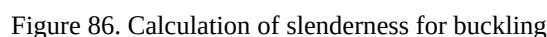
Table 13. Discrete cross-sectional area list and names

Number	Name of Section	Cross-section area (cm ²)	I _{x,y} (cm ⁴)	Radius of gyration (mm)
1	CHS 33.7x3.0	2,89	3,44	10,9
2	CHS 42.4x3.0	3,71	7,25	14
3	CHS 48.3x3.0	4,27	11	16,1
4	CHS 60.3x3.0	5,4	22,2	20,3
5	CHS 76.1x3.0	6,89	46,1	25,9
6	CHS 60.3x4.0	7,07	28,2	20
7	CHS 88.9x3.0	8,1	74,8	30,4
8	CHS 76.1x4.0	9,06	59,1	25,5
9	CHS 114.3x3.0	10,5	163	39,4
10	CHS 88.9x4.0	10,7	96,3	30
11	CHS 88.9x5.0	13,2	116	29,7
12	CHS 114.3x4.0	13,9	211	39
13	CHS 88.9x6.3	16,3	140	29,3
14	CHS 114.3x5.0	17,2	257	38,7
15	CHS 114.3x6.0	20,4	300	38,3
16	CHS 139.7x5.0	21,2	481	47,7
17	CHS 139.7x6.0	25,2	564	47,3
18	CHS 139.7x6.3	26,4	589	47,2
19	CHS 193.7x5.0	29,6	1320	66,7
20	CHS 168.3x6.0	30,6	1010	57,4
21	CHS 219.1x5.0	33,6	1930	75,7
22	CHS 193.7x6.3	37,1	1630	66,3
23	CHS 219.1x6.3	42,1	2390	75,3
24	CHS 244.5x6.3	47,1	3350	84,2
25	CHS 219.1x8.0	53,1	2960	74,7
26	CHS 219.1x10.0	65,7	3600	74
27	CHS 406.4x6.3	79,2	15800	141

Table 14. Grouping of 120-bar dome members

Group	Members
1	1-12
2	13-24
3	25-36
4	37-60
5	61-84
6	85-96
7	97-120

As mentioned before, the first thing is calculating the slenderness ratio to apply these formulations. To calculate the slenderness, length of elements and radius of gyration are needed. The length of elements is obtained by using the start and end points of lines from the Dynamo model and the radius of gyration is entered into the code block manually as can be seen from Figure 86.



The next step is using this information and calculating critical stress for buckling. For this operation, a Python code has been created (see Figure 87) to run inside the Dynamo. Python code takes lengths and radius of gyration values as an input and applies the formulations that are given above and gives a list of critical stresses for elements as depicted in Figure 88.

```

1# Load the Python Standard and DesignScript Libraries
2import sys
3import clr
4import math
5clr.AddReference('ProtoGeometry')
6from Autodesk.DesignScript.Geometry import *
7
8# The inputs to this node will be stored as a list in the IN variables.
9dataEnteringNode = IN
10
11# Given data
12radius_of_gyration = IN[0] # mm
13L = IN[1] # mm
14E = 210000 # n/mm2
15Yield_stress = 240 # n/mm2
16
17slenderness = []
18final_stresses = []
19Cc = math.sqrt(2 * math.pi ** 2 * E / Yield_stress)
20
21
22for i in range(len(L)):
23    slenderness_ratio = L[i] / radius_of_gyration[i] # Calculate slenderness ratio
24    slenderness.append(slenderness_ratio)
25
26for i, ratio in enumerate(slenderness):
27    if ratio > Cc:
28        Pcr = math.pi ** 2 * E / ratio ** 2
29        final_stresses.append(Pcr)
30
31    else:
32        Pc = Yield_stress * (1 - 0.5 * ratio ** 2 / Cc ** 2)
33        final_stresses.append(Pc)
34
35
36
37# Assign your output to the OUT variable.
38OUT = final_stresses

```

Figure 87. Python code for critical stress

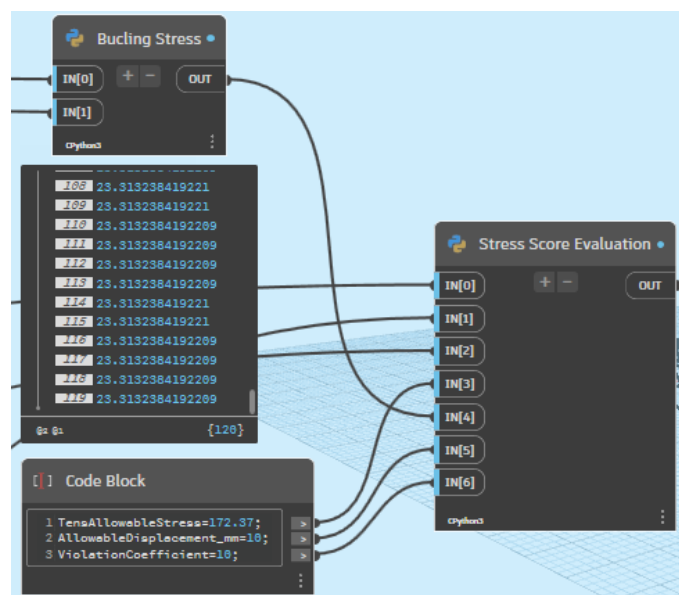


Figure 88. Critical stresses after calculations

Lastly, the weight score calculator that is detailly explained in the methodology section has to be updated to get new max stress values that are calculated with buckling formulations and separate the stresses that comes from RSA into two group as compression stress and tension stress based on if they are positive or negative as illustrated in Figure 90. The rest is the same as the previous code. The overall view of used Dynamo nodes and scripts to create this dome geometry is given in Figure 89. The copy of high resolution “.png” photo of the script has been added to Appendix C, Dynamo scripts section along with the text copy of all codes in Appendix A

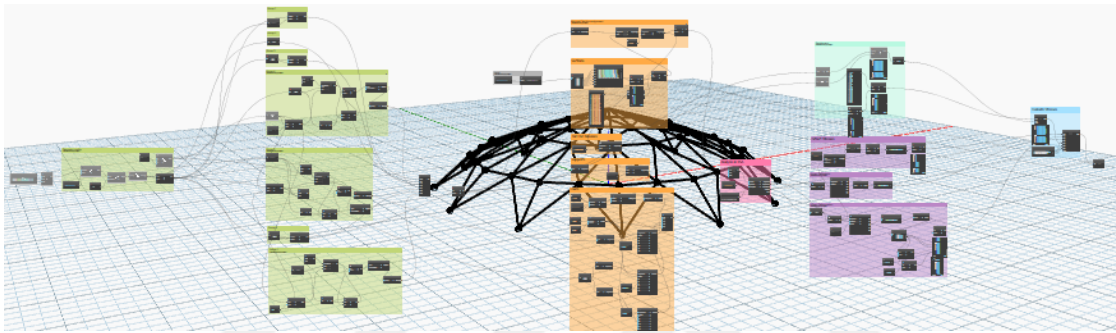


Figure 89. The script for creating a 120-bar dome truss

```

8 #Actual stress in the bar
9 stress=IN[0]
10
11 Displacement=IN[1]
12 #Set the maximal allowable stress
13 weight=IN[2]
14 #Sets the scoring system for the stress ratios
15 TensAllowableStress=IN[3]
16 bucling=IN[4]
17 AllowableDisplacement=IN[5]
18
19 violation_coef=IN[6]
20
21 tensstress=[]
22 compstress=[]
23 compindex=[]
24 for index, value in enumerate(stress):
25     if value < 0:
26         compstress.append((value))
27         compindex.append((index))
28     else:
29         tensstress.append((value))
30
31 compstress=[abs(value) for value in compstress]
32
33 CompAllowableStress=[bucling[index] for index in compindex]
34
35
36 stress_ratio=[]
37 violation = 0 # Initialize violation as a float or integer
38
39 for i in tensstress:
40     ratio = i / TensAllowableStress
41     stress_ratio.append(ratio)
42
43
44 for item1, item2 in zip(compstress, CompAllowableStress):
45     ratio = item1 / item2
46     stress_ratio.append(ratio)
47
48
49 for d in Displacement:
50     ratio = d / AllowableDisplacement
51     stress_ratio.append(ratio)
52
53 for r in stress_ratio:
54     if r - 1 > 0:
55         violation += r - 1
56
57 weight_score=weight*(1+(violation*violation_coef))
58
59 #Assign your output to the OUT variable.
60 OUT = weight_score

```

Figure 90. Updated weight score calculator for buckling

The weight score changes of the problem, for which a population size and generation size of 20 were used for the runs and calculations, are shown in Figure 91. Notably, in the figure, it is evident that there was no change in the weight score for seven generations after the best result was achieved. Consequently, the optimization process was concluded, and the best results were recorded in Table 15. The table includes the areas found for 120 elements, divided into 7 groups, along with their comparison to literature results. It is clear in this table that, area groups such as A1, A2, A4, and A7 were found to be smaller than those reported in the study from Saka et al. (Saka & Ulker, 1991). However, the remaining groups have higher values than the ones used for comparison, which led to the final result being slightly higher than the result Saka et al. found in their study. It is worth noting that conducting additional runs and increasing number of analyses for the same study is expected to yield better weight scores because the study conducted by Kooshkbaghi et al. (Kooshkbaghi & Kaveh, 2020) needed 8400 analysis to find the result depicted in Table 15 while only 400 analysis has been made in this thesis for 120-bar truss problem. It is also important to mention that in the study conducted by Kooshkbaghi et al. the displacement limitations are taken as ± 5 mm which is half of amount taken in the Saka's study which resulted higher weight for Kooshkbaghi's study. The reason for analyzing this problem in the examples section is to demonstrate that the proposed algorithm is effective and performs well as the number of elements increases. The results indicate that the proposed methodology can be applied to complex structures in future studies. Finally, Figure 92 presents the optimized structure and provides detailed information, including corresponding section properties, as found in this thesis.

Table 15. Comparison of 120-bar dome truss results

	Saka et al. 1991	Ebenau et al. 2005	Kooshkbaghi et al. 2020	This Thesis
Weight (kg)	7587	8512.2	9101.31	8343.13
A1 (cm²)	17.50	16.66	19.88	10.70
A2 (cm²)	45.56	44.84	21.54	42.10
A3 (cm²)	25.45	24.89	26.36	47.10
A4 (cm²)	8.44	9.66	17.79	2.89
A5 (cm²)	22.30	21.93	10.25	26.40
A6 (cm²)	15.96	16.59	21.18	30.58
A7 (cm²)	3.9	11.74	15.67	2.89

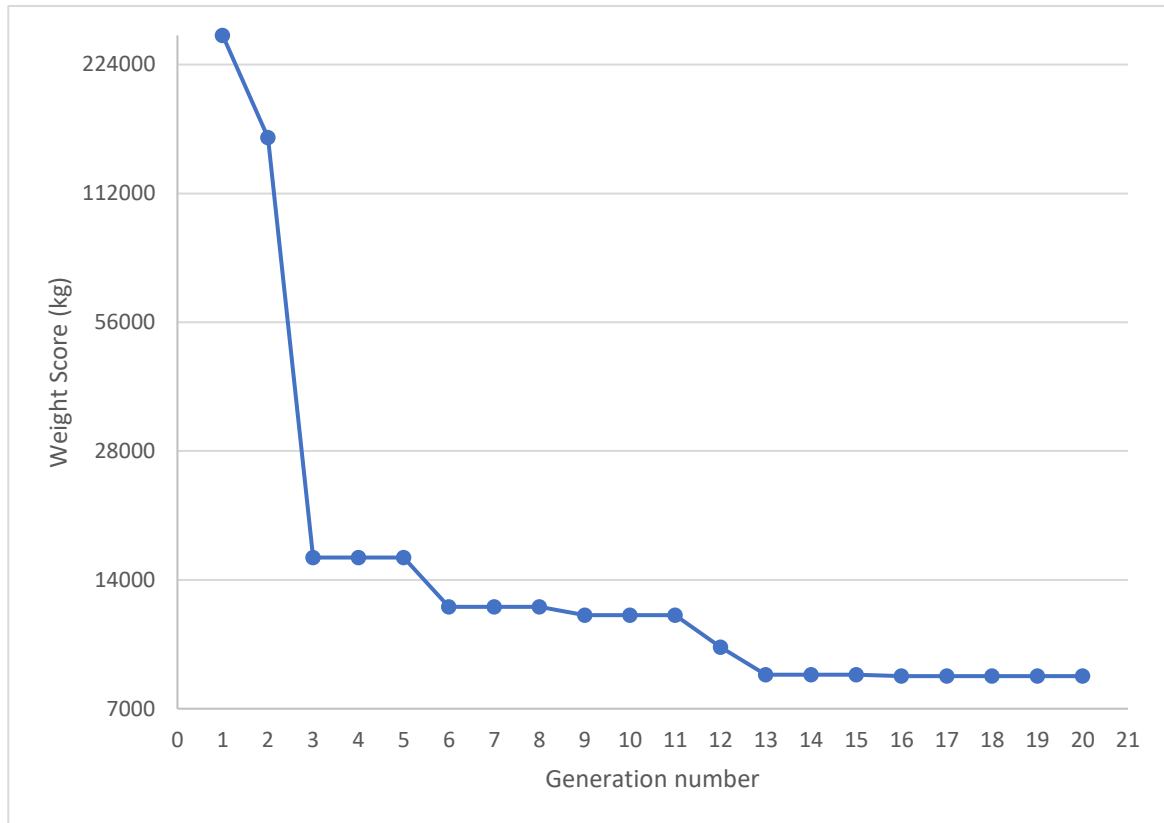
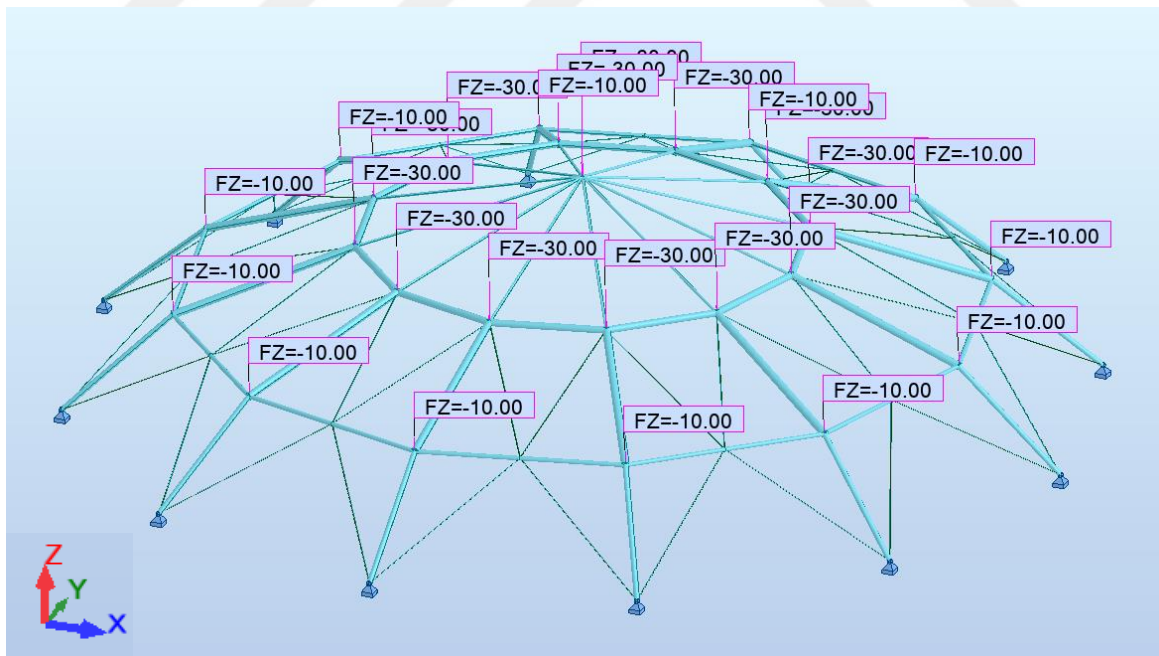


Figure 91. Weight score change for 120-bar truss by generation number



Total weight: 8343.134 (kg)
 Weight score: 8343.134 (kg)
 Maximum deflection: 9.25 (mm)
 Maximum stress: 36.68 (MPa)

Figure 92. The final version of the 120-bar dome truss

4.3.1. LCA and Cost Analysis for 120-Bar Truss

This section of the thesis provides LCA-cost analysis results of the optimized 120-bar dome truss obtained using the methodology explained in section 3.5. As the first step, the best model after optimization has been imported into Revit from Dynamo using the “Revit importer” custom node. Once the model is ready in Revit, as shown in Figure 93, then LCA analysis can be performed by using “Tally” as previously explained. After the analysis, a detailed report is generated, which includes information on factors such as global warming potential, acidification, and smog formation potential, as demonstrated in Figure 94 and Table 16. The complete report is available in Appendix B, LCA Report section.

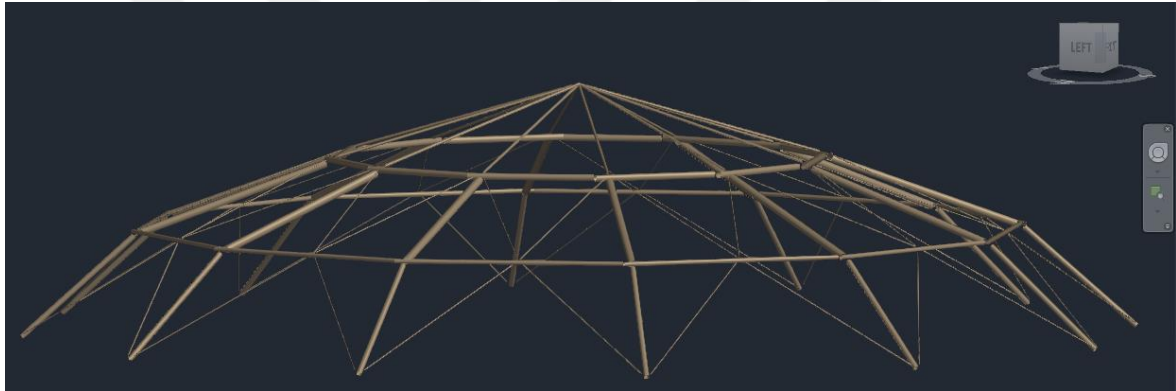
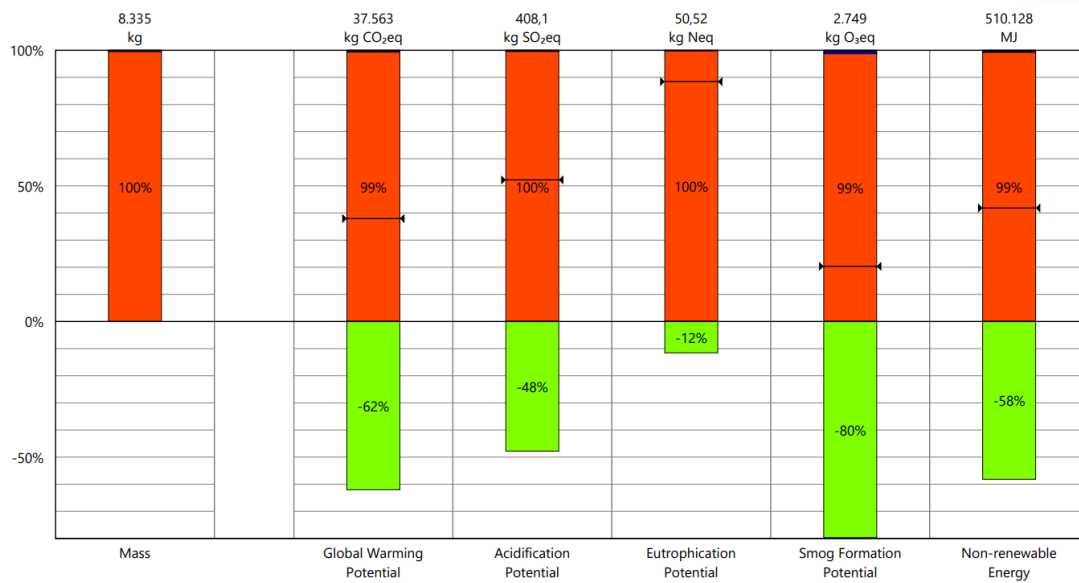


Figure 93. 120 bar truss on Revit

	Product Stage [A1-A3]	Construction Stage [A4]	Use Stage [B2-B5]	End of Life Stage [C2-C4]	Module D [D]
Environmental Impact Totals					
Global Warming (kg CO ₂ eq)	37.314	195,7	44,29	8,812	-23.306
Acidification (kg SO ₂ eq)	406,5	0,9069	0,7056	0,05809	-195
Eutrophication (kg Neq)	50,42	0,07384	0,01798	0,01103	-5,84
Smog Formation (kg O ₃ eq)	2.714	29,97	3,947	0,872	-2.190
Ozone Depletion (kg CFC-11eq)	0,001901	6,703E-012	2,566E-011	1,623E-012	-1,421E-004
Primary Energy (MJ)	568.589	2.846	871,2	151,3	-307.435
Non-renewable Energy (MJ)	506.473	2.778	735,4	141,3	-296.691
Renewable Energy (MJ)	61.702	68,82	134,1	9,972	-10.909
Environmental Impacts / Area					
Global Warming (kg CO ₂ eq/m ²)	48,15	0,2525	0,05715	0,01137	-30,1
Acidification (kg SO ₂ eq/m ²)	0,5245	0,00117	9,105E-004	7,495E-005	-0,2517
Eutrophication (kg Neq/m ²)	0,06506	9,528E-005	2,321E-005	1,423E-005	-0,007539
Smog Formation (kg O ₃ eq/m ²)	3,502	0,03867	0,005093	0,001125	-2,83
Ozone Depletion (kg CFC-11eq/m ²)	2,453E-006	8,649E-015	3,311E-014	2,095E-015	-1,834E-007
Primary Energy (MJ/m ²)	733,7	3,672	1,124	0,1952	-397
Non-renewable Energy (MJ/m ²)	653,5	3,585	0,9488	0,1824	-383
Renewable Energy (MJ/m ²)	79,61	0,0888	0,173	0,01287	-14,1

Figure 94. Summary of LCA

Table 16. Results per Life Cycle Stage



For cost analysis, a bill of quantity table has been created, as shown in Table 17. Fields in this table can be adjusted by specific needs. As an example, unit prices have been obtained from the referenced website (URL-24, 2023), and the total cost has been calculated automatically within the table.

Table 17. Revit bills of quantity table

<Structural Beams & Bracing Quantities>				
A	B	C	D	E
Structural Member	Type	Length	Cost	Final Cost (\$)
Circular Hollow Sections				
Circular Hollow Sections	CHS33.7x3.2	289.385	11.66	3374.226768
	CHS33.7x3.2: 48	289.385		3374.226768
Circular Hollow Sections	CHS88.9x4	86.681	18.91	1639.130146
	CHS88.9x4: 12	86.681		1639.130146
Circular Hollow Sections	CHS139.7x6.3	77.645	86.59	6723.263232
	CHS139.7x6.3: 24	77.645		6723.263232
Circular Hollow Sections	CHS168.3x6.3	54.322	107.47	5837.942352
	CHS168.3x6.3: 12	54.322		5837.942352
Circular Hollow Sections	CHS219.1x6.3	43.115	103.52	4463.254448
	CHS219.1x6.3: 12	43.115		4463.254448
Circular Hollow Sections	CHS244.5x6.3	70.895	170.45	12084.01866
	CHS244.5x6.3: 12	70.895		12084.01866
Circular Hollow Sections: 120		622.042		34121.835606
Grand total: 120		622.042		34121.835606

5. DISCUSSION

In this section, a transparent comment has been made about what aspects of this thesis could have been improved or were not fully addressed along with a brief comment about the results found at the end of this thesis.

Firstly, the methodology could have been enhanced by devoting more time to the proposed Dynamo script and Python codes. Due to the fixed time of the master thesis journey, some solutions to the problems that were encountered during the design process have been solved in inefficient ways in the programming aspect and enough time could not be spent on the script to find more efficient ways to achieve the same outcome. That caused longer optimization time which indirectly influenced the results of the study. Also, a connection between RSA and Dynamo has been established by employing a Dynamo package, which restricts the optimization process and the ability to use the full performance of RSA since any changes inside this package are not possible. If there is a problem during the optimization caused by this package, it is really hard to detect and solve that problem. That's why using API to perform FEM analysis on RSA is looking better option for future studies. Improving these aspects will help the designer to reach the best solutions with less analysis and in less time in future studies. Besides, the "Revit importer" section in the framework should be enhanced in terms of parametric design. Comparing an initial model with an optimized one and reporting changes in global warming effects, CO₂ emissions, or costs after optimization will better demonstrate the advantages and outcomes of LCA and Revit. However, to do this, there needs to be an initial design to optimize and the examples that are in the literature were not suitable for this action.

Furthermore, the algorithm used in the thesis and the definitions of the initial population size, mutation rate, and elite rate significantly affect the results of the examples. Additionally, the number of design variables and the size of the list containing these variables have a substantial influence on finding optimal results. While having a large list of design parameters allows for the evaluation of various options, it also comes at a significant time cost. The primary challenge with this method is the time required for certain tasks, particularly when importing the model into RSA, assigning boundary conditions, and conducting the analysis. These processes consume a significant amount of

time compared to methods where all operators and programs are coded within a single software using one programming language where the generation number does not impose a limitation on the study. However, in this thesis, as the generation number increases, the waiting time also rises due to issues like RAM leaks. Improving the interaction between RSA and Dynamo and reducing the model creation time in RSA could help address this problem. Due to these reasons, in this thesis, the run number was kept low and as soon as a solution close to the literature results was found, the optimization process was terminated. Additionally, the extended waiting times prevented the author from conducting more optimizations for the same example and parameters, selecting the best one, and conducting a thorough exploration of various parameters, including elite rate, population size, and mutation rate, to determine the optimal settings, all of which also impacted the time required to reach the best solutions. Despite the reasons explained so far, the solutions obtained by using the model proposed in this thesis are meaningful from the viewpoint of engineering manner and computational efforts. For the 10-bar truss, the relative error (%) compared to the literature study is 11.33%. However, for the 36-bar truss, although the relative error was found to be 9.43%, it required 1600 analyses to obtain this result, whereas only 400 analyses were conducted for other examples. Despite using a higher generation number for the 36-bar example, it took 10 separate runs to reach the result in the thesis. The reason for this is the coded simple GA struggled with a large number of design variables, specifically 21 in the case of the 36-bar truss example. The results can be improved by updating the GA or finding optimal parameter settings, as mentioned earlier. On the other hand, for the 120-bar truss, the relative error is 9.98%, and this result was achieved with 400 analyses in a single run. This demonstrates that the created algorithm performs well for cases with a low number of design variables, but it may require updates for analyses involving a large number of design variables in future studies. Overall, the results obtained are approximately 10% higher than those in the literature. This difference is considered acceptable given that the main focus of this thesis is to establish a workflow for structural optimization using visual programming, parametric modeling, and other BIM tools, rather than coding a GA that works best among the ones created in the literature.

6. CONCLUSION

In this thesis, a SOO methodology, which aims to integrate the structural optimization process into the BIM environment using VP, has been developed. This methodology was then validated by comparing the results of some 2D and 3D benchmark problems from the open literature. Also, to show the advantages of using BIM during the optimization process, LCA and Cost analysis have been added to the proposed methodology and results sections.

Even though structural optimization with meta-heuristic algorithms is a well-known subject, its application and utilization in real-world problems are not common. That is why this problem is approached by using technological advancements like Parametric design, VP, BIM, etc. by using software like Dynamo, Revit, and RSA during the optimization process in this thesis. Dynamo was the main tool in this thesis which runs the VP and textual programming scripts and Dynamo's ability to become a bridge between RSA and Revit made this thesis possible. During the thesis, another bridge has been established between scientific ideas and practical coding by using another edge technology which is GPT. The outcomes of using GPT demonstrated the potential of modern technologies, making it possible for scholars to incorporate programming into their projects by removing the need for significant knowledge of programming when using it in civil engineering workflows. Moreover, the results of this thesis showed the adaptability of VP and Dynamo to the optimization process and the advantages of using BIM tools.

In summary, this thesis has laid a solid foundation for further research and innovation in the field of structural optimization. The combination of BIM, metaheuristic algorithms, and advanced technologies opens up exciting possibilities for creating more sustainable, cost-effective, and efficient structures in the years to come.

7. SUGGESTION

This section provides recommendations and proposals based on the findings in this thesis.

Firstly, Graphics and tables have been created using Excel in this study even though CPython3 allows users to use Python libraries. That is how graphics and tables can be created directly inside the Dynamo to provide detailed information about optimizations and analysis processes. Additionally, examples of size, shape, and topology optimization can be taken from the literature and the same examples can be optimized with the proposed methodology. This will provide different aspects for the proposed methodology than optimizing only sections but also trying to find the best truss configuration, height, width, connections, or even support location.

Secondly, the proposed script allows the user to change the metaheuristic algorithm before optimization. Different algorithms like ACO, PSO, or GA can be tested in the same problem to compare results or find better results. Moreover, MOO has not performed with the proposed script even though it is possible to implement it in the script. In future works, the proposed methodology can be updated in this direction.

Lastly, extending the study for other type of structures such as grids, frames, retaining walls, etc., would be a good improvement for future work, as in this study only trusses have been analyzed. To achieve this making the connection between RSA and Dynamo by directly utilizing RSA API instead of using a package that doesn't allow the user to reach all RSA features could help to designer. If the connection is made by the designer, detailed analysis can be made and special regulation can be used with its constraints. This will help to improve the optimization process and obtain more accurate results.

8. REFERENCES

- AISC. (1961). *Specification for the Design, Fabrication & Erection of Structural Steel for Buildings*. American Institute of Steel Construction. New York.
- Arora, J. S. (1989). *Introduction to optimum design*. Elsevier.
- Artar, M., & Carbas, S. (2021). Discrete sizing design of steel truss bridges through teaching-learning-based and biogeography-based optimization algorithms involving dynamic constraints. *Structures*, 34, 3533–3547. <https://doi.org/10.1016/J.ISTRUC.2021.09.101>
- Azizi, M., Aickelin, U., Khorshidi, H. A., & Shishehgarkhaneh, M. B. (2022). Shape and size optimization of truss structures by Chaos game optimization considering frequency constraints. *Journal of Advanced Research*, 41, 89–100. <https://doi.org/10.1016/J.JARE.2022.01.002>
- Bakhtiary, N., Allinger, P., Friedrich, M., Mulfinger, F., Sauter, J., & Puchinger, M. (1996). A New Approach for Sizing, Shape and Topology Optimization. *Journal Of Materials & Manufacturing*, 105, 745–761. <https://www.jstor.org/stable/44734105>
- Beghini, L. L., Beghini, A., Katz, N., Baker, W. F., & Paulino, G. H. (2014). Connecting architecture and engineering through structural topology optimization. *Engineering Structures*, 59, 716–726. <https://doi.org/10.1016/j.engstruct.2013.10.032>
- Białkowski, S. (2016). Structural optimisation methods as a new toolset for architects. *Complexity & Simplicity: Proceedings of the 34th ECAADe International Conference. Oulu School of Architecture, Oulu, Finland*, 255–264.
- Blum, C., & Roli, A. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, 35(3), 268–308. <https://doi.org/10.1145/937503.937505>
- Brooks Jr, F. P. (1975). *The mythical man-month: essays on software engineering*. Pearson Educations.
- Cai, Z., & Wang, Y. (2006). A multiobjective optimization-based evolutionary algorithm for constrained optimization. *IEEE Transactions on Evolutionary Computation*, 10(6), 658–675. <https://doi.org/10.1109/TEVC.2006.872344>
- Camp, C. V., & Bichon, B. J. (2004). Design of Space Trusses Using Ant Colony Optimization. *Journal of Structural Engineering*, 130(5), 741–751. [https://doi.org/10.1061/\(ASCE\)0733-9445\(2004\)130:5\(741\)](https://doi.org/10.1061/(ASCE)0733-9445(2004)130:5(741))
- Cao, H., Qian, X., Chen, Z., & Zhu, H. (2017). Enhanced particle swarm optimization for size and shape optimization of truss structures. *Engineering Optimization*, 49(11), 1939–1956. <https://doi.org/10.1080/0305215X.2016.1273912>

- Capriles, P. V. S. Z., Fonseca, L. G., Barbosa, H. J. C., & Lemonge, A. C. C. (2006). Rank-based ant colony algorithms for truss weight minimization with discrete variables. *Communications in Numerical Methods in Engineering*, 23(6), 553–575. <https://doi.org/10.1002/cnm.912>
- Cardoso, J. M. P., Coutinho, J. G. F., & Diniz, P. C. (2017). Additional topics. In J. M. P. Cardoso, J. G. F. Coutinho, & P. C. Diniz (Eds.), *Embedded Computing for High Performance* (pp. 255–280). Elsevier. <https://doi.org/10.1016/B978-0-12-804189-5.00008-9>
- Chaszar, A., & Joyce, S. C. (2016). Generating freedom: Questions of flexibility in digital design and architectural computation. *International Journal of Architectural Computing*, 14(2), 167–181. <https://doi.org/10.1177/1478077116638945>
- Choi, S. W., Oh, B. K., & Park, H. S. (2017). Design technology based on resizing method for reduction of costs and carbon dioxide emissions of high-rise buildings. *Energy and Buildings*, 138, 612–620. <https://doi.org/10.1016/j.enbuild.2016.12.095>
- Christensen, P. W., & Klarbring, A. (2008). *An introduction to structural optimization* (Vol. 153). Springer Science & Business Media.
- Cicconi, P., Germani, M., Bondi, S., Zuliani, A., & Cagnacci, E. (2016). A Design Methodology to Support the Optimization of Steel Structures. *Procedia CIRP*, 50, 58–64. <https://doi.org/10.1016/J.PROCIR.2016.05.030>
- Coello Coello, C. A. (2002). Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191(11–12), 1245–1287. [https://doi.org/10.1016/S0045-7825\(01\)00323-1](https://doi.org/10.1016/S0045-7825(01)00323-1)
- Daloğlu, A., & Armutçu, M. (1998). Optimal Design of Plane Steel Frames with Genetic Algorithm (in Turkish). *Teknik Dergi*, 9(42).
- Darwin, C. (1964). *On the origin of species: A facsimile of the first edition*. Harvard University Press.
- Davis, D., & Peters, B. (2013). Design ecosystems: Customising the architectural design environment with software plug-ins. *Architectural Design*, 83(2), 124–131. <https://doi.org/10.1002/ad.1567>
- Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2–4), 311–338. [https://doi.org/10.1016/S0045-7825\(99\)00389-8](https://doi.org/10.1016/S0045-7825(99)00389-8)
- Deb, K., & Deb, K. (2014). Multi-objective Optimization. In *Search Methodologies* (pp. 403–449). Springer US. https://doi.org/10.1007/978-1-4614-6940-7_15
- Dede, T., Kripka, M., Togan, V., Yepes, V., & Rao, R. V. (2019). *Usage of Optimization Techniques in Civil Engineering During the Last Two Decades*.

- Dorigo, M., Birattari, M., & Stutzle, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4), 28–39. <https://doi.org/10.1109/MCI.2006.329691>
- Ebenau, C., Rottschäfer, J., & Thierauf, G. (2005). An advanced evolutionary strategy with an adaptive penalty function for mixed-discrete structural optimisation. *Advances in Engineering Software*, 36(1), 29–38. <https://doi.org/10.1016/j.advengsoft.2003.10.008>
- Finnveden, G., & Potting, J. (2014). Life Cycle Assessment. *Encyclopedia of Toxicology: Third Edition*, 74–77. <https://doi.org/10.1016/B978-0-12-386454-3.00627-8>
- Fischer, T., & Herr, C. M. (2001). *Teaching Generative Design*. 147–160.
- Fu, F. (2018). Design and Analysis of Complex Structures. In *Design and Analysis of Tall and Complex Structures* (pp. 177–211). Elsevier. <https://doi.org/10.1016/B978-0-08-101018-1.00006-X>
- Gero, J. S. (1992). Artificial Intelligence in Design'92. In J. S. Gero & F. Sudweeks (Eds.), *Artificial Intelligence in Design '92*. Springer Science & Business Media. <https://doi.org/10.1007/978-94-011-2787-5>
- Glover, F., & Sörensen, K. (2015). Metaheuristics. *Scholarpedia*, 10(4), 6532. <https://doi.org/10.4249/scholarpedia.6532>
- Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. In *Optimization, Machine Learning*. Addison-wesley.
- Goldberg, D. E., & Samtani, M. P. (1986). Engineering optimization via genetic algorithm. *Electronic Computation*, 471–482.
- Gordon, J. E. (1978). *Structures: or why things don't fall down (Appendix 4)*. Da Capo Press (Plenum Publishing Co.), New York.
- Groenwold, A. A., Stander, N., & Snyman, J. A. (1999). A regional genetic algorithm for the discrete optimal design of truss structures. *International Journal for Numerical Methods in Engineering*, 44(6), 749–766. [https://doi.org/10.1002/\(SICI\)1097-0207\(19990228\)44:6<749::AID-NME523>3.0.CO;2-F](https://doi.org/10.1002/(SICI)1097-0207(19990228)44:6<749::AID-NME523>3.0.CO;2-F)
- Han, Z., Cao, N., Liu, G., & Yan, W. (2019). *MOPSO for BIM: A Multi-Objective Optimization Tool Using Particle Swarm Optimization Algorithm on a BIM-based Visual Programming Platform*.
- Han, Z., Ning, C., & Wei, Y. (2019). MOPSO for BIM: a multi-objective optimization tool using particle swarm optimization algorithm on a BIMbased visual programming platform. *Hello, Culture*, 39–51.
- He, T., Wang, H., & Yoon, S. W. (2018). Comparison of Four Population-Based Meta-Heuristic Algorithms on Pick-and-Place Optimization. *Procedia Manufacturing*, 17, 944–951. <https://doi.org/10.1016/J.PROMFG.2018.10.112>

- Holland, J. H. (1992). Genetic Algorithms. *Scientific American*, 267(1), 66–73. <http://www.jstor.org/stable/24939139>
- Holzer, D. (2011). BIM's Seven Deadly Sins. *International Journal of Architectural Computing*, 9(4), 463–480. <https://doi.org/10.1260/1478-0771.9.4.463>
- Jiao, L., Li, L., Shang, R., Liu, F., & Stolkin, R. (2013). A novel selection evolutionary strategy for constrained optimization. *Information Sciences*, 239, 122–141. <https://doi.org/10.1016/j.ins.2013.03.002>
- Kaveh, A., Mirzaei, B., & Jafarvand, A. (2014). Shape-size optimization of single-layer barrel vaults using improved magnetic charged system search. *International Journal of Civil Engineering*, 12(4).
- Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *Proceedings of ICNN'95 - International Conference on Neural Networks*, 4, 1942–1948. <https://doi.org/10.1109/ICNN.1995.488968>
- Kirkpatrick, S. (1984). Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics*, 34(5–6), 975–986. <https://doi.org/10.1007/BF01009452/METRICS>
- Kooshkbaghi, M., & Kaveh, A. (2020). Sizing Optimization of Truss Structures with Continuous Variables by Artificial Coronary Circulation System Algorithm. *Iranian Journal of Science and Technology - Transactions of Civil Engineering*, 44(1), 1–20. <https://doi.org/10.1007/S40996-019-00254-2/FIGURES/17>
- Lin, J.-R., Zhang, Y., Kong SAR, H., & Xiao, J. (2019, December 8). A Framework to Automate Reliability-based Structural Optimization based on Visual Programming and OpenSees. <https://doi.org/10.13140/RG.2.2.23831.73126>
- Liu, J., Teo, K. L., Wang, X., & Wu, C. (2016). An exact penalty function-based differential search algorithm for constrained global optimization. *Soft Computing*, 20(4), 1305–1313. <https://doi.org/10.1007/s00500-015-1588-6>
- Machairas, V., Tsangrassoulis, A., & Axarli, K. (2014). Algorithms for optimization of building design: A review. *Renewable and Sustainable Energy Reviews*, 31, 101–112. <https://doi.org/10.1016/J.RSER.2013.11.036>
- Mani, A., & Patvardhan, C. (2009). A novel hybrid constraint handling technique for evolutionary optimization. *2009 IEEE Congress on Evolutionary Computation*, 12(1), 2577–2583. <https://doi.org/10.1109/CEC.2009.4983265>
- Mei, L., & Wang, Q. (2021). Structural optimization in civil engineering: A literature review. In *Buildings* (Vol. 11, Issue 2, pp. 1–28). MDPI AG. <https://doi.org/10.3390/buildings11020066>
- Mirniazmandan, S., Alaghmandan, M., Barazande, F., & Rahimianzarif, E. (2018). Mutual effect of geometric modifications and diagrid structure on structural optimization of tall buildings. <https://doi.org/10.1080/00038628.2018.1477043>, 61(6), 371–383. <https://doi.org/10.1080/00038628.2018.1477043>

- Noone, M., & Mooney, A. (2018). Visual and textual programming languages: a systematic review of the literature. *Journal of Computers in Education*, 5(2), 149–174. <https://doi.org/10.1007/s40692-018-0101-5>
- Peng, C., Liu, H. L., & Gu, F. (2018). A novel constraint-handling technique based on dynamic weights for constrained optimization problems. *Soft Computing*, 22(12), 3919–3935. <https://doi.org/10.1007/s00500-017-2603-x>
- Rajeev, S., & Krishnamoorthy, C. S. (1992). Discrete Optimization of Structures Using Genetic Algorithms. *Journal of Structural Engineering*, 118(5), 1233–1250. [https://doi.org/10.1061/\(ASCE\)0733-9445\(1992\)118:5\(1233\)](https://doi.org/10.1061/(ASCE)0733-9445(1992)118:5(1233))
- Rajwar, K., Deep, K., & Das, S. (2023). An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges. *Artificial Intelligence Review* 2023, 1–71. <https://doi.org/10.1007/S10462-023-10470-Y>
- Rasheed, K. M. (1998). *GADO: A genetic algorithm for continuous design optimization*. Rutgers The State University of New Jersey, School of Graduate Studies.
- Ringertz, U. T. (1988). ON METHODS FOR DISCRETE STRUCTURAL OPTIMIZATION. *Engineering Optimization*, 13(1), 47–64. <https://doi.org/10.1080/03052158808940946>
- Runarsson, T. P., & Xin Yao. (2000). Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3), 284–294. <https://doi.org/10.1109/4235.873238>
- Saka, M. P., & Ulker, M. (1991). Optimum design of geometrically nonlinear space trusses. *Computers & Structures*, 41(6), 1387–1396. [https://doi.org/10.1016/0045-7949\(91\)90276-R](https://doi.org/10.1016/0045-7949(91)90276-R)
- Salleh, M. N. M., Hussain, K., Cheng, S., Shi, Y., Muhammad, A., Ullah, G., & Naseem, R. (2018). *Exploration and Exploitation Measurement in Swarm-Based Metaheuristic Algorithms: An Empirical Analysis* (pp. 24–32). https://doi.org/10.1007/978-3-319-72550-5_3
- Savic, D. (2002, June 1). *Single-objective vs. Multiobjective Optimisation for Integrated Decision Support*.
- Schmit, L. A. (2012). Structural synthesis - Its genesis and development. <https://doi.org/10.2514/3.7859>, 19(10), 1249–1263. <https://doi.org/10.2514/3.7859>
- Schutte, J. F., & Groenwold, A. A. (2003). Sizing design of truss structures using particle swarms. *Structural and Multidisciplinary Optimization*, 25(4), 261–269. <https://doi.org/10.1007/S00158-003-0316-5/METRICS>
- Sörensen, K., & Glover, F. W. (2013). Metaheuristics. In *Encyclopedia of Operations Research and Management Science* (pp. 960–970). Springer US. https://doi.org/10.1007/978-1-4419-1153-7_1167

- Sotiropoulos, S., & Lagaros, N. D. (2020). Topology Optimization of Framed Structures using SAP2000. *Procedia Manufacturing*, 44, 68–75. <https://doi.org/https://doi.org/10.1016/j.promfg.2020.02.206>
- Sun, S. (2020). *Nonlinear Analysis of Topology-Optimized Scissor-Like Elements During Deployment and Structural Performance Analysis* [Master's Thesis]. Massachusetts Institute of Technology.
- Takahama, T., & Sakai, S. (2010). Efficient constrained optimization by the ε constrained adaptive differential evolution. *IEEE Congress on Evolutionary Computation*, 1–8. <https://doi.org/10.1109/CEC.2010.5586545>
- Tejani, G. G., Savsani, V. J., & Patel, V. K. (2016). Modified sub-population teaching-learning-based optimization for design of truss structures with natural frequency constraints. *Mechanics Based Design of Structures and Machines*, 44(4), 495–513. <https://doi.org/10.1080/15397734.2015.1124023>
- Tessema, B., & Yen, G. G. (2009). An adaptive penalty formulation for constrained evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans*, 39(3), 565–578. <https://doi.org/10.1109/TSMCA.2009.2013333>
- Toğan, V. (2004). *Size and Shape Optimization of Truss Systems Under Static and Live Loads Using Genetic Algorithm (in Turkish)* [Master's Thesis]. Karadeniz Technical University.
- Toğan, V., & Daloğlu, A. T. (2008). An improved genetic algorithm with initial population strategy and self-adaptive member grouping. *Computers & Structures*, 86(11–12), 1204–1218. <https://doi.org/10.1016/J.COMPSTRUC.2007.11.006>
- URL-1. (2023, July 25). *BIM Benefits/Why Use BIM/Autodesk*. <https://www.autodesk.com/industry/aec/bim/benefits-of-bim>
- URL-2. (2023, July 5). *What Is Computational Design? | Constructible*. <https://constructible.trimble.com/construction-industry/what-is-computational-design>
- URL-3. (2023, August 13). *What is Generative Design? | Siemens Software*. <https://www.plm.automation.siemens.com/global/en/our-story/glossary/generative-design/27063>
- URL-4. (2023, July 6). *What is Generative Design | Tools Software | Autodesk*. <https://www.autodesk.com/solutions/generative-design>
- URL-5. (2023, August 13). *Multi-objective optimization - Wikipedia*. https://en.wikipedia.org/wiki/Multi-objective_optimization
- URL-6. (2023, August 12). *Brute-force search - Wikipedia*. https://en.wikipedia.org/wiki/Brute-force_search
- URL-7. (2023, July 8). *Brute force approach - javatpoint*. <https://www.javatpoint.com/brute-force-approach>

- URL-8. (2023, August 12). *Shape optimization* - Wikipedia. https://en.wikipedia.org/wiki/Shape_optimization
- URL-9. (2023, August 13). *Qatar National Convention Centre [QNCC Qatar]*. <https://www.qncc.qa/>
- URL-10. (2023, July 12). *Metaheuristic* - Wikipedia. <https://en.wikipedia.org/wiki/Metaheuristic>
- URL-11. (2023, August 14). *A Gentle Introduction to Particle Swarm Optimization - MachineLearningMastery.com*. <https://machinelearningmastery.com/a-gentle-introduction-to-particle-swarm-optimization/>
- URL-12. (2023, August 14). *Life-cycle assessment* - Wikipedia. https://en.wikipedia.org/wiki/Life-cycle_assessment
- URL-13. (2023, July 22). Tally. <https://choosetally.com/>
- URL-14. (2023, July 10). *Autodesk Revit Software | Get Prices & Buy Official Revit 2024*. <https://www.autodesk.eu/products/revit/overview?term=1-YEAR&tab=subscription&plc=RVT>
- URL-15. (2023, July 10). *Robot Structural Analysis Professional | Download & Pricing*. <https://www.autodesk.com/products/robot-structural-analysis/overview?term=1-YEAR&tab=subscription>
- URL-16. (2023, July 10). *Explore - Dynamo BIM*. <http://dynamobim.org/explore/#intro>
- URL-17. (2023, July 10). *Custom Node Introduction | The Dynamo Primer*. https://primer.dynamobim.org/10_Custom-Nodes/10-1_Introduction.html
- URL-18. (2023, August 15). *Python (programming language)* - Wikipedia. [https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))
- URL-19. (2023, August 15). *The History of Python: A Brief Timeline of Python*. <https://python-history.blogspot.com/2009/01/brief-timeline-of-python.html>
- URL-20. (2023, July 10). *About Python™ | Python.org*. <https://www.python.org/about/>
- URL-21. (2023, August 15). *PYPL PopularitY of Programming Language index*. <https://pypl.github.io/PYPL.html>
- URL-22. (2023, August 15). *PEP 206 – Python Advanced Library* | [peps.python.org](https://peps.python.org/pep-0206/). <https://peps.python.org/pep-0206/>
- URL-23. (2023, August 18). *Robot Structural Analysis 2018 Help | Section Database | Autodesk*. <https://help.autodesk.com/view/RSAPRO/2018/ENU/?guid=GUID-727CAC1A-7ABE-4986-B7A5-4E31ADF1A6AA>
- URL-24. (2023, September 14). *ParkerSteel* - UK Steel Stockholders. <https://www.parkersteel.co.uk/>

- Vermeulen, D. (2018). *Structural Dynam(o)ite-Optimized Design and Fabrication Workflows w/ Dynamo Structural Dynam(o)ite Optimized Design and Fabrication Workflows with Dynamo*. www.linkedin.com/in/dietervermeulen
- Vermeulen, D., & Eng, S. (2015). *Dynam(o)ite Your Design for Engineers Learning Objectives*. www.linkedin.com/in/dietervermeulen
- Wang, Y., Cai, Z., Guo, G., & Zhou, Y. (2007). Multiobjective optimization and hybrid evolutionary algorithm to solve constrained optimization problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 37(3), 560–575. <https://doi.org/10.1109/TSMCB.2006.886164>
- Wang, Y., Cai, Z., Zhou, Y., & Zeng, W. (2008). An adaptive tradeoff model for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 12(1), 80–92. <https://doi.org/10.1109/TEVC.2007.902851>
- Wang, Y., Li, J. P., Xue, X., & Wang, B. C. (2020). Utilizing the Correlation between Constraints and Objective Function for Constrained Evolutionary Optimization. *IEEE Transactions on Evolutionary Computation*, 24(1), 29–43. <https://doi.org/10.1109/TEVC.2019.2904900>
- Williams, M. G., & Buehler, J. N. (1999). Comparison of visual and textual languages via task modeling. *International Journal of Human-Computer Studies*, 51(1), 89–115. <https://doi.org/10.1006/IJHC.1999.0270>
- Yeniay, Ö. (2005). Penalty Function Methods for Constrained Optimization with Genetic Algorithms. *Mathematical and Computational Applications*, 10(1), 45–56. <https://doi.org/10.3390/mca10010045>
- Zielinski, K., & Laur, R. (2006). Constrained Single-Objective Optimization Using Differential Evolution. *2006 IEEE International Conference on Evolutionary Computation*, 223–230. <https://doi.org/10.1109/CEC.2006.1688312>

9. APPENDICES

9.1. Appendix A, Python Scripts

9.1.1. Section 3.3.1, Figure 37. Python script used for evaluating and ranking the results

```
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
#The inputs to this node will be stored as a list in the IN variables.
dataEnteringNode = IN

#Actual stress in the bar
Tensstress=IN[0]
Compstress=IN[1]
Displacement=IN[2]
#Set the maximal allowable stress
weight=IN[3]
#Sets the scoring system for the stress ratios
TensAllowableStress=IN[4]
CompAllowableStress=IN[5]
AllowableDisplacement=IN[6]

violation_coef=IN[7]

stress_ratio=[]
violation = 0 # Initialize violation as a float or integer

for i in Tensstress:
    ratio = i / TensAllowableStress
    stress_ratio.append(ratio)

for i in Compstress:
    ratio = i / CompAllowableStress
    stress_ratio.append(ratio)

for d in Displacement:
    ratio = d / AllowableDisplacement
    stress_ratio.append(ratio)

for r in stress_ratio:
    if r - 1 > 0:
        violation += r - 1
```

```
weight_score=weight*(1+(violation*violation_coef))
#Assign your output to the OUT variable.
OUT = weight_score
```

9.1.2. Section 3.4.2, Figure 42. Python script to create an initial population

```
# Load the Python Standard and DesignScript Libraries
import sys
import clr
import random
import sys
from io import StringIO
sys.stdout = StringIO()
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

# The inputs to this node will be stored as a list in the IN variables.
dataEnteringNode = IN

#population=[[group] for group in population1]
lower_limits = IN[0]
upper_limits = IN[1]
population_size = IN[3]
chromosome_length = IN[2]

# Define the limits for each group
group_limits = [(int(lower_limits[i]), upper_limits[i]) if lower_limits[i] == 0 else (lower_limits[i],
upper_limits[i]) for i in range(len(lower_limits))]

# Generate the groups of individuals
groups = []
for group_limit in group_limits:
    group = []
    for i in range(population_size):
        if group_limit[0] == 0:
            chromosome = [random.uniform(group_limit[0], group_limit[1]) for j in
range(chromosome_length)]
        else:
            chromosome = [random.uniform(group_limit[0], group_limit[1]) for j in
range(chromosome_length)]
        group.append(chromosome)
    groups.append(group)

group.append([0])

OUT =groups
```

9.1.3. Section 3.4.3, Figure 50. Next population generator

```
# Load the Python Standard and DesignScript Libraries
import sys
import clr
import random
import sys
from io import StringIO
sys.stdout = StringIO()
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

# The inputs to this node will be stored as a list in the IN variables.
dataEnteringNode = IN

lower_limits = IN[1]
upper_limits = IN[2]

population_size = 1
chromosome_length = 5

original_list= IN[0]

inner_list = original_list[1][1]
fitness = inner_list[-1][1:] # Exclude the first element
variables= inner_list[:-1]
population=[[lst] for lst in variables]

#population = [[group] for group in population1]
#print("population",population)
elite_rate = 3
mutation_rate =0.1

population = [[[round(num, 1) for num in inner_lst] for inner_lst in outer_lst] for outer_lst in
population]
#print("population",population)

elite_count = int(population_size * elite_rate)

elite_indices1 = sorted(enumerate(fitness), key=lambda x: x[1])[:elite_count]
#print(elite_indices1)

elite_indices=[tup[0] for tup in elite_indices1]
#print(elite_indices)
```

```

elites = [[group[0][index] for index in elite_indices] for group in population]
#print("elites",elites)

```

```

# Define the number of individuals in each population
POPULATION_SIZE = 5

```

```

# # Define the range of possible values for each element in the list
# MIN_VALUE = [0, 0, 0, 0, 1]
# MAX_VALUE = [4, 4, 4, 4, 20]

```

```

def decimal_to_binary(number):
    binary = ""
    binary1 = ""
    if number < 0:
        binary += '-'
        number = abs(number)

    integer_part = int(number)

    if not float(number).is_integer():
        decimal_part1 = str(number)
        decimal_part = int(decimal_part1.split('.')[1])
    else:
        decimal_part=0

    ##print("integer_part",integer_part,"decimal_part",decimal_part)
    # Convert integer part to binary
    if integer_part == 0:
        binary += '0'
    else:
        while integer_part > 0:
            binary = str(integer_part % 2) + binary
            integer_part = integer_part // 2

    # Add decimal point if necessary
    if decimal_part == 0:
        binary1 += '0'
    else:
        while decimal_part > 0:
            binary1 = str(decimal_part % 2) + binary1
            decimal_part = decimal_part // 2

    binary=binary+'.'+binary1
    return binary

```

```

a=decimal_to_binary(2.0)
##print("aaaaaaaaaaaa",a)
def binary_to_decimal(binary):

```

```

# Check if the number is negative
negative = False
if binary[0] == '-':
    negative = True
    binary = binary[1:]

# Split the binary string into integer and decimal parts
int_part, frac_part = binary.split('.')
##print("int_part",int_part,"frac_part",frac_part)

# Convert the integer part to decimal
integer_part = 0

for i, digit in enumerate(int_part[::-1]):
    if digit == '1':
        integer_part += 2**i

# Convert the decimal part to decimal
decimal_part = 0

for i, digit in enumerate(frac_part[::-1]):
    if digit == '1':
        decimal_part += 2**i

##print("integer_partturn",integer_part,"decimal_partturn",decimal_part)
# Combine the integer and decimal parts
number = float(str(integer_part)+ '.'+ str(decimal_part))
##print(number)
# Apply the negative sign if necessary
if negative:
    number *= -1

return round(number,1)

def find_differing_positions(num1, num2):
    differing_positions = []
    # Convert the binary numbers to strings
    num1_str = str(num1)
    num2_str = str(num2)

    # Find the length of the longer number
    max_length = max(len(num1_str), len(num2_str))

    # Pad the numbers with leading zeros to make them equal in length
    num1_str = num1_str.zfill(max_length)
    num2_str = num2_str.zfill(max_length)

    # Compare the numbers bit by bit
    for i in range(max_length):
        if num1_str[i] != num2_str[i]:

```

```

    differing_positions.append(i)

return differing_positions

def binary_crossover(individual1, individual2):
    # Convert to lists if arguments are not already lists
    if type(individual1) is not list:
        individual1 = [individual1]
    if type(individual2) is not list:
        individual2 = [individual2]

    offspring1 = []
    offspring2 = []
    ##print("individual1",individual1)
    ##print("individual2",individual2)
    for i in range(min(len(individual1), len(individual2))):
        gene1 = decimal_to_binary(individual1[i])
        gene2 = decimal_to_binary(individual2[i])
        ##print("gene1",gene1)
        ##print("gene2",gene2)

        # Split the binary string into integer and fractional parts
        int_part1, frac_part1 = gene1.split('.')
        int_part2, frac_part2 = gene2.split('.')

        # Determine the maximum length of the integer and fractional parts
        max_int_len = max(len(int_part1), len(int_part2))
        max_frac_len = max(len(frac_part1), len(frac_part2))

        # Pad the integer and fractional parts with zeros to make them equal length
        int_part1 = int_part1.zfill(max_int_len)
        int_part2 = int_part2.zfill(max_int_len)
        frac_part1 = frac_part1.ljust(max_frac_len, '0')
        frac_part2 = frac_part2.ljust(max_frac_len, '0')
        """##print("int_part1",int_part1,
        "int_part2",int_part2,
        "frac_part1", frac_part1,
        "frac_part2", frac_part2)"""

    diff_index_int= find_differing_positions(int_part1, int_part2)
    diff_index1_frac= find_differing_positions(frac_part1, frac_part2)
    ##print("diff_index_int",diff_index_int,"diff_index1_frac",diff_index1_frac)
    if not diff_index1_frac:
        diff_index1_frac = [0]

    if not diff_index_int:
        diff_index_int = [0]

```

```

# Perform crossover on the integer part
crossover_index = random.choice(diff_index_int)
##print("crossover_indexint",crossover_index)
int_part1_new = list(int_part1)
int_part2_new = list(int_part2)
int_part1_new[crossover_index] = int_part2[crossover_index]
int_part2_new[crossover_index] = int_part1[crossover_index]
int_part1_new = ".join(int_part1_new)
int_part2_new = ".join(int_part2_new)

# Perform crossover on the fractional part

crossover_index = random.choice(diff_index1_frac)
##print("crossover_indexfrac",crossover_index)
frac_part1_new = list(frac_part1)
frac_part2_new = list(frac_part2)
frac_part1_new[crossover_index] = frac_part2[crossover_index]
frac_part2_new[crossover_index] = frac_part1[crossover_index]
frac_part1_new = ".join(frac_part1_new)
frac_part2_new = ".join(frac_part2_new)

# Combine the integer and fractional parts of the offspring
gene1_new = int_part1_new + '.' + frac_part1_new
gene2_new = int_part2_new + '.' + frac_part2_new
##print("gene1_new",gene1_new)
##print("gene2_new",gene2_new)

# Convert the binary string back to float gene
gene1_new = binary_to_decimal(gene1_new)
gene2_new = binary_to_decimal(gene2_new)

offspring1.append(gene1_new)
offspring2.append(gene2_new)

return offspring1, offspring2

offspring = []
for i, elite in enumerate(elites):
    #print(f"New populations for elite {i+1}: {elite}")
    population = []
    for k in range(POPULATION_SIZE):
        # Choose different parents from the elite
        parents = random.sample(elite, 2)
        parent1, parent2 = parents[0], parents[1]

        offspring1 = binary_crossover(parent1,parent2)
        #print(offspring1)
        #offspring1 = mutation(offspring1)
        # offspring2 = mutation(offspring2)
        population.append(offspring1)

```

```

population = [element for sublist in population for element in sublist]
#print(f"Population {i+1}: {population}")

unique_population = []
for sublist in population:
    if sublist not in unique_population:
        unique_population.append(sublist)
#print("unique_population",unique_population)

unique_population=[item for item in unique_population if lower_limits[i] <= item[0] <
upper_limits[i]]

# Count the number of elements in the elites list
count = len(elites[0])
#print("count",count,elites[0])
# Find the average of the elements in the elites list
average = sum(elites[0]) / count
#print("average",average,sum(elites[0]))

# If there are still elements in the elites list
if len(unique_population) > chromosome_length:
    while len(unique_population) > chromosome_length:
        # Find the element in the elites list that is farthest from the average
        farthest_element = max(unique_population, key=lambda x: abs(x[0] - average))
        #print("uzun",len(unique_population))

        # Remove the farthest element from the elites list
        unique_population.remove(farthest_element)
elif len(unique_population) < chromosome_length:
    while len(unique_population) < chromosome_length:
        unique_population.append([random.uniform(lower_limits[i], upper_limits[i])])
# Return the filtered and processed elites list
elif [0] in unique_population:
    index = unique_population.index([0])
    unique_population[index] = [random.uniform(lower_limits[i], upper_limits[i])]

population4=unique_population

#print(f"mutation {1}: {population4}")

offspring.append(unique_population)
#print("offspring",offspring)

flattened_list = [[inner[0] for inner in outer] for outer in offspring]

flattened_list.append([0])
OUT =flattened_list

```

9.1.4. Section 3.4.4, Figure 53. Offspring list and the selection operator

```
# Load the Python Standard and DesignScript Libraries
import sys
import clr
import random
import sys
from io import StringIO
sys.stdout = StringIO()
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

# The inputs to this node will be stored as a list in the IN variables.
dataEnteringNode = IN

population_size = 1
chromosome_length = 5

original_list= IN[0]
original_list2= IN[1]

inner_list1 = original_list2[1][1]
last_list1 = inner_list1[-1][1:] # Exclude the first element
variables1 = inner_list1[:-1]
variables1 = [[lst] for lst in variables1]

## print(last_list1)
## print(variables1)

inner_list = original_list[1]
fitness1 = inner_list[-1][1:] # Exclude the first element
variables= inner_list[:-1]
population1=[[lst] for lst in variables]

population = []
for sublist1, sublist2 in zip(population1, variables1):
    combined_sublist = sublist1[0] + sublist2[0]
    population.append([combined_sublist])

fitness= fitness1+last_list1
# print("fitness",fitness,"population",population)

elite_rate = 3
mutation_rate =0.1
population_size = 1
chromosome_length = 5
```

```

population = [[round(num, 1) for num in inner_lst] for inner_lst in outer_lst] for outer_lst in
population]
# # print("population",population)

elite_count = 5

elite_indices1 = sorted(enumerate(fitness), key=lambda x: x[1])[:elite_count]
# print(elite_indices1)

elite_indices=[tup[0] for tup in elite_indices1]
# print(elite_indices)

elites = [[group[0][index] for index in elite_indices] for group in population]
# print("elites",elites)

elites_fitness = [fitness[index] for index in elite_indices]
elites_fitness=[0]+elites_fitness
# print("elites_fitness",elites_fitness)

output=[0]+[elites+[elites_fitness]]

inner_list2 = original_list2[2][1]
last_list2 = inner_list2[-1][1:] # Exclude the first element
variables2 = inner_list2[:-1]
variables2 = [[lst] for lst in variables2]

hsfitness=[0]+last_list2+fitness1

hspop=[]

for sublist1, sublist2 in zip(variables2, population1):
    combined_sublist = sublist1[0] + sublist2[0]
    hspop.append([combined_sublist])

hspop = [item for sublist in hspop for item in sublist]
output2=["history"]+[hspop+[hsfitness]]
result=output+output2
# print("output",output)
OUT =output,output2

```

9.1.5. Section 3.5.1, Figure 57. Python script that chooses the best individual and gives its parameter

```
# Load the Python Standard and DesignScript Libraries
import sys
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

# The inputs to this node will be stored as a list in the IN variables.
dataEnteringNode = IN
original_list=IN[0]
# Place your code below this line
desired_list = original_list[1][1]
fitness = original_list[1][1][5]
fitness.remove(0)
population = desired_list[:-1]
#print(population)

elite_indices1 = sorted(enumerate(fitness), key=lambda x: x[1])[:1]
#print(elite_indices1)

elite_indices = [tup[0] for tup in elite_indices1]
#print(elite_indices)

elites = [group[index] for index in elite_indices for group in population]
#print("elites", elites)
# Assign your output to the OUT variable.
OUT = elites
```

9.1.6. Section 4.3, Figure 90. Updated weight score calculator for buckling

```
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

#The inputs to this node will be stored as a list in the IN variables.
dataEnteringNode = IN

#Actual stress in the bar
stress=IN[0]

Displacement=[1]
#Set the maximal allowable stress
weight=IN[2]
#Sets the scoring system for the stress ratios
TensAllowableStress=IN[3]
```

```

bucling=IN[4]
AllowableDisplacement=IN[5]

violation_coef=IN[6]

tensstress=[]
compstress=[]
compindex=[]
for index, value in enumerate(stress):
    if value < 0:
        compstress.append((value))
        compindex.append((index))
    else:
        tensstress.append((value))

compstress=[abs(value) for value in compstress]

CompAllowableStress=[bucling[index] for index in compindex]

stress_ratio=[]
violation = 0 # Initialize violation as a float or integer

for i in tensstress:
    ratio = i / TensAllowableStress
    stress_ratio.append(ratio)

for item1, item2 in zip(compstress, CompAllowableStress):
    ratio = item1 / item2
    stress_ratio.append(ratio)

for d in Displacement:
    ratio = d / AllowableDisplacement
    stress_ratio.append(ratio)

for r in stress_ratio:
    if r - 1 > 0:
        violation += r - 1

weight_score=weight*(1+(violation*violation_coef))

#Assign your output to the OUT variable.
OUT = weight_score

```

9.1.7. Section 4.3, Figure 88. Critical stresses after calculations

```
# Load the Python Standard and DesignScript Libraries
import sys
import clr
import math
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

# The inputs to this node will be stored as a list in the IN variables.
dataEnteringNode = IN

# Given data
radius_of_gyration = IN[0] # mm
L = IN[1] # mm
E = 210000 # n/mm2
Yield_stress = 240 # n/mm2

slenderness = []
final_stresses = []
Cc = math.sqrt(2 * math.pi ** 2 * E / Yield_stress)

for i in range(len(L)):
    slenderness_ratio = L[i] / radius_of_gyration[i] # Calculate slenderness ratio
    slenderness.append(slenderness_ratio)

for i, ratio in enumerate(slenderness):
    if ratio > Cc:
        Pcr = math.pi ** 2 * E / ratio ** 2
        final_stresses.append(Pcr)

    else:
        Pc = Yield_stress * (1 - 0.5 * ratio ** 2 / Cc ** 2)
        final_stresses.append(Pc)

# Assign your output to the OUT variable.
OUT = final_stresses
```

9.2. Appendix B, LCA Report

This section provides the full report of the LCA analysis conducted for the 120-bar dome truss in Examples and Result section.

Project Name

14.09.2023

Full building summary

Table of Contents

Report Summary	1
LCA Results	
Results per Life Cycle Stage	2
Results per Life Cycle Stage, itemized by Division	3
Results per Division	4
Results per Division, itemized by Tally Entry	5
Results per Division, itemized by Material	6
Results per Revit Category	7
Results per Revit Category, itemized by Family	8
Results per Building Element	9
Appendix	
Calculation Methodology - Life Cycle Assessment Methods	10
Calculation Methodology - Life Cycle Stages	11
Calculation Methodology - Environmental Impact Categories	12
LCI Data	13

Report Summary

Created with Tally

Non-commercial Version 2022.04.08.01

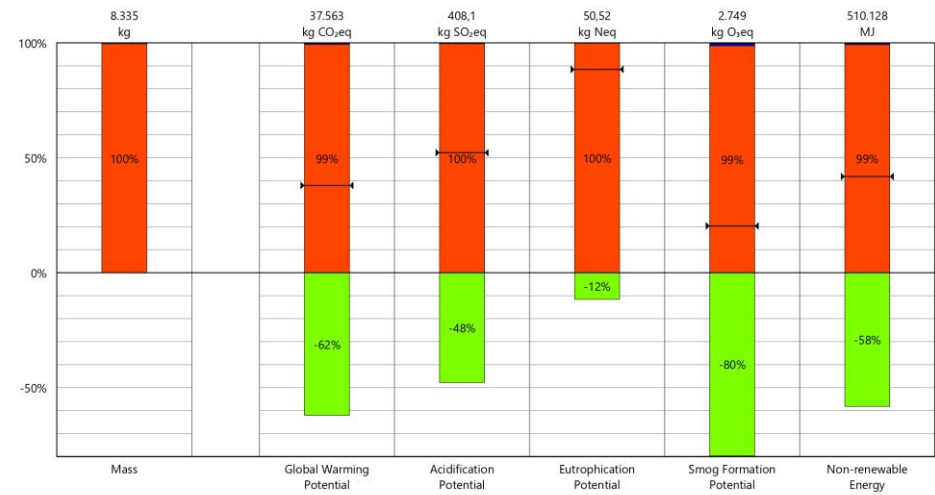
Goal and Scope of Assessment

Master sample project LCA test

Author	Feyzullah Yavan
Company	KIT
Date	22.07.2023
Project	Master Thesis
Location	Germany
Gross Area	50 m ²
Building Life	60 years
Boundaries	Cradle to grave, inclusive of biogenic carbon; see appendix for a full list of materials and processes

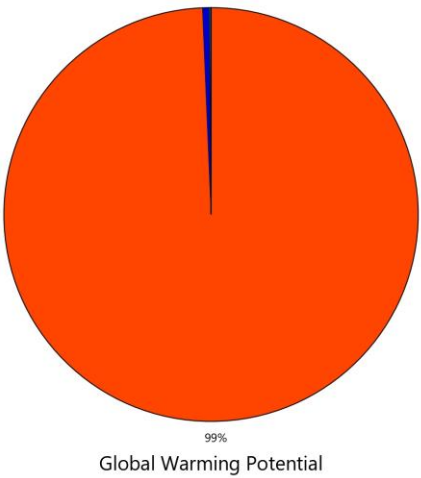
Environmental Impact Totals	Product Stage [A1-A3]	Construction Stage [A4]	Use Stage [B2-B5]	End of Life Stage [C2-C4]	Module D [D]
Global Warming (kg CO ₂ eq)	37.314	195,7	44,29	8,812	-23.306
Acidification (kg SO ₂ eq)	406,5	0,9069	0,7056	0,05809	-195
Eutrophication (kg Neq)	50,42	0,07384	0,01798	0,01103	-5,84
Smog Formation (kg O ₃ eq)	2.714	29,97	3,947	0,872	-2.190
Ozone Depletion (kg CFC-11eq)	0,001901	6,703E-012	2,566E-011	1,623E-012	-1,421E-004
Primary Energy (MJ)	568.589	2.846	871,2	151,3	-307.435
Non-renewable Energy (MJ)	506.473	2.778	735,4	141,3	-296.691
Renewable Energy (MJ)	61.702	68,82	134,1	9,972	-10.909
Environmental Impacts / Area					
Global Warming (kg CO ₂ eq/m ²)	48,15	0,2525	0,05715	0,01137	-30,1
Acidification (kg SO ₂ eq/m ²)	0,5245	0,00117	9,105E-004	7,495E-005	-0,2517
Eutrophication (kg Neq/m ²)	0,06506	9,528E-005	2,321E-005	1,423E-005	-0,007539
Smog Formation (kg O ₃ eq/m ²)	3,502	0,03867	0,005093	0,001125	-2,83
Ozone Depletion (kg CFC-11eq/m ²)	2,453E-006	8,649E-015	3,311E-014	2,095E-015	-1,834E-007
Primary Energy (MJ/m ²)	733,7	3,672	1,124	0,1952	-397
Non-renewable Energy (MJ/m ²)	653,5	3,585	0,9488	0,1824	-383
Renewable Energy (MJ/m ²)	79,61	0,0888	0,173	0,01287	-14,1

Results per Life Cycle Stage



Legend

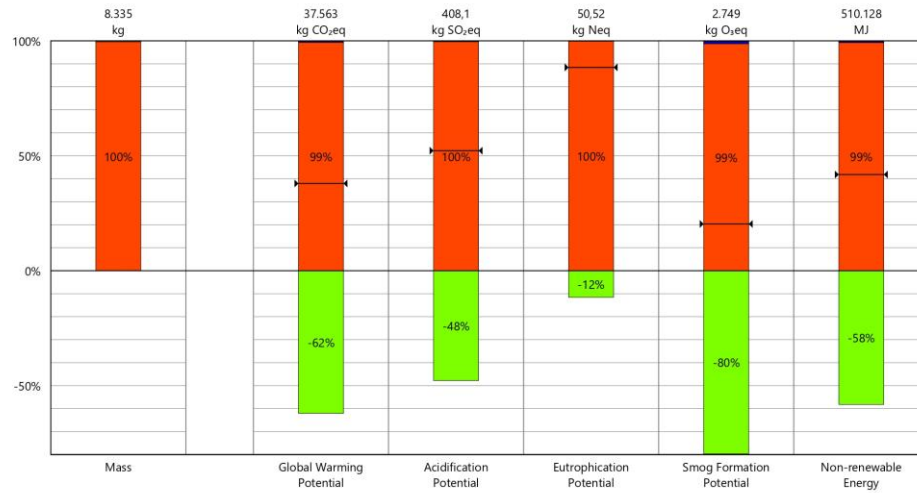
- Net value (impacts + credits)
- Life Cycle Stages
- Product [A1-A3]
 - Transportation [A4]
 - Maintenance and Replacement [B2-B5]
 - End of Life [C2-C4]
 - Module D [D]



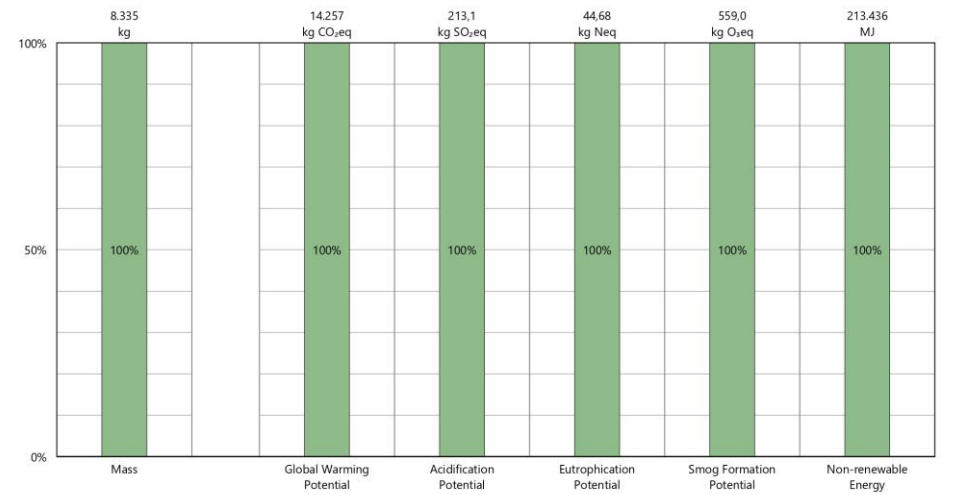
Project Name
Full building summary

14.09.2023

Results per Life Cycle Stage, itemized by Division

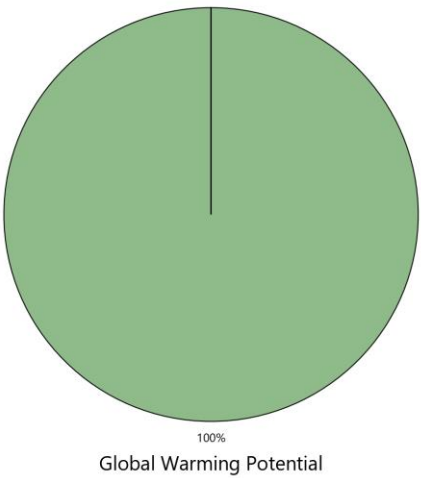


Results per Division



Legend

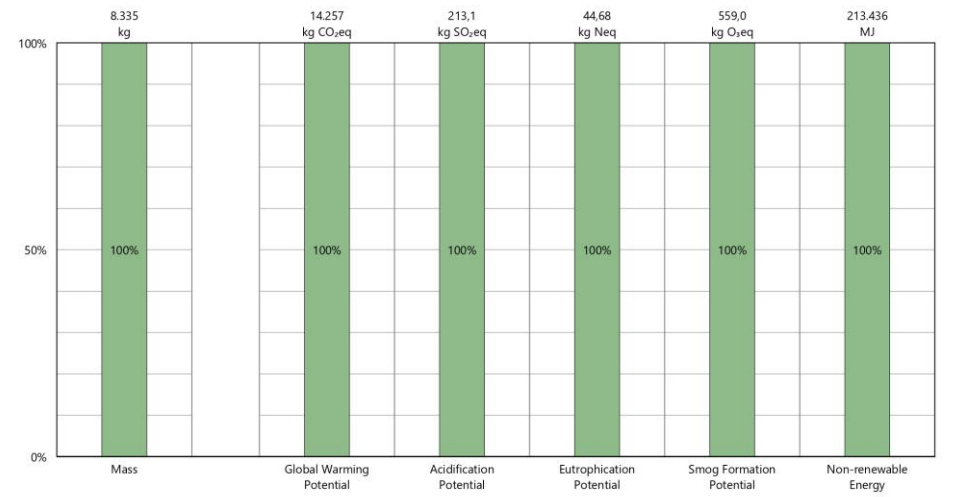
Divisions
05 - Metals



Project Name
Full building summary

14.09.2023

Results per Division, itemized by Tally Entry



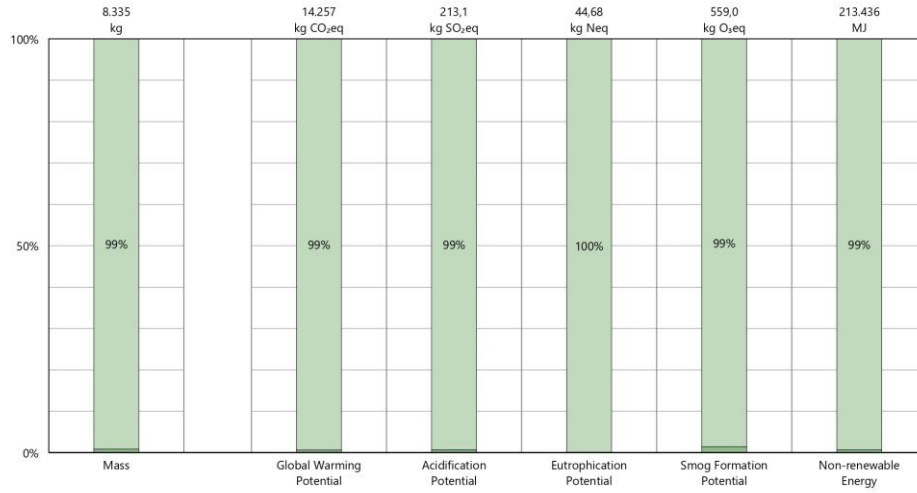
Legend

05 - Metals
Steel, HSS section

Project Name
Full building summary

14.09.2023

Results per Division, itemized by Material

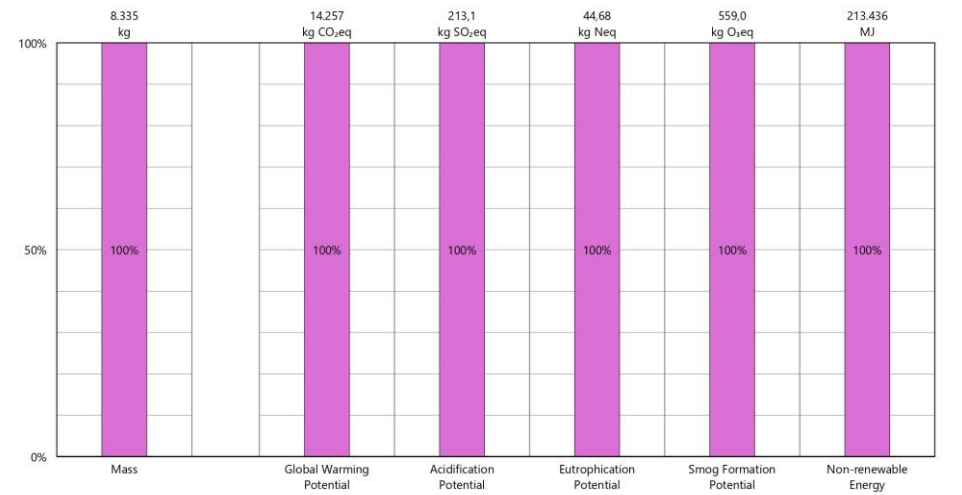


Legend

05 - Metals

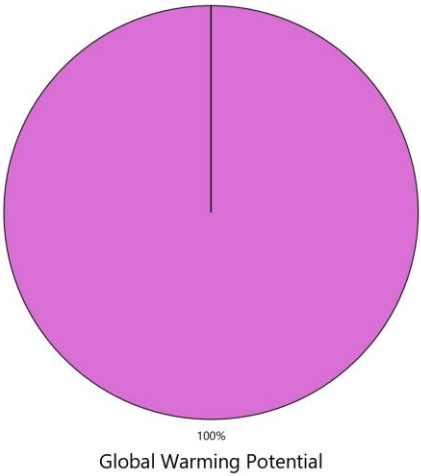
- Paint, exterior metal coating, silicone-based, by area
- Stainless steel, extruded, chromium 18/8

Results per Revit Category



Legend

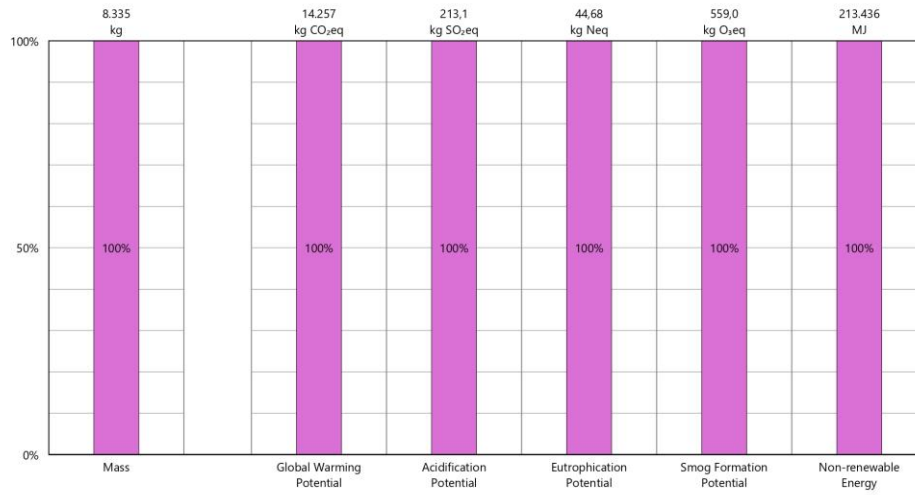
Revit Categories
Structure



Project Name
Full building summary

14.09.2023

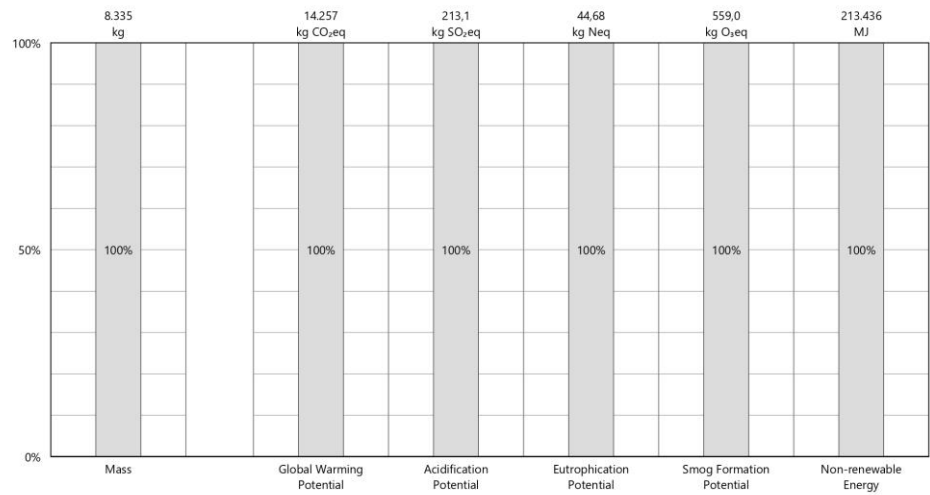
Results per Revit Category, itemized by Family



Legend

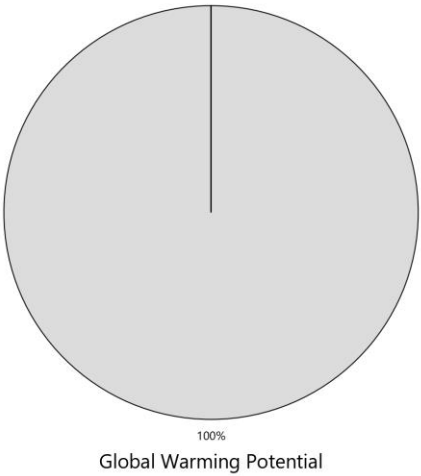
Structure
 Circular Hollow Sections

Results per Building Element



Legend

Building Elements
Undefined



Project Name

14.09.2023

Full building summary

Calculation Methodology

LIFE CYCLE ASSESSMENT METHODS

The following provides a description of terms and methods associated with the use of Tally to conduct life cycle assessment for construction works and construction products. Tally methodology is consistent with LCA standards ISO 14040-14044, ISO 21930:2017, ISO 21931:2010, EN 15804:2012, and EN 15978:2011. For more information about LCA, please refer to these standards or visit www.choosetally.com.

Studied objects

The life cycle assessment (LCA) results reported represent an analysis of a single building, multiple buildings, or a comparative analysis of two or more building design options. The assessment may represent the complete architectural, structural, and finish systems of the building(s) or a subset of those systems. This may be used to compare the relative environmental impacts associated with building components or for comparative study with one or more reference buildings. Design options may represent a full or partial building across various stages of the design process, or they may represent multiple schemes of a full or partial building that are being compared to one another across a range of evaluation criteria.

Functional unit and reference unit

A functional unit is the quantified performance of a product, building, or system that defines the object of the study. The functional unit of a single building should include the building type (e.g. office, factory), relevant technical and functional requirements (e.g. regulatory requirements, energy performance), pattern of use (e.g. occupancy, usable floor area), and the required service life. For a design option comparison of a partial building, the functional unit is the complete set of building systems or products that perform a given function. It is the responsibility of the modeler to assure that reference buildings or design options are functionally equivalent in terms of scope and relevant performance. The expected life of the building has a default value of 60 years and can be modified by the modeler.

The reference unit is the full collection of processes and materials required to produce a building or portion thereof and is quantified according to the given goal and scope of the assessment over the full life of the building. If construction impacts are included in the assessment, the reference unit also includes the energy, water, and fuel consumed on the building site during construction. If operational energy is included in the assessment, the reference unit includes the electrical and thermal energy consumed on site over the life of the building.

Data source

Tally utilizes a custom designed LCA database that combines material attributes, assembly details, and architectural specifications with environmental impact data resulting from the collaboration between KieranTimberlake and thinkstep. LCA modeling was conducted in GaBi 8.5 using GaBi 2018 databases and in accordance with [GaBi databases and modeling principles](#).

The data used are intended to represent the US and the year 2017. Where representative data were unavailable, proxy data were used. The datasets used, their geographic region, and year of reference are listed for each entry. An effort was made to choose proxy datasets that are technologically consistent with the relevant entry.

Data quality and uncertainty

Uncertainty in results can stem from both the data used and their application. Data quality is judged by: its measured, calculated, or estimated precision; its completeness, such as unreported emissions; its consistency, or degree of uniformity of the methodology applied on a study serving as a data source; and geographical, temporal, and technological representativeness. The [GaBi LCI databases](#) have been used in LCA models worldwide in both industrial and scientific applications. These LCI databases have additionally been used both as internal and critically reviewed and published studies. Uncertainty introduced by the use of proxy data is reduced by using technologically, geographically, and/or temporally similar data. It is the responsibility of the modeler to appropriately apply the predefined material entries to the building under study.

System boundaries and delimitations

The analysis accounts for the full cradle to grave life cycle of the design options studied across all life cycle stages, including material manufacturing, maintenance and replacement, and eventual end of life. Optionally, the construction impacts and operational energy of the building can be included within the scope. Product stage impacts are excluded for materials and components indicated as existing or salvaged by the modeler. The modeler defines whether the boundary includes or excludes the flow of biogenic carbon, which is the carbon absorbed and generated by biological sources (e.g. trees, algae) rather than from fossil resources.

Architectural materials and assemblies include all materials required for the product's manufacturing and use including hardware, sealants, adhesives, coatings, and finishing. The materials are included up to a 1% cut-off factor by mass except for known materials that have high environmental impacts at low levels. In these cases, a 1% cut-off was implemented by impact.

Project Name
Full building summary

14.09.2023

Calculation Methodology

LIFE CYCLE STAGES

The following describes the scope and system boundaries used to define each stage of the life cycle of a building or building product, from raw material acquisition to final disposal. For products listed in Tally as Environmental Product Declarations (EPD), the full life cycle impacts are included, even if the published EPD only includes the Product stage [A1-A3].

Product [EN 15978 A1 - A3]

This encompasses the full manufacturing stage, including raw material extraction and processing, intermediate transportation, and final manufacturing and assembly. The product stage scope is listed for each entry, detailing any specific inclusions or exclusions that fall outside of the cradle to gate scope. Infrastructure (buildings and machinery) required for the manufacturing and assembly of building materials are not included and are considered outside the scope of assessment.

Transportation [EN 15978 A4]

This counts transportation from the manufacturer to the building site during the construction stage and can be modified by the modeler.

Construction Installation [EN 15978 A5] (Optional)

This includes the anticipated or measured energy and water consumed on-site during the construction installation process, as specified by the modeler.

Maintenance and Replacement [EN 15978 B2-B5]

This encompasses the replacement of materials in accordance with their expected service life. This includes the end of life treatment of the existing products as well as the cradle to gate manufacturing and transportation to site of the replacement products. The service life is specified separately for each product. Refurbishment of materials marked as existing or salvaged by the modeler is also included.

Operational Energy [EN 15978 B6] (Optional)

This is based on the anticipated or measured energy and natural gas consumed at the building site over the lifetime of the building, as indicated by the modeler.

End of Life [EN 15978 C2-C4]

This includes the relevant material collection rates for recycling, processing requirements for recycled materials, incineration rates, and landfilling rates. The impacts associated with landfilling are based on average material properties, such as plastic waste, biodegradable waste, or inert material. Stage C2 encompasses the transport from the construction site to end-of-life treatment based on national averages. Stages C3-C4 account for waste processing and disposal, i.e., impacts associated with landfilling or incineration.

Module D [EN 15978 D]

This accounts for reuse potentials that fall beyond the system boundary, such as energy recovery and recycling of materials. Along with processing requirements, the recycling of materials is modeled using an avoided burden approach, where the burden of primary material production is allocated to the subsequent life cycle based on the quantity of recovered secondary material. Incineration of materials includes credit for average US energy recovery rates.

PRODUCT	CONSTRUCTION	USE	END-OF-LIFE	MODULE D
A1. Extraction A2. Transport (to factory) A3. Manufacturing	A4. Transport (to site) A5. Construction Installation	B1. Use B2. Maintenance B3. Repair B4. Replacement B5. Refurbishment B6. Operational energy B7. Operational water	C1. Demolition C2. Transport (to disposal) C3. Waste processing C4. Disposal	D. Benefits and loads beyond the system boundary from: 1. Reuse 2. Recycling 3. Energy recovery

Life-Cycle Stages as defined by EN 15978. Processes included in Tally modeling scope are shown in bold. Italics indicate optional processes.

Project Name
Full building summary

14.09.2023

Calculation Methodology

ENVIRONMENTAL IMPACT CATEGORIES

A characterization scheme translates all emissions and fuel use associated with the reference flow into quantities of categorized environmental impact. As the degree that the emissions will result in environmental harm depends on regional ecosystem conditions and the location in which they occur, the results are reported as impact potential. Potential impacts are reported in kilograms of equivalent relative contribution (eq) of an emission commonly associated with that form of environmental impact (e.g. kg CO₂eq).

The following list provides a description of environmental impact categories reported according to the TRACI 2.1 characterization scheme, the environmental impact model developed by the US EPA to quantify environmental impact risk associated with emissions to the environment in the United States. TRACI is the standard environmental impact reporting format for LCA in North America. Impacts associated with land use change and fresh water depletion are not included in TRACI 2.1. For more information on TRACI 2.1, reference Bare 2010, EPA 2012, and Guinée 2001. For further description of measurement of environmental impacts in LCA, see Simonen 2014.

Acidification Potential (AP)

kg SO₂eq

A measure of emissions that cause acidifying effects to the environment. The acidification potential is a measure of a molecule's capacity to increase the hydrogen ion (H⁺) concentration in the presence of water, thus decreasing the pH value. Potential effects include fish mortality, forest decline, and the deterioration of building materials.

Eutrophication Potential (EP)

kg Neq

A measure of the impacts of excessively high levels of macronutrients, the most important of which are nitrogen (N) and phosphorus (P). Nutrient enrichment may cause an undesirable shift in species composition and elevated biomass production in both aquatic and terrestrial ecosystems. In aquatic ecosystems, increased biomass production may lead to depressed oxygen levels caused by the additional consumption of oxygen in biomass decomposition.

Global Warming Potential (GWP)

kg CO₂eq

A measure of greenhouse gas emissions, such as carbon dioxide and methane. These emissions are causing an increase in the absorption of radiation emitted by the earth, increasing the natural greenhouse effect. This may, in turn, have adverse impacts on ecosystem health, human health, and material welfare.

Ozone Depletion Potential (ODP)

kg CFC-11eq

A measure of air emissions that contribute to the depletion of the stratospheric ozone layer. Depletion of the ozone leads to higher levels of UVB ultraviolet rays reaching the earth's surface with detrimental effects on humans and plants. As these impacts tend to be very small, ODP impacts can be difficult to calculate and are prone to a larger margin of error than the other impact categories.

Smog Formation Potential (SFP)

kg O₃eq

A measure of ground level ozone, caused by various chemical reactions between nitrogen oxides (NO_x) and volatile organic compounds (VOCs) in sunlight. Human health effects can result in a variety of respiratory issues, including increasing symptoms of bronchitis, asthma, and emphysema. Permanent lung damage may result from prolonged exposure to ozone. Ecological impacts include damage to various ecosystems and crop damage.

Primary Energy Demand (PED)

MJ (lower heating value)

A measure of the total amount of primary energy extracted from the earth. PED tracks energy resource use, not the environmental impacts associated with the resource use. PED is expressed in energy demand from non-renewable resources and from renewable resources. Efficiencies in energy conversion (e.g. power, heat, steam, etc.) are taken into account when calculating this result.

Non-Renewable Energy Demand

MJ (lower heating value)

A measure of the energy extracted from non-renewable resources (e.g. petroleum, natural gas, etc.) contributing to the PED. Non-renewable resources are those that cannot be regenerated within a human time scale. Efficiencies in energy conversion (e.g. power, heat, steam, etc.) are taken into account when calculating this result.

Renewable Energy Demand

MJ (lower heating value)

A measure of the energy extracted from renewable resources (e.g. hydropower, wind energy, solar power, etc.) contributing to the PED. Efficiencies in energy conversion (e.g. power, heat, steam, etc.) are taken into account when calculating this result.

Project Name

14.09.2023

Full building summary

LCI Data

END-OF-LIFE [C2-C4]

A Life Cycle Inventory (LCI) is a compilation and quantification of inputs and outputs for the reference unit. The following LCI provides a summary of all energy, construction, transportation, and material inputs present in the study. Materials are listed in alphabetical order along with a list of all Revit families and Tally entries in which they occur, along with any notes and system boundaries accompanying their database entries. Each entry lists the detailed scope for the LCI data sources used from the GaBi LCI database and identifies the LCI data source.

For LCI data sourced from an Environmental Product Declaration (EPD), the product manufacturer, EPD identification number, and Program Operator are listed. Where the LCI source does not provide data for all life cycle stages, default North American average values are used. This is of particular importance for European EPD sources, as EPD data are generally only provided for the product stage, and North American average values are used for the remaining life cycle stages.

Where specific quantities are associated with a data entry, such as user inputs, energy values, or material mass, the quantity is listed on the same line as the title of the entry.

TRANSPORTATION [A4]

Default transportation values are based on the three-digit material commodity code in the 2012 Commodity Flow Survey by the US Department of Transportation Bureau of Transportation Statistics and the US Department of Commerce where more specific industry-level transportation is not available.

Transportation by Barge

Scope:
The data set represents the transportation of 1 kg of material from the manufacturer location to the building site by barge.

LCI Source:
GLO: Average ship, 1500t payload capacity/ canal ts (2017)
US: Diesel mix at filling station ts (2014)

Transportation by Container Ship

Scope:
The data set represents the transportation of 1 kg of material from the manufacturer location to the building site by container ship.

LCI Source:
GLO: Container ship, 27500 dwt payload capacity, ocean going ts (2017)
US: Heavy fuel oil at refinery (0.3wt.% S) ts (2014)

Transportation by Rail

Scope:
The data set represents the transportation of 1 kg of material from the manufacturer location to the building site by cargo rail.

LCI Source:
GLO: Rail transport cargo - Diesel, average train, gross tonne weight 1000t / 726t payload capacity ts (2017)
US: Diesel mix at filling station ts (2014)

Transportation by Truck

Scope:
The data set represents the transportation of 1 kg of material from the manufacturer location to the building site by diesel truck.

LCI Source:
US: Truck - Trailer, basic enclosed / 45,000 lb payload - 8b ts (2017)
US: Diesel mix at filling station ts (2014)

Project Name

14.09.2023

Full building summary

LCI Data (continued)

END-OF-LIFE [C2-C4]

Specific end-of-life scenarios are detailed for each entry based on the US construction and demolition waste treatment methods and rates in the 2016 WARM Model by the US Environmental Protection Agency except where otherwise specified. Heterogeneous assemblies are modeled using the appropriate methodologies for the component materials.

End-of-Life Landfill

Scope:

Materials for which no recycling or incineration rates are known, no recycling occurs within the US at a commercial scale, or which are unable to be recycled are landfilled. This includes glass, drywall, insulation, and plastics. The solids contents of coatings, sealants, and paints are assumed to go to landfill, while the solvents or water evaporate during installation. Where the landfill contains biodegradable material, the energy recovered from landfill gas utilization is reflected as a credit in Module D.

LCI Source:

US: Glass/inert on landfill ts (2017)
US: Biodegradable waste on landfill, post-consumer ts (2017)
US: Plastic waste on landfill, post-consumer ts (2017)

Concrete End-of-Life

Scope:

Concrete (or other masonry products) are recycled into aggregate or general fill material or they are landfilled. It is assumed that 55% of the concrete is recycled. Module D accounts for both the credit associated with off-setting the production aggregate and the burden of the grinding energy required for processing.

LCI Source:

US: Diesel mix at refinery ts (2014)
GLO: Fork lifter (diesel consumption) ts (2016)
EU - 28 Gravel 2/32 ts (2017)
US: Glass/inert on landfill ts (2017)

Metals End-of-Life

Scope:

Metal products are modeled using the avoided burden approach. The recycling rate at end of life is used to determine how much secondary metal can be recovered after having subtracted any scrap input into manufacturing (net scrap). Net scrap results in an environmental credit in Module D for the corresponding share of the primary burden that can be allocated to the subsequent product system using secondary material as an input. If the value in Module D reflects an environmental burden, then the original product (A1-A3) contains more secondary material than is recovered.

LCI Source:

Aluminum - RNA: Primary Aluminum Ingot AA/ts (2010)
Aluminum - RNA: Secondary Aluminum Ingot AA/ts (2010)
Brass - GLO: Zinc mix ts (2012)
Brass - GLO: Copper (99.99% cathode) ICA (2013)
Brass - EU-28: Brass (CuZn20) ts (2017)
Copper - DE: Recycling potential copper sheet ts (2016)
Steel - GLO: Value of scrap worldsteel (2014)
Zinc - GLO: Special high grade zinc IZA (2012)

Wood End-of-Life

Scope:

End of Life waste treatment methods and rates for wood are based on the 2014 Municipal Solid Waste and Construction Demolition Wood Waste Generation and Recovery in the United States report by Dovetail Partners, Inc. It is assumed that 63.5% of wood is sent to landfill, 22% to incineration, and 14.5% to recovery.

LCI Source:

US: Untreated wood in waste incineration plant ts (2017)
US: Wood product (OSB, particle board) waste in waste incineration plant ts (2017)
US: Wood products (OSB, particle board) on landfill, post-consumer ts (2017)
US: Untreated wood on landfill, post-consumer ts (2017)
RNA: Softwood lumber CORRIM (2011)

Project Name
Full building summary

14.09.2023

LCI Data

MODEL ELEMENTS

Revit Categories

Ceilings
Curtainwall Mullions
Curtainwall Panels
Doors
Floors
Roofs
Stairs and Railings
Structure
Walls
Windows

LCA

Worksets
Workset1

Phases
Demo
Existing
Phase 1
Phase 2
Phase 3

PRODUCT [A1-A3]

Materials and components are listed in alphabetical order along with a list of all Revit families and Tally entries in which they occur. The masses given here refer to the quantity of each material used over the building's life-cycle, which includes both Product [A1-A3] and Use [B2-B5] stages.

Additional provided data describing scope boundaries for each life cycle stage may be useful for interpretation of the impacts associated with the specific material or component. Each material or component is listed with its service life, or period of time after installation it is expected to meet the service requirements prior to replacement or repair. This value is indicated in parentheses next to the mass of the material associated with the listed Revit family. Values for transportation distance or service life shown with an asterisk (*) indicate user-defined changes to default values. Values for service life shown with a dagger (†) indicate materials identified by the modeler as existing or salvaged.

Paint, exterior metal coating, silicone-based, by area **70,4 kg**

Used in the following Revit families:
Circular Hollow Sections 70,4 kg (30 yrs)

Used in the following Tally entries:
Steel, HSS section

Description:
Silicone-based metal paint, with a default coating thickness of 100 microns

Life Cycle Inventory:
23% Binding agent
35% Pigments and fillers
40% Water
1.5% Organic solvents

Product Scope:
Cradle to gate, including emissions during application

Transportation Distance:
By truck: 642 km

End-of-Life Scope:
100% to landfill (plastic waste)

LCI Source:
DE: Application coating silicone (building, exterior, white) ts (2017)

Stainless steel, extruded, chromium 18/8 **8.264,4 kg**

Used in the following Revit families:
Circular Hollow Sections 8.264,4 kg (60 yrs)

Used in the following Tally entries:
Steel, HSS section

Description:
Stainless steel, extruded, Type 304 (Chromium 18/8)

Life Cycle Inventory:
100% Stainless steel

Product Scope:
Cradle to gate

Transportation Distance:
By truck: 431 km

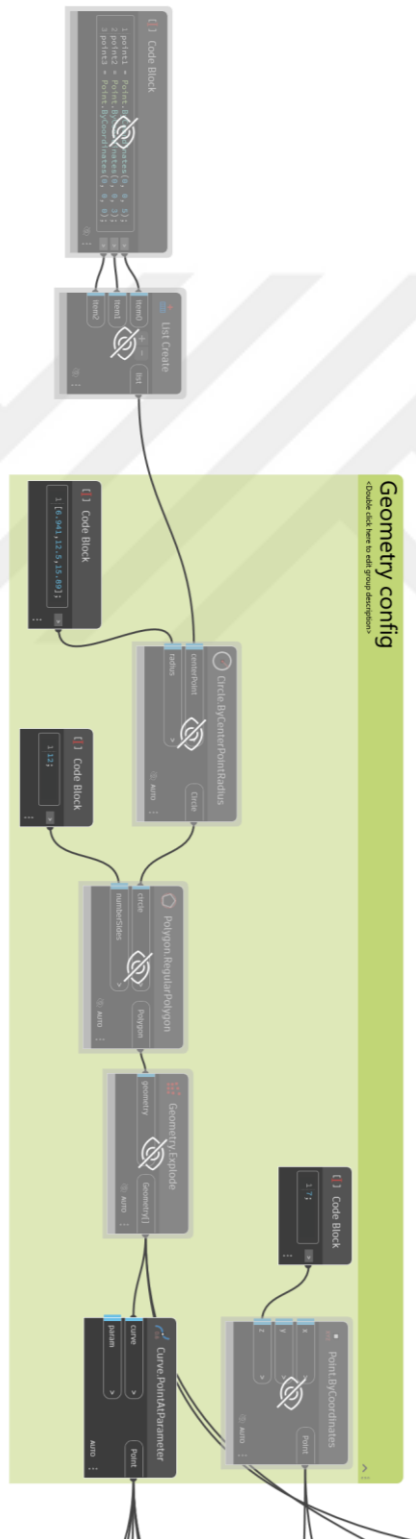
End-of-Life Scope:
98% Recovered
2% Landfilled (inert material)

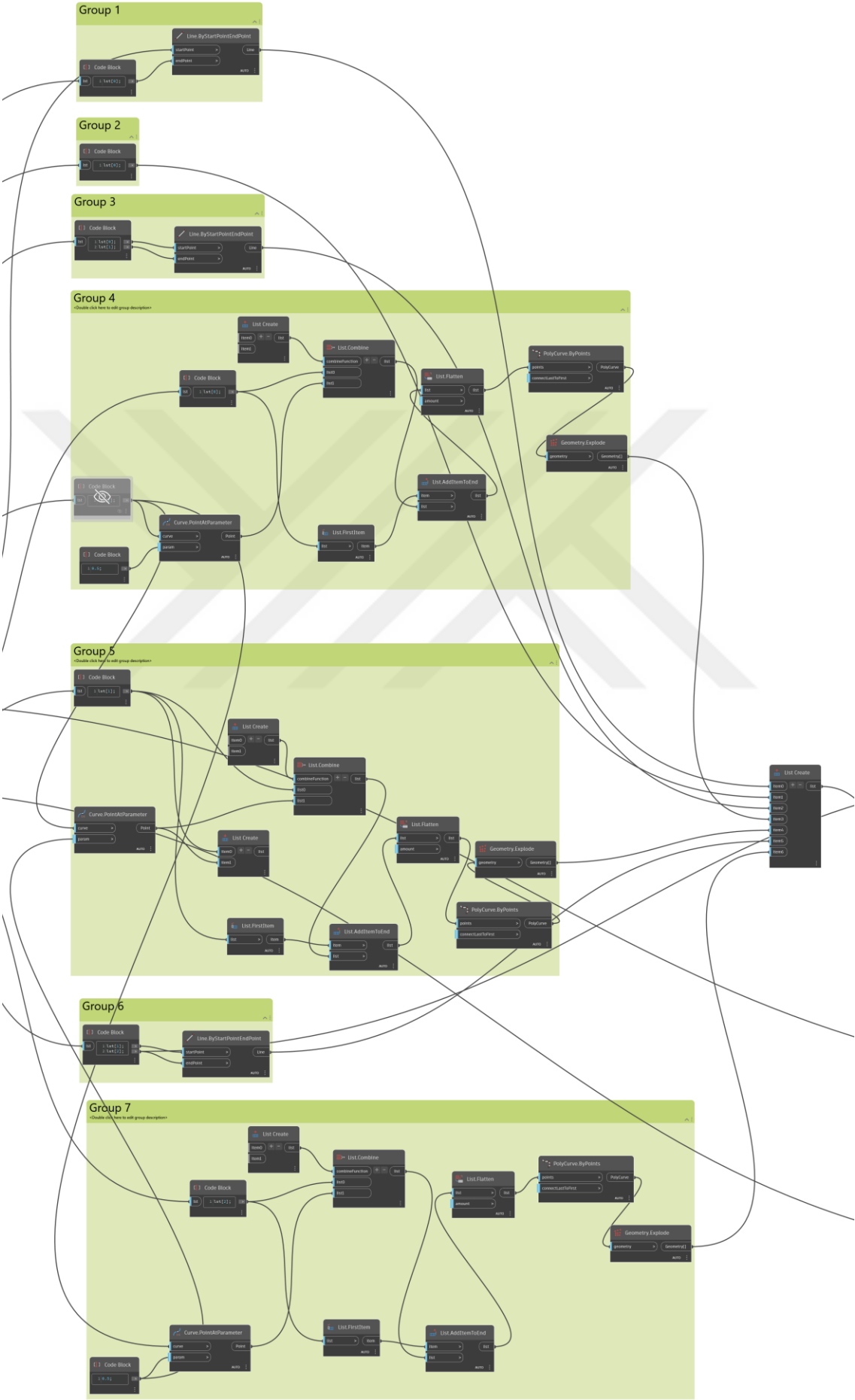
Module D Scope:
Product has 52% scrap input while remainder is processed and credited as avoided burden

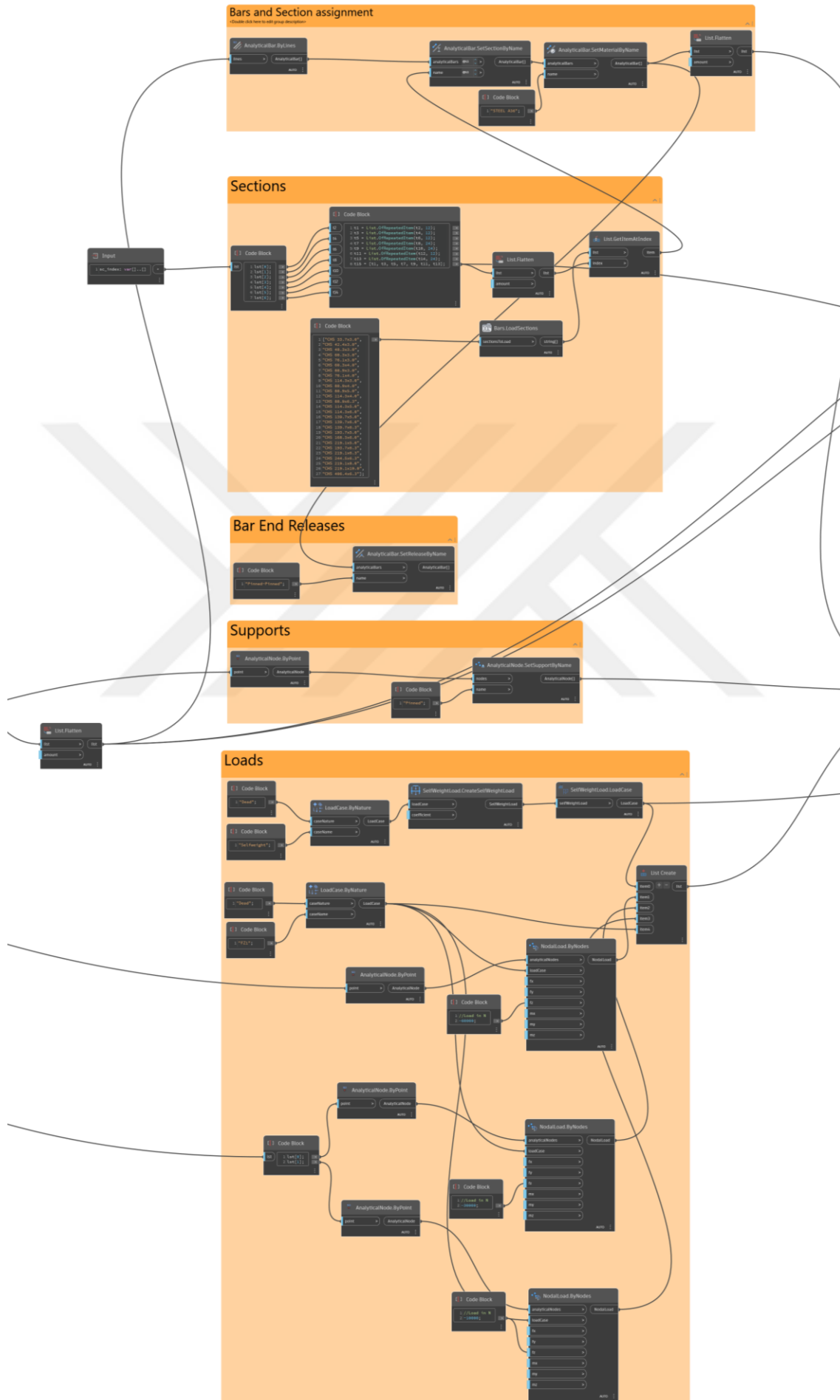
LCI Source:
RER: Stainless steel cold rolled coil (304) Eurofer (2010)
GLO: Steel sheet stamping and bending (5% loss) ts (2017)
US: Electricity grid mix ts (2014)
US: Lubricants at refinery ts (2014)
GLO: Compressed air 7 bar (medium power consumption) ts (2014)
RER: Stainless steel flat product (304) - value of scrap Eurofer (2010)

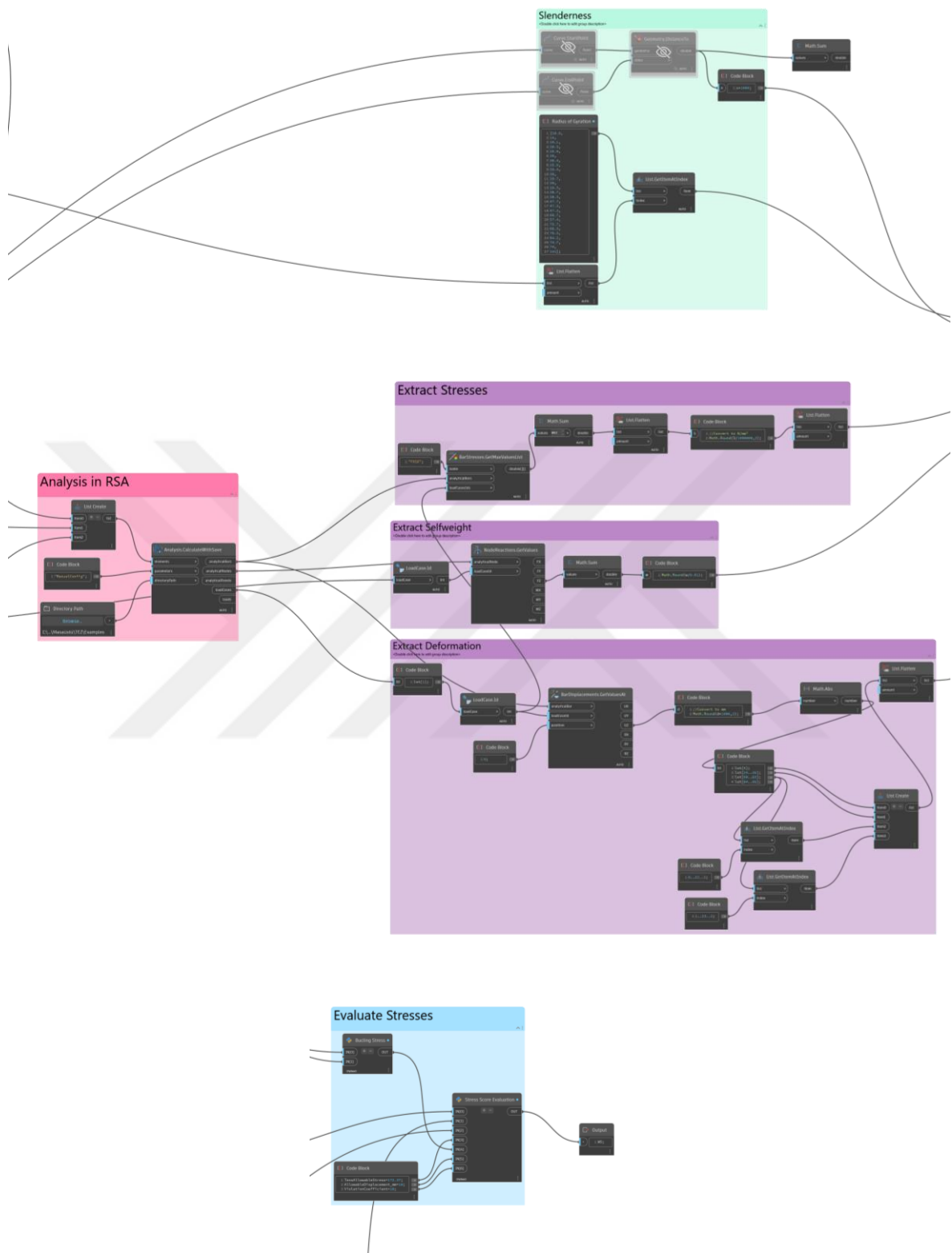
9.3. Appendix C, Dynamo Scripts

9.3.1. Dynamo Script to Create 120-Bar Dome Truss

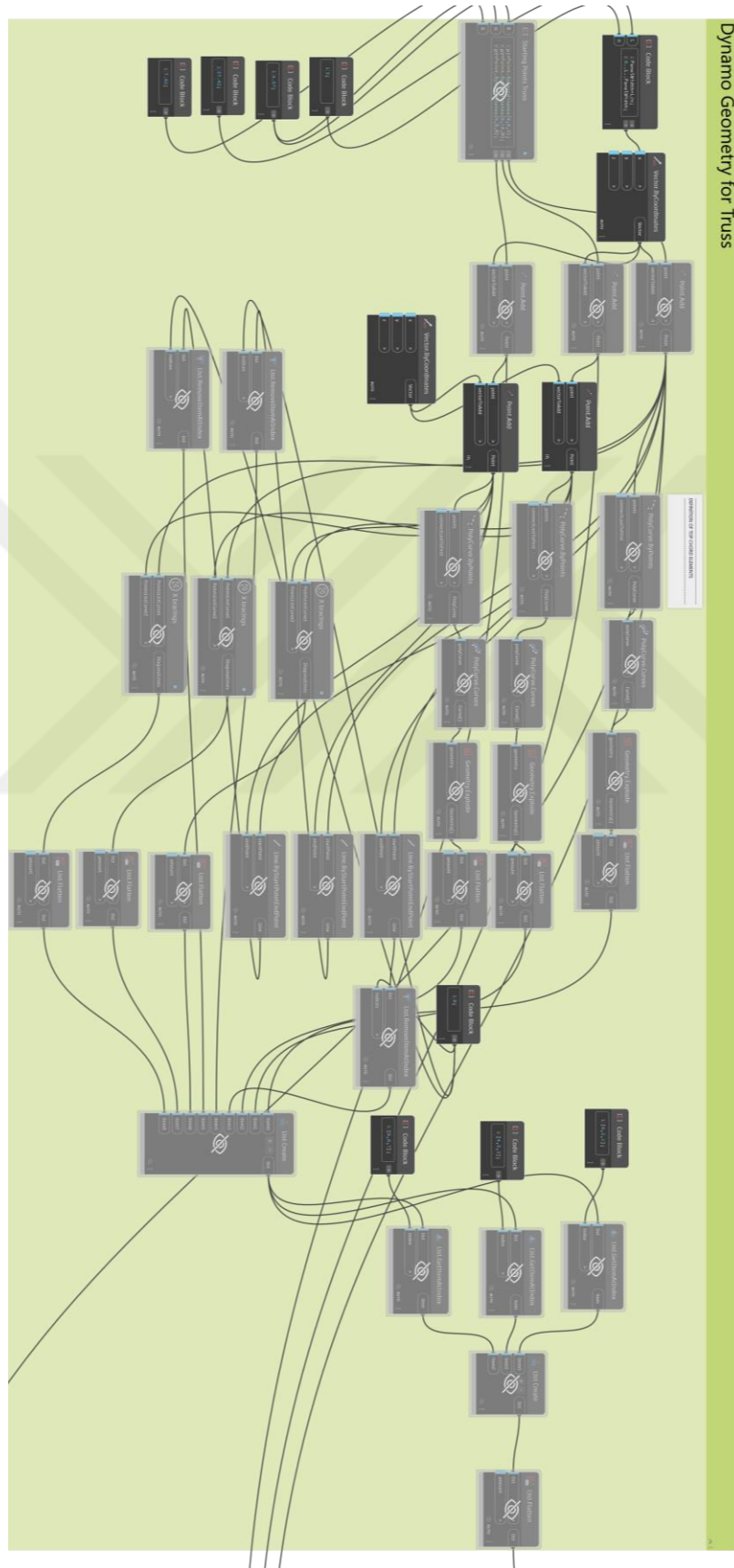


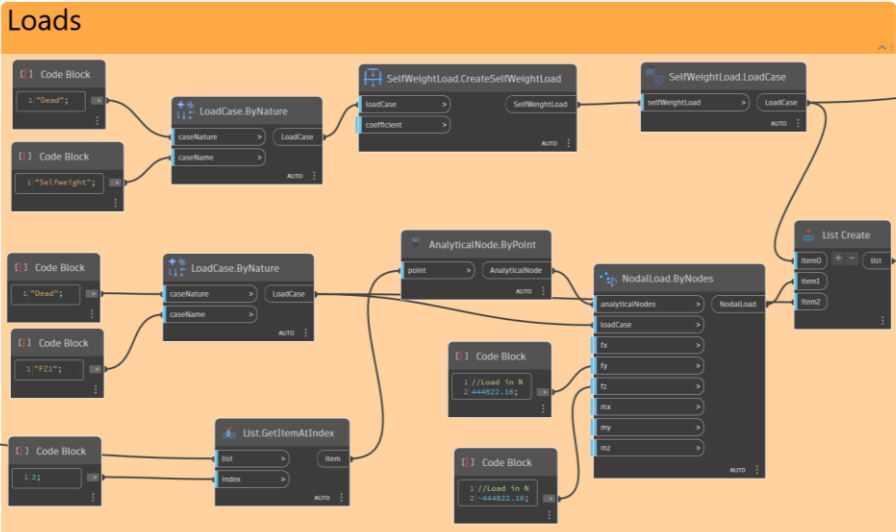


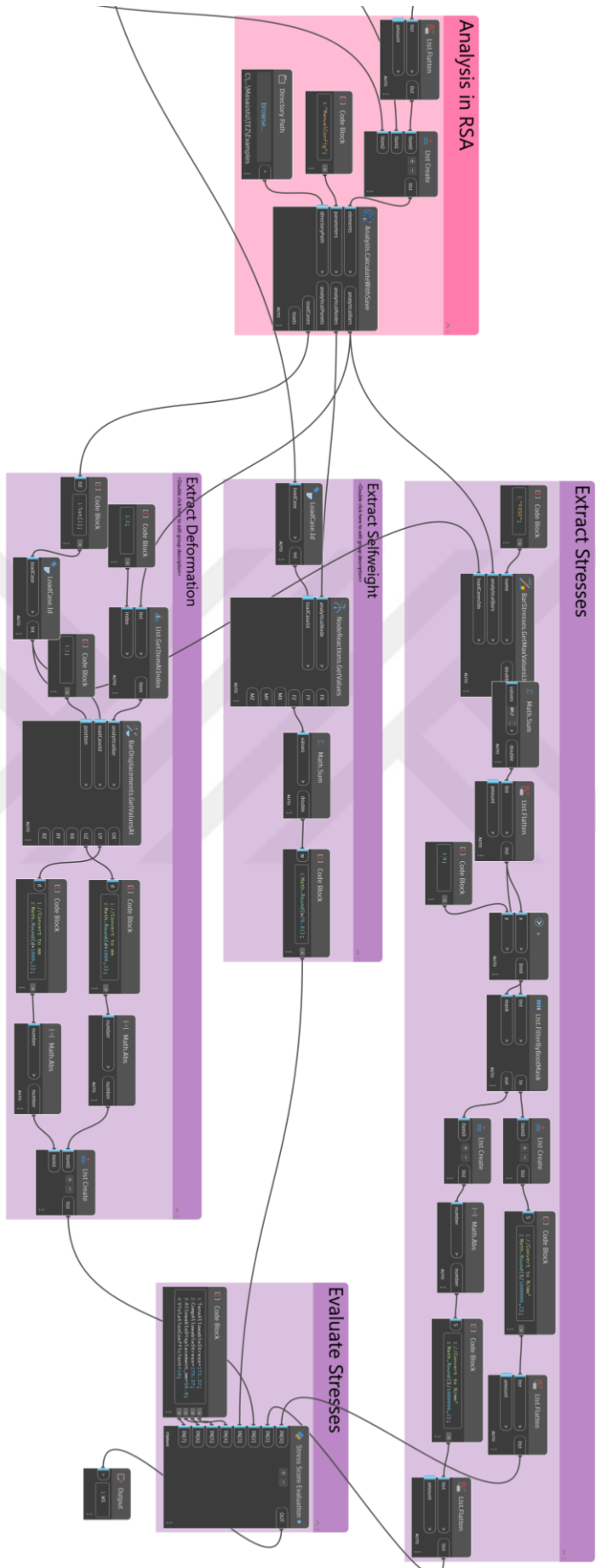




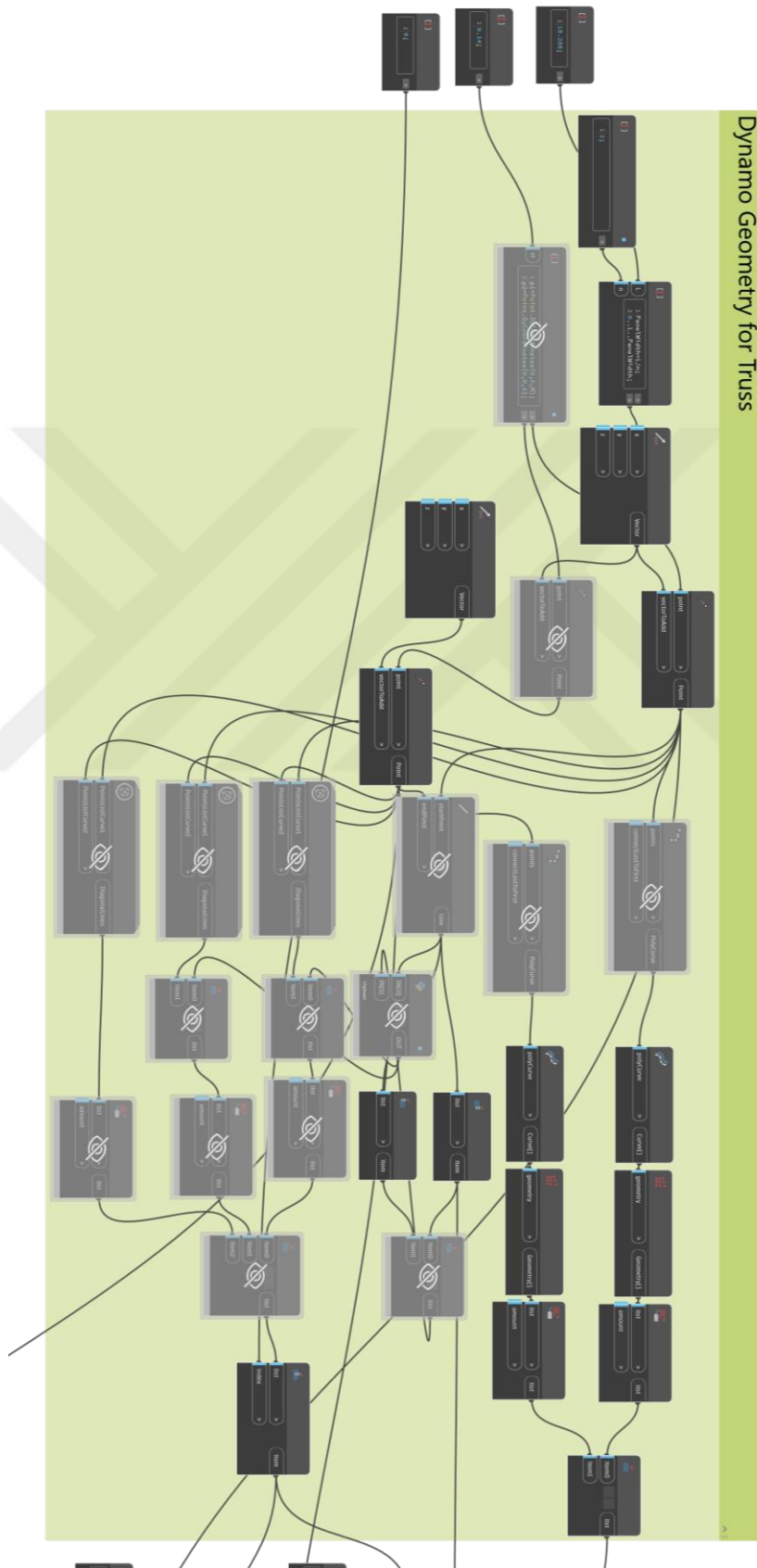
9.3.1. Dynamo Script to Create 36-Bar Truss

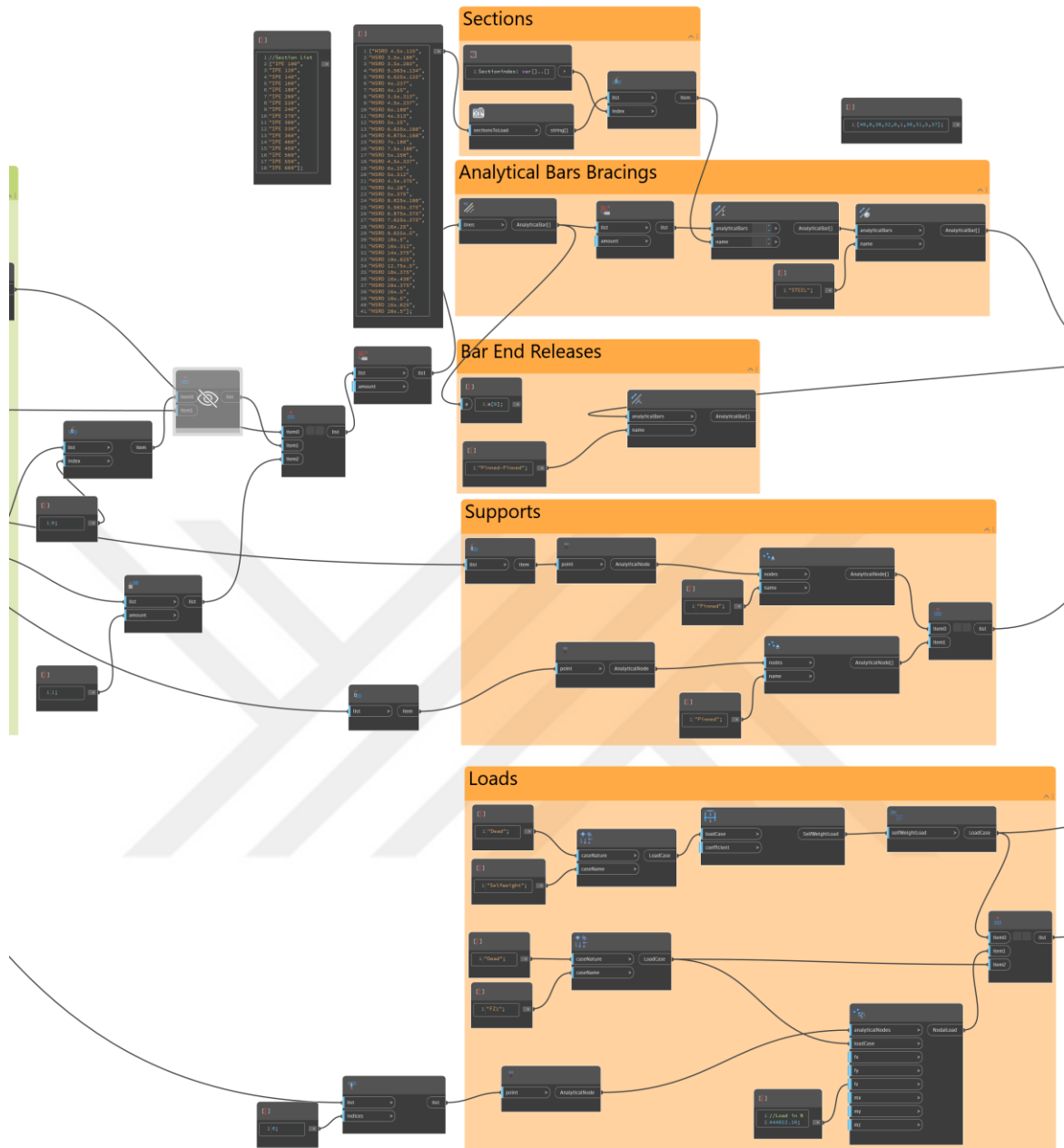


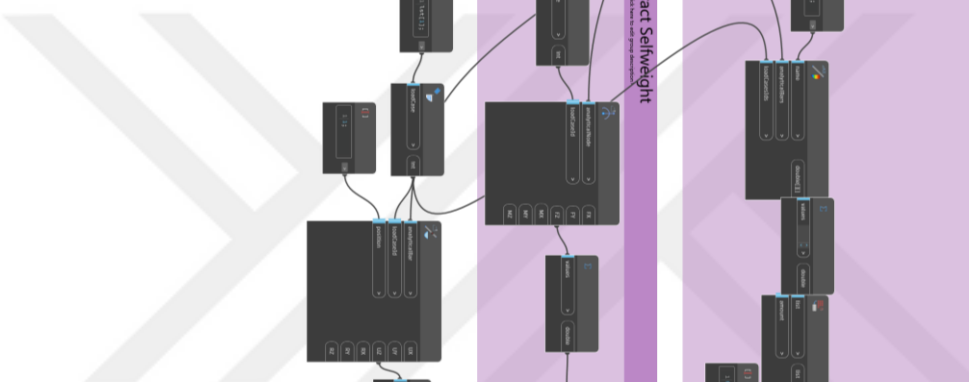




9.3.1. Dynamo Script to Create 10-Bar Truss







AUTOBIOGRAPHY

Feyzullah YAVAN, He completed his primary, middle school, and high school education in Trabzon. In 2016, after a year of preparatory education for English at the Sakarya University Department of Foreign Languages, he began his studies in the Department of Civil Engineering at Sakarya University. In 2018, he participated in the Erasmus program and studied for one semester in the Czech Republic. Later, he returned to Turkey and completed his undergraduate education as a first rank honor student. Following that, he undertook a 5-month internship in Germany through the Erasmus program for the second time.

After his internship, he pursued a master's degree in Project Management subject at Karadeniz Technical University. During his master's studies, he also worked as a research student under the mentorship of Prof. Dr. Reza Maleek at the Karlsruhe Institute of Technology in Germany. Currently, he continues to be involved in various projects at KIT and is working on his master's thesis.