CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Civil Engineering

Department of Concrete and Masonry Structures

Bachelor Thesis

Genetic Algorithm Optimization of Concrete Structures

Margarita Kuvaldina

Prague

May 2018

Supervisor: Ing. Martin Petřík, Ph.D.

Insert page signed by all parties aka zadani

Proclamation

I, Margarita Kuvaldina, hereby declare that I have prepared my Bachelor Thesis on the Genetic Algorithm Optimization of Concrete Structures, using only resources and literature mentioned in the bibliography.

Prague, Czech Republic

May 2018

....

Margarita Kuvaldina

Acknowledgements

On this page I would like to thank everyone who has supported me throughout this very intense writing process with the knowledge, advice and words of encouragement. And especially You, Ing. Martin Petřík Ph.D., for Your patience and guidance. Thank You.

Genetic Algorithm Optimization of Concrete Structures

Margarita Kuvaldina

Abstract

This abstract can include spoilers to the thesis. If you would like to enjoy the process of reading more, I kindly suggest you skipping it and getting back to it later on.

In the following work, the author has developed a code that uses the Finite Element Analysis to calculate structure's characteristics, such as deflections and stress in each member. In the next step she focused on optimizing chosen 2D truss structure to the best shape possible via Genetic Algorithm application. The optimization is performed on selected nodes locations and crosssections of each element.

In the first part of the work, the author discusses methods used to reach the goal. All of them were implemented via programming language Python and programming framework PyCharm.

Then a guide through the written code is provided. The main objective was to provide a reader with an excessive amount of information to create a similar code or to continue in a development of the existing one and expanding it to different usages, such as, for example, 3D structures.

Further, a few examples of the work are being discussed in detail to introduce the reader to how, by adjusting different parameters, the outcome can be changed. A discussion on the implications of the model is provided and options for the future work are outlined.

Author has decided to share her progress with others and have everything open-sourced to allow anyone to contribute and continue with the idea and build upon it. You can find the code on GitHub (`https://github.com/margkuval/Thesis`), a platform created mainly for code sharing. In case you shall have any questions related to the code, please address them using the GitHub platform. As for the thesis itself, you can find an online version on Research Gate `https://www.researchgate.net/profile/Margarita_Kuvaldina`. The author would kindly like to ask you for citing whenever you use any part of her code or thesis. Thank you for understanding.

Keywords

Optimization, concrete truss, genetic algorithm, civil engineering optimization, structural engineering, Python, Finite Element Analysis, Finite Element Method.

Contents

Chapter 1

Introduction

For hundreds of years, human kind was slowly reaching at first creation, and then adjustments in many different fields to come up with quality and efficient structures, thoughts and concepts.

And even though it would be incredible to continue and move our civilization's development forward in the same pace as it was during the times of industry revolutions, it is highly impossible to keep this curve of development with the same gradient for longer, at least in some fields. Looking into more efficient possibilities of improvement becomes more and more desirable these days. The approach of adjusting to current needs and looking on efficiency is very applicable in the field of civil engineering. Nowadays, we are reaching physical limitations in building construction. For example, in delivering the structures such as Burj Khalifa; pouring concrete to 850 meters in height, or the Akashi-Kaikyo bridge spanned over almost 2 kilometers [Miyata et al. 2002]. The concept of "regular" engineering with standard beam- and column- system, frames and truss structures is being challenged. At this point, more focus needs to be channeled into the optimization and maximization of structures potential.

Problems associated with optimum design have been the subject of considerable attention for a number of years. During the last several decades the field of optimum design has made remarkable progress in systems design, control of dynamics, and engineering analysis [Ertas and Jones n.d.].

1.1 Motivation

When choosing the topic of this work, three main components were considered - concrete as a material, optimization as a solution and difficulty of a task as a challenge to tackle.

Concrete is one of the most used building materials as its durability, bearing capacity and relatively low cost provide a great opportunity to widen its usage to many different construction designs. And even though the material itself is still considered to be less environmentally friendly, for now it is one of the best and most commonly used building materials. Taking this into the account, development of this thesis focused a lot on deepening the knowledge of structural design and widening the scope of different methods used especially for a concrete structures design.

Taking into the consideration information from a research, that has been conducted at the VTT Technical Research Centre of Finland, saying that better construction and use of buildings in the European Union would influence 42% of final energy consumption, about 35% of our greenhouse gas emissions and more than 50 percent of all extracted materials; and that it could also help to save up to 30% of water consumption[Ruuska and Häkkinen 2014]. Therefore, it is clearly visible that being able to use concrete in smaller amounts, we would help to lower the impact on Earth biosphere and conserve, potentially improve, current planet's state.

Developing this idea further has led me to uncover more about structures efficiency and optimization. When performance of a computation method improves and time needed for calculation is shortened, it also leads to a reduction in a number of human resources needed to conduct it. Moreover, not only HR optimization is being addressed but also the amount of material needed for the construction itself. We can see that increased efficiency in structural design affects many more fields and issues than only its main one.

The complexity of the problematic was also considered an advantage when deciding upon a larger exploration of the optimization and genetic algorithms usage. Different study majors interlinkage leads to a better, more challenging environment for students and widens their capability of critical thinking through introducing more complicated ideas for an analysis. Especially these days, I find this skill very useful and also exploring the knowledge limitations I have, was a great motivation to conduct the work.

1.2 Outline

At first, methods of structural design and its delivery that have been used in this work, are introduced. Followed by a discussion on why it is important to have at least basic knowledge in programming. Then, the implementation of GA itself is being introduced to the reader with concrete examples of application on case studies. Last but not least a discussion over results is performed with an outline of possible improvements for future work and exploration of other potential usage of genetic algorithms.

Chapter 2

Methods and methodology

Optimum design can be defined as the best possible design from the standpoint of the most significant effects - this means, minimizing the most significant undesirable effects and/or maximizing the most significant and desirable effects on design. [Ertas and Jones n.d.]. At first, it is important to define an objective. In other words, the focus of optimization. It can be many factors, such as, shape, weight and/or cost of the construction. Only after that the calculation shall be performed and used to its full potential.

For the truss problems further discussed in this thesis, many different methods of calculation and optimization can be used. They differ from the accuracy, through the clarity of calculation, down to the time consumption. In this work three main components have been used - Finite Element Analysis (FEA) to calculate stress and deflection of structures, Genetic Algorithm (GA) to provide comprehensive answer to the optimization problem and programming as a framework for an implementation of the two methods used for calculation performance.

2.1 Finite Element Analysis

Numerical analysis technique that receives much attention in engineering schools and in industry. Although originally developed to study stresses in complex airframe structures, it has since been extended and applied to the broad field of continuum mechanics. [Huebner et al. 2008]

2.1.1 History introduction

During the mid- to late-1950s and early 1960s a series of papers were published covering linear structural analysis and methods to explore efficient solutions to different structural problems. Turner et al delivered solutions to plane stress problems via triangular elements with properties determined from elasticity theory. As the digital computer evolved, these solution algorithms allowed the analysis of more complex problems [Champion 1992]. As it can be seen, Champion already spoke about connection of technology and advanced computation methods in early 1990s. Almost thirty years later there is an extensive difference in technology that we use as well as the advance of Finite Element Analysis (FEA) application. which is most obvious in engineering. Nowadays, it is very interesting to observe how computer's hardware requirements have also been discussed in the book and what kind of emphasis was brought on it.

2.1.2 Theory into practice

An analysis of a structure usually starts with its artificial split into smaller parts and only then the analysis of their behaviour in time is conducted. Structural elements are usually fragments that have a relatively simple shape (triangle, bar, node). Nodes, elements and degrees of freedom of the structure have global numbering that is used to identify them among others [Szmelter 1979]. Further, elements have also local inner forces numbering. Those are used for local element equations that are then via Global Stiffness Method transformed into global coordinates

Steps that are required to use this method can be split into three main steps. Firstly we need to introduce and define overall conditions; definition of the finite element mesh, generation of local Finite Element equations, and introduction of the exact boundary conditions. Second step introduces the assembly of the local element equations into global ones. This crucial step enables us to design a calculation for the whole structure and not only a small part of it. As the last, overall computation and solution derivation of the defined problem is performed and data are being analyzed.

FEA is possibly the most popular method of discretization available in engineering. Based on domain and boundary discretization, FEA reduces the infinite number of degrees of freedom of a continuum problem to a finite number of unknowns defined at element nodes.[Portela and Charafi 2012]

2.2 Genetic Algorithms

Technology went through great innovative times since Champion's book has been published. This evolution has been one of the prerequisites for further implementation development of FEA. We have been given the method that enables us to reduce our mental capacity usage on "simple steps" and provides us with a chance to focus on more advanced implementation. Great tool that allow us to make this step and make computation process even simpler are algorithms.

2.2.1 Brief introduction to Genetic Algorithm

In the last decade of twentieth century, genetic algorithm, which are stochastic optimization methods based on Darwin's evolution principle, became very popular and widely accepted methods that are nowadays commonly used [Hynek 2008].

Genetic algorithms (GA) represent modern optimization techniques. They are inspired by biological evolution, incorporating and adapting concepts like chromosomes, genes, natural selection, individual fitness, crossover combination or mutation. [Cazacu and Grama 2014]. The genetic analogy is maintained in the terminology used in the method. An initial ‘population’ is generated by random selection of the individual bits in a binary string of given length. [Jenkins 1991] In this work, individual is a node, which location is being optimized or a member’s cross-section.

2.2.2 What is the principle of its work?

Genetics algorithms draw direct inspiration from nature and seek to unlock the computational power of DNA. The rationale goes that if selective reproduction, natural selection, and random mutation could be responsible for the complexity of life on this planet, surely we can harness these principles to solve (or more accurately, approximate) NP-complete problems in polynomial time. [Park 2014]

If this process is repeated for many generations, better individuals (that is, a better solution to the problem) are preferred when choosing individuals for a crossover. Their characteristics spread within the population and combine with the characteristics of other proficient individuals. After a number of generations, then, in this way, populations with one or more individuals that correspond to an acceptable (or even optimal) solution to the problem, will usually be created. [Hynek 2008]

add something more?

2.2.3 Why was it used?

Genetic algorithms and genetic programming are very good at finding solutions to very large problems. They do it by taking millions of samples from the search space, making small changes, possibly recombining parts of the best solutions, comparing the resultant fitness against that of the current best solution, and keeping the better of the two. This process repeats until a stop condition like one of the following occurs: the known solution is found, a solution meeting all requirements is found, a certain number of generations has passed, a specific amount of time has passed, etc. [Sheppard 2016]

However, exciting process creating a GA and analyzing its results, is highly related to the framework that is used for its implementation. Choice of the framework can be crucial not only for the GA implementation - for some frameworks it is much easier to find experts that can be consulted or resources available on the topic. But also future work and reciprocity should be considered. It can be highly valuable to use a tool that is widely spread and will encourage others to reuse our outcomes.

2.3 Programming and why I used Python

While discovering different methods to approach the topic of the thesis, it was ultimately clear from the very beginning that computations will need to be performed by using programming. Not only it allows the highest freedom for methods implementation, but also code transformation, in case an error occurs, can be relatively fast. Moreover, programming provides an additional value, which can be described as "thinking about how computer thinks about my thinking". This expands the horizons and limits of our mind.

2.3.1 Why is programming important?

The basis for education in the last millennium was "reading, writing, and arithmetic"; now it is reading, writing, and computing [Sedgewick, Wayne, and Dondero 2015]. Authors continue with saying that it is an essential part of the education of every student in the sciences and engineering, but also stating that programming can be beneficial not only for students whose studies are connected to fields of mathematics and engineering. These days we can observe more and more, especially on generations that grew up with having technology under their pillow, that computer science becomes the basics. The basics for many different fields of studies and fields of our lives. We can see it on everyday things like smart phones usage, traffic lights programming so we can cross the street faster; to rocket science, something that is a reality for only a few. Technology nowadays is one of the only limitations we have to become even more proficient and achieve greater impact on the world.

Seeing all the possibilities via interlinking different majors is one of the advantages that programming offers and that is not yet as widely used as it could be and hopefully will be in the future. The message is that programming is not difficult to learn and that harnessing power of the computer is rewarding [Sedgewick, Wayne, and Dondero 2015].

If the general public, starting with children at schools, through teenagers, adults, but also elderly, would be more aware of what programming can do for them, how the basic thoughts and concepts can be developed further, their limits of thinking would stretch tremendously.

2.3.2 What are differences between languages?

It might sound like a cliche but programming languages can be compared to international tongues we widely learn at schools. Some of them are very popular and commonly used, others not. Also different languages have distinct sets of words that usually don't allow us to fully describe everything in just one word. As well as speaking one language does not necessarily mean that one will understand all other languages, though learning some might be an advantage for learning a few other ones.

Also, since computers are binary machines that recognize only zeros and ones and each instruction has to be written as binary digits (bits).[Ben-Ari 1996] All the commands that used types have to be translated into bits. Many people joke that programmers are those that type 0101 but the truth is that they type a command and the programming language knows how to translate an information received from user (command) into a language that our computers understand (bit).

The amount of information, and/or functionality, different languages have in one command usually differs. For a visual comparison a simple example is shown below for three programming languages - C, C++, Python and MATLAB syntax. All are asking computer to print "Hello world!" phrase.

C

```
#include
int main(void)
{
puts("Hello,
world!");
}
```

C++

```
#include
int main()
{
std::cout
<< "Hello,
world!";
return 0;
}
```

Python

```
print "Hello,
world!"
```

MATLAB

```
disp ('Hello,
world!')
```

As we can see, complexity of codes is clearly distinguishable and this simple example also reflects the difficulty and directness of each language for this particular task.

2.3.3 Why Python?

Python code designed in 1990 by Guido van Rossum [Sanner 1998] was created to be readable, and hence, reusable and maintainable much more than traditional scripting languages [Lutz 2008]. In practice it means that the code is not only easy to read even if you are not the author, but also easy to duplicate and reciprocate. User can easily understand someone else' code and it is also clear, compared to some other languages, what different functions mean and where they come from.

Moreover, having many technical advantages, for example, possibility of running on different platforms, supporting libraries (NumPy, SciPy etc.), having a simple syntax, dynamic typing, lack of compile steps, and built-in tool set, makes it a great candidate as a programming language. Comparably to other languages it is also much more concise. Python code is typically one-third to one-fifth the size of equivalent C++ or Java code [Lutz 2008]. In this work the class and object programming has been greatly used.

All these traits were important for decision making about which language to choose for creating a program including methods such as genetic algorithms are. So not only because of implementable components and functions the Python have been chosen but also because of the simplicity of its syntax.

Python's popularity in engineering application has been increasing for many reasons. Firstly, it has been designed as a language that is accessible for everyone to download. Programs themselves can be written on many different platforms that are for free as well. And since some other platforms used for engineering calculations, for example MATLAB, can be quite costly when used outside of scholar environment. And secondly, another great reason to use Python in engineering is that it is designed for programming in general, not calculations only.Learning how to operate with it gives much more opportunities for other than computing implications. With this, coming back to how important is to understand interconnections between disciplines, Python incorporates many of the aspects needed for a good further professional development and curriculum vitae (CV) buildup.

Chapter 3

Introduction to case studies

3.1 Introduction to the GA implementation

To assure code work-ability, at first, a basic triangle structure was selected. Triangle is a shape that repeatedly occurs in construction and it was highly valuable to learn how to implement a new calculation method on a figure of this kind. In the following paragraphs a guide through the written code is provided. The main objective is to give reader the excessive amount of information to create a similar code or continue in a development of this one.

Reasons for creating this project were to also learn how to alter not only the node locations, and thus lower the stress and deflections, but also adjust members' cross-section areas. In other words the main objective was to optimize the structure, so its shape would be the most effective, as well as, the lightest to assure sustainable design.

Starting with the structure's movement, in all case studies is has been restricted by two supports allocated at the bottom (see Figure 3.1). One is restricting the movement to both x- and y- direction (pin) and the second in y-direction (roller). Since they are fixing three degrees of freedom (DOF), structure is statically determinate. Code representation of this information can be found in Listing 1.

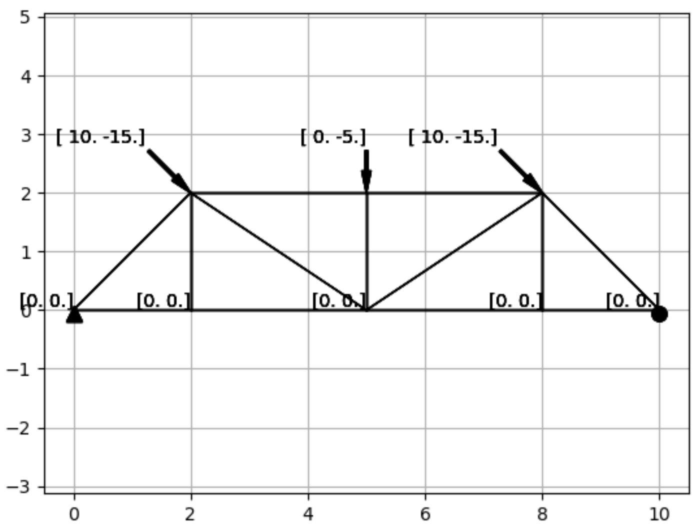

Figure 3.1: Truss structure 1x4

```
def calculation(self):
    numelem = len(self.mem_begin)  # count number of beginnings

    """Structural characteristics"""

    "Fixed Degrees of Freedom (DOF)"  # CH
    dof = np.zeros((2 * len(np.unique(self.mem_begin)), 1))  # dof vector
    # counts unique values in mem_begin
    dof[0] = 1  # 1 = fixed
    dof[1] = 1
    dof[9] = 1
```

Listing 1: GA: Degrees of freedom in structure

Overall, the code for this work is compiled out of four main elements:

- **Solver** that is implementable on any structure, no adjustments needed. Created to solve deflections, stresses and weights of the structure.
- **GA** where structure parameters and characteristics are defined, as well as, all the algorithm with all the actions typical for the GA - fitness function assignment, crossover and mutation. In this part solver is being called and based on the output fitness is assigned.
- **Brain** is the part that is being run and knows the sequence of actions and/or functions that are defined in the GA. It also calls the last part, that is plotting.

- **Plot** was created to isolate visual interpretation of the code from its crucial computing parts.

These four together create a system that is able to automatically calculate previously stated variables, such as, deflections, stress and weight of the construction and based on work with those, find the most optimal design. To fully understand the process, all parts will be now described.

3.2 Solver

It is important to mention two things about matrices before the start. Matrices in Python have a predefined horizontal shape so when creating a list of X numbers in a matrix, the matrix will be created as 1 by X. And the second thing is that matrix start with the first position numbered as zero. For example in a matrix [2, 4, 5], the number "2" is on a position number "0" and "4" on a position "1". Influence of these two rules can be observed on many parts of the thesis code.

In the process of designing the triangle truss, at first, nodes were defined by creating a matrix (array in programming terms). Not supported bottom nodes' locations in x direction have been enabled to be adjusted and, at first, a population was created with their locations randomly generated from the interval of +-0.7 meters to each side in x direction from the original location. This way it could be assured that fraud results would not occur and they will be in a reasonable range for creation of a statically well designed structure.

```
        "Structural dimentions (m)"""
        # CH = change if implementing on a new structure
        a = 2   # CH
        h = a   # triangle height   # CH

        "Original coordinates"
        xcoord = np.array([0, a, 2.5 * a, 4 * a, 5 * a, a, 2.5 * a, 4 * a])   # CH
        ycoord = np.array([0, 0, 0, 0, 0, h, h, h])   # CH

        "Choose a random number in range +- (m) from the original coordinate"
        x2GA = rnd.randrange(np.round((xcoord[2] - 0.7) * 100),
                             np.round((xcoord[2] + 0.7) * 100)) / 100   # CH
        x1GA = rnd.randrange(np.round((xcoord[1] - 0.7) * 100),
                             np.round((xcoord[1] + 0.7) * 100)) / 100
        x3GA = rnd.randrange(np.round((xcoord[3] - 0.7) * 100),
                             np.round((xcoord[3] + 0.7) * 100)) / 100

        "New coordinates"
        xcoord = np.array([0, x1GA, x2GA, x3GA, 5 * a, a, 2.5 * a, 4 * a])  # CH
        ycoord = np.array([0, 0, 0, 0, 0, h, h, h])  # can use np.ix_?      # CH
```

Listing 2: GA: Original and new nodes' location determination

Next step to a full structure creation was to use a connectivity matrix that assures that nodes are connected in a sequence that is defined by a user. This matrix works autonomously without a need for rewriting every time the structure is changed or adjusted. The only variable that need to be manually re-written are member beginning- and end-nodes. `mem_begin, mem_end`.

```python
class GA:
    def __init__(self, pop):
            self.mem_begin = np.array([0, 1, 2, 3, 4, 7, 6, 5, 1, 5, 6, 2, 7])
                # beginning of an edge    # CH
            self.mem_end   = np.array([1, 2, 3, 4, 7, 6, 5, 0, 5, 2, 2, 7, 3])
                # end of an edge          # CH
```

Listing 3: GA: Nodes connection determination

As it can be seen on a previous Listing 3, the first member starts at a node number "0" and ends at a node number "1". Symbol "CH" written as a code comment means that this variable needs to be changed if different structure in being analyzed.

```python
def deflection(xcoord, ycoord, mem_begin, mem_end, numelem, E, A, F, dof):

    "Link x and y coordinates with member's beginning and end"""
    xi = xcoord[np.ix_(mem_begin)]
    xj = xcoord[np.ix_(mem_end)]  # take mem_end and replace with corresponding xcoord
    yi = ycoord[np.ix_(mem_begin)]
    yj = ycoord[np.ix_(mem_end)]

    "Connectivity matrix"
    ij = np.vstack([[2 * mem_begin, 2 * mem_begin + 1],
                    [2 * mem_end, 2 * mem_end + 1]]).transpose()
```

Listing 4: Solver: Connectivity matrix assuring correct node connection process

To keep the consistency in code, the only two sheets that need to be adjusted are GA and Brain. Solver and Plot work anonymously without any additional input needed. This was chosen to simplify the process when changing to a different structure, therefore we can see, that listings stated above are mainly from the GA, even though the values are worked with in Solver.

As was discussed in chapter Methods and Methodologies, global stiffness method (a part of Finite Element Analysis) has been applied. Each member's stiffness was calculated and, based on that, inner forces and stress in each member. To create a global stiffness matrix a new, empty matrix has been created as it can be seen on Listing 5, line 2. Its size is based on the number of total degrees of freedom that the structure has. "For" function (Listing 5, lines 7 - 17) is responsible for the creation of all local stiffness matrices and then the last (18th) line is in charge of mapping them back to the right locations in the global stiffness matrix.

```python
"""Global Stiffness Matrix"""
glob_stif = np.zeros((dof_tot, dof_tot))  # empty Global Stiffness Matrix
length = np.sqrt(pow((xj - xi), 2) + pow((yj - yi), 2))  # defines length of members
c = (xj - xi) / length  # cos
s = (yj - yi) / length  # sin

for p in range(numelem):
    # takes p from the range of numelem 1 by 1 and creates multiple k1 (local) matrices
    # at the end maps k1 matrices on right places in glob_stiff matrix
    n = ij[p]
    cc = c[p] * c[p]
    cs = c[p] * s[p]
    ss = s[p] * s[p]
    k1 = E[p] * A[p] / length[p] * np.array([[cc, cs, -cc, -cs],
                                             [cs, ss, -cs, -ss],
                                             [-cc, -cs, cc, cs],
                                             [-cs, -ss, cs, ss]])
    glob_stif[np.ix_(n, n)] += k1
```

Listing 5: Solver: Global Stiffness Matrix creation

The key step in derivation of the truss element stiffness matrix is to write equilibrium equations that relate the nodal displacements to nodal forces [Huebner et al. 2008]. The following code lines in Listing 6 are devoted to exactly this step. Dependency of inner forces on the stiffness and deflection of the member is defined as following 1) global stiffness * u = F.

```python
    "Solve deflections"
    u = np.zeros((dof_tot, 1))  # empty deflections matrix; 1 = number of columns
    u1 = np.linalg.solve(glob_stif[np.ix_(dof_active, dof_active)],
                         F[np.ix_(dof_active)]) # solve equation glob_stif*u = F
    u[np.ix_(dof_active)] = u1  # map back to the empty deflection matrix
    deflection = np.round(u, 3)

    return deflection
```

Listing 6: Solver: Deflections

The second equation correlates inner forces in a member and their cross-section areas with stress in each element. The relationship is defined in the following manner: 2) stress = Flocal / A.

```
    """Stress calculation"""

    "Deflections in x, y directions"
    # using mem_end and mem_begin to calculate new nodes location
    length = np.sqrt(pow((xj - xi), 2) + pow((yj - yi), 2))  # members length
    k = E * A / length

    u = deflection
    uxi = u[np.ix_(2 * mem_begin)].transpose()
    uxj = u[np.ix_(2 * mem_end)].transpose()
    uyi = u[np.ix_(2 * mem_begin + 1)].transpose()
    uyj = u[np.ix_(2 * mem_end + 1)].transpose()

    "Inner forces"
    c = (xj - xi) / length  # cos
    s = (yj - yi) / length  # sin

    Flocal = k * ((uxj - uxi) * c + (uyj - uyi) * s)

    "Stress (kPa)"
    stress = Flocal[0] / A
    stress_normed = [i / sum(abs(stress)) for i in abs(stress)]

    xinew = xi + uxi[0]  # [[ in u array, now solved by taking "list 0" from the MAT
    xjnew = xj + uxj[0]
    yinew = yi + uyi[0]
    yjnew = yj + uyj[0]

    return stress, stress_normed, xi, xj, yi, yj, xinew, xjnew, yinew, yjnew,
           F_x2, numnode, dof_x2, length
```

Listing 7: Solver: Stress

Weight of each member has been calculated as well. Reinforced concrete type C30/37 has been chosen and if a member was designed as a steel one, steel type S235 was chosen with a volumetric mass density equal to 7850 kg/m3. If cross-sections would be reduced to dimensions where steel reinforcement would become (percentage wise) more powerful and could influence calculations because of its higher density, and therefore weight, higher wight, adjustments shall be made and the density matrix shall reflect that.

```
    "Weight calculation"
    weight = length * A * ro

    return weight
```

Listing 8: Solver: Wight

Paragraphs above described a structure of the Solver that computes stress, deflections and weight of members. Calculated values will be used in the genetic algorithm. This is a very important step towards the optimal structure design.

3.2.1 Comparison to SCIA

Experiment started with a simply supported triangle to tryout the code, link nodes to each other and create the basic inner forces code solver. After learning needed skill and creating a working code, calculation has been extended to the Truss 1x4 and then compared with a result generated from SCIA Engineering - structural software. The calculation has been performed through the truss module.

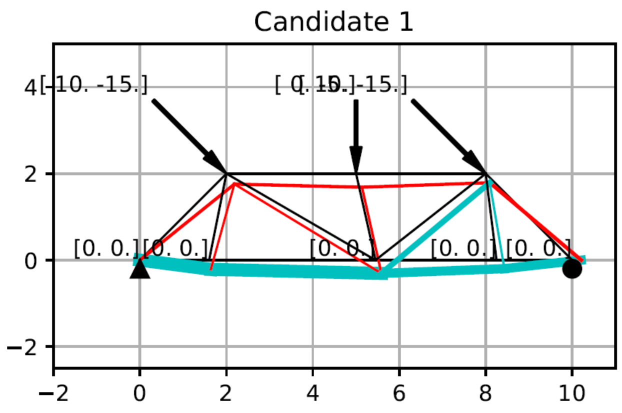

Figure 3.2: SCIA generated stress distribution

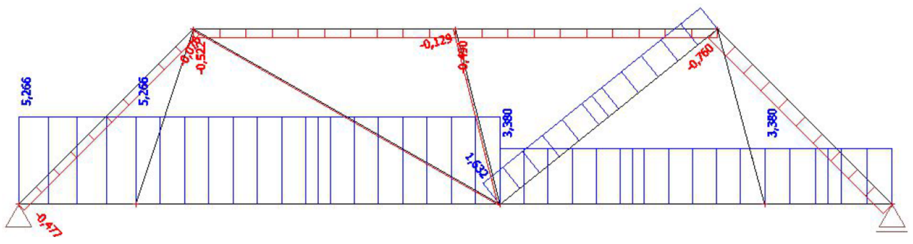

Figure 3.3: SCIA generated stress distribution

Two computation methods comparison							
Members	0	1	2	3	4	5	6
Areas (m2)	0.0064	0.0064	0.0064	0.0064	0.04	0.04	0.04
Stress Solver (MPa)	5.172	5.237	3.396	3.308	-0.759	-0.495	-0.517
Stress SCIA (MPa)	5.266	5.266	3.380	3.380	-0.760	-0.490	-0.522
Members	7	8	9	10	11	12	-
Areas (m2)	0.04	0.04	0.04	0.04	0.0064	0.0064	-
Stress Solver (MPa)	-0.478	0.003	-0.079	-0.139	1.646	0.037	-
Stress SCIA (MPa)	-0.477	0	-0.076	-0.129	1.632	0	-

Figure 3.4: Comparison of Solver and SCIA calculation

As you can see on a Figure 3.4, results are comparable and deviations are fewer than 1 MPa. The difference between them is most likely to be caused by a discrepancy in the modulus of elasticity of concrete in both methods.

3.3 Genetic Algorithm Implementation

After Genetic Algorithm is run from the Brain, it will start with creation of a randomly generated population based on user's parameters. For example that new nodes can be located in a radius of maximum 0.7 meters from the original one. After this step is done, it accesses Solver and runs the whole population through the computing mechanism. The output is then used to rate individuals. This is done by a fitness function.

3.3.1 Fitness Function

The objective of the optimization is to minimize the total mass of the structure while keeping it below the maximum allowable stress and displacement. The stresses and displacements evaluation of candidate solutions is done using the finite element method for the case of plane truss structures [Cazacu and Grama 2014]. The main goal of the fitness function is to screen through the whole population and rate each individual based on its qualities. Those usually vary with the aim of the task assigned. The other objective of the fitness function is to determine whether the optimization is working well or if the algorithm moves in a wrong direction. It evaluates how much the current structure developed and if there is a visible trend in its development. Figure below is an illustration of how the output can look like.

Fitness is a function that makes it possible to organize all potential solutions in terms of their suitability without major problems [Hynek 2008]. A fundamental aspect of solving problems using genetic algorithms is that they must provide feedback that helps the engine select the better of two guesses. That feedback is called the fitness, for how closely the guess fits the desired result. More

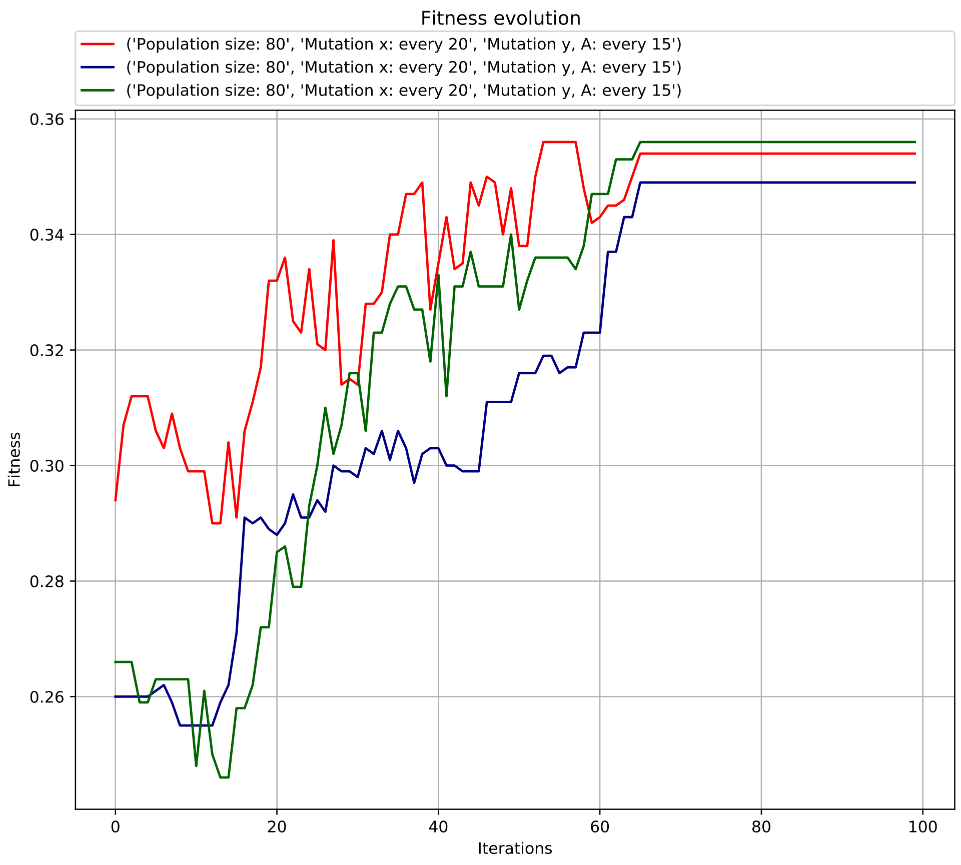

Figure 3.5: Three fitness evolution in time, truss 1x4

importantly it implies a general progression. [Sheppard 2016]

Generally speaking, structural design is required to conform to a number of inequality constraints related to stress, deflection, dimensional relationships and other variables. In the genetic algorithm these are conveniently handled by a ‘penalty’ approach where a constraint violation is penalized to deter the future use of that set of parameters. The levels of penalties imposed can be ‘tuned’ so that minor infringements suffer small penalties whereas major violations are penalized out of existence. [Jenkins 1991]. In the code developed in this thesis, penalization has been used for members which stress overcomes the border of the modulus of elasticity of either concrete or steel based on the location of the member. This penalization moved the member to the last place within the range of calculated individuals.

Overall, to summarize, fitness function helps to determine whether generated solution is moving in a desirable direction, algorithm works well, and results improve with each iteration. It simply knows what is needed from the calculation and can rate the result based on the importance of variables set within.

For all observed case studies in this thesis, the best fitness is the highest number possible. The number consists of three components. All three of them are multiplied by coefficients that determine whether it is highly valuable for the structure to have the component's value as low as possible or if other components are of the higher priority and that one is only complementary. In our example the highest priority has been allocated to the sum of absolute stress in structure (0.5), followed closely by the sum of absolute deflections (0.4) and as the lowest, the weight has been set (0.1). Distribution can be adjusted at any point and, for example, in case considering the lowest price of the construction, weight can be set as a very high priority component.

Fitness for each case study

Characteristics for each calculation

To summarize, the basic thought behind the fitness function is to rate each individual in population based on specified factors, and hence find out the "goodness" of each of them so the best can be more likely picked for the future development.

In this work fitness function was designed as following.

$$f(x) = 1/(\sum |deflection| * coef_d + \sum |stress| * coef_s + \sum weight * coef_w)$$

Since deflections, stresses and weights had all different scales(amount of numbers after the comma) and it was difficult to compare them between each other, sums were normalized. In other words they were reduced(transformed, changed, adjusted) to the same scale, so they would actually be comparable.

Coefficients were based on the importance of factors in the design. For example stress and deflection have been identified as two focal points that will be the most desirable to lower for this work. Weight coefficient has been chosen significantly four to five times lower that than the other two constants since weight has been taken as a complementary characteristics for the optimization. Sum of all coefficients equals to one and the highest value is for stress = 0.5.

3.3.2 Crossover

The general trend should be simple, best individuals' genes stay in the population.

Let's imagine two parent personas with two different sets of chromosomes. Randomized choice is applied to detect a point inside both sets (splitting point) where a split into two parts is initiated and chromosome parts are interchanged between each other. Since parent sets are split in one location only, this kind of crossover is called single-point crossover. [Panchenko 2007]

Possible other solution for a split is a multiple-point crossover, when multiple points in sets are selected and multiple parts of a chromosome line are interchanged. As an example two matrices with three members can be used [0, 1, 2] and [3, 5, 7]. In a multiple-point crossover one of the solutions is to switch 1 and 5. This particular case is an example of a two-point crossover. In the case study the single point crossover has been used, mainly because the switch of so called "chromosomes" was in the nodes location and since it is a 2D structure only two dimensions are present, an implementation of more than a single point crossover is not needed.

Crossover types division can be also based on the identification of the split location - how the location of a split point is determined. Few of the types are uniform, triadic, shuffler crossover and crossover with reduced surrogate. In this work aspect of the localization of a split did not have to be worked on, for the same reason as the situation stated above - the structure is a 2D object. If a structure would ever be a 3D, this problematic would have to be raised and covered.

As it could be seen above, the crossover, through combination of two different parents, generates new solutions as a next population. From the previous statement it can be also seen that the number of crossings between a limited population is finite [Panchenko 2007]. Thus, possibilities of reaching the best solution are fully dependent on a size of a population and considering that this can lead to a local minimum or maximum localization, which, in this case, can be taken as the ultimately best solution, even if a global minimum or maximum is present on a different location, a possibility to introduce new deformed and unique chromosomes should be assured. This is a moment when a mutation is introduced in a code.

3.3.3 Mutation

The whole process can be seen as a parallel with a biological process of a human population development. If one limited amount of humans would, for a certain amount of generation, never cross their genes with other nations, it could cause a finite amount of possibilities of combinations. A mutation would, in this case, help to introduce a new set of genes that would cause an enlargement of a pool of chromosomes and sequentially an enrichment in a development of a new population.

The mutation operator is crucial for the algorithm to explore new portions of the design domain and not fall into the trap of local minima. [Cazacu and Grama 2014] The graph provided below shows local and global maxima and minima. It is very important to understand that if a structure design stops at one of the local minimum or maximum, mutation is a great tool how to untie it from its limitations. The correlation between mutation is further discussed in Chapter 4.

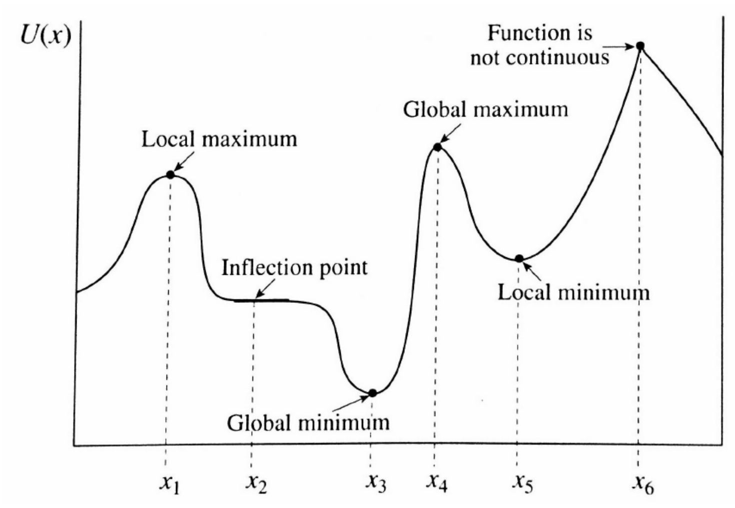

Figure 3.6: Global and local optima [Ertas and Jones 1993]

3.4 Elitism

The selection mechanism must mimic the natural selection process where a stronger individual has a greater chance of being selected. Then it is to be expected that the offspring of quality individuals will be characterized by even better qualities and thus higher fitness. However, on the other hand, the selection criterion must be chosen so that sufficient diversity is maintained in the population of selected individuals. [Hynek 2008] Keeping the best (or a few of the best) members in a pool can greatly help to improve code's efficiency. Especially in the manner of time.

Chapter 4

Results and Discussion

4.1 Genetic Algorithm setup

In all case studies, four main components could have been adjusted.

- **Population size** shows how many individuals were/should be created before the main computation process starts. It defines size of a pool where all individuals are placed and being randomly drawn for a crossover or a mutation.
- **Number of iterations** represents the amount of cycles that the whole pool will run through and since each was prescribed one crossover, and mutation after set amount of cycles, it also defines a number of crossovers and mutations in population.
- **Frequency of x coordinate mutation**
- **Frequency of y coordinate and cross-section mutation** and their influence of construction behaviour are being further discussed in next sections.

Population, mutation frequency and number of cycles were adjusted mainly based on an experiential approach. Population and number of iterations made a great difference in the precision of the calculation as well as the evolution of the fitness. The population "evolves" towards the better chromosomes by applying genetic operators modeling the genetic processes occurring in the nature—selection, recombination and mutation. [Shopova and Vaklieva-Bancheva 2006] Since mutation is also one of the most important processes that ensures diversity of a population, it is important that different scenarios are discussed.

4.1.1 Correlation between mutation and fitness evolution

The incidence of mutation is controlled by the user through the prescription of a mutation probability. [Jenkins 1991] To compare the influence of mutation on fitness evolution. Following experiment has been performed. A Tri truss structure introduced below on a Figure 4.1.

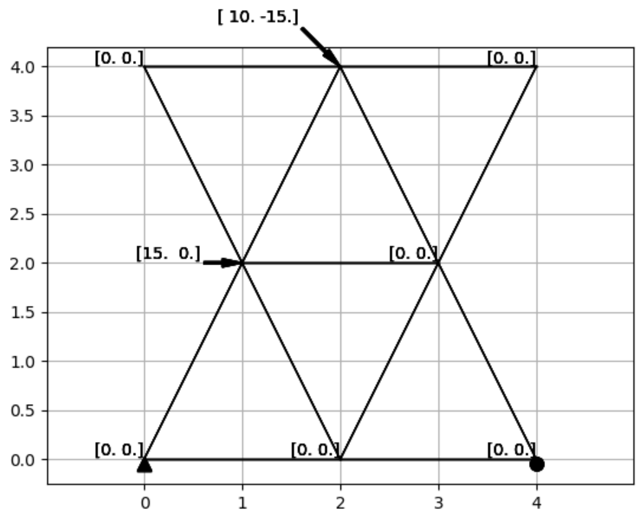

Figure 4.1: Tri truss structure

Below, Figure 4.2 and 4.3 provide a comparison between six different runs of the GA with same initial parameters preformed on a Tri truss. If a mutation was called in an iteration, only one member was mutated, unless stated differently. Mutation was performed on a structure more than once to observe which setting leads to fitter individuals.

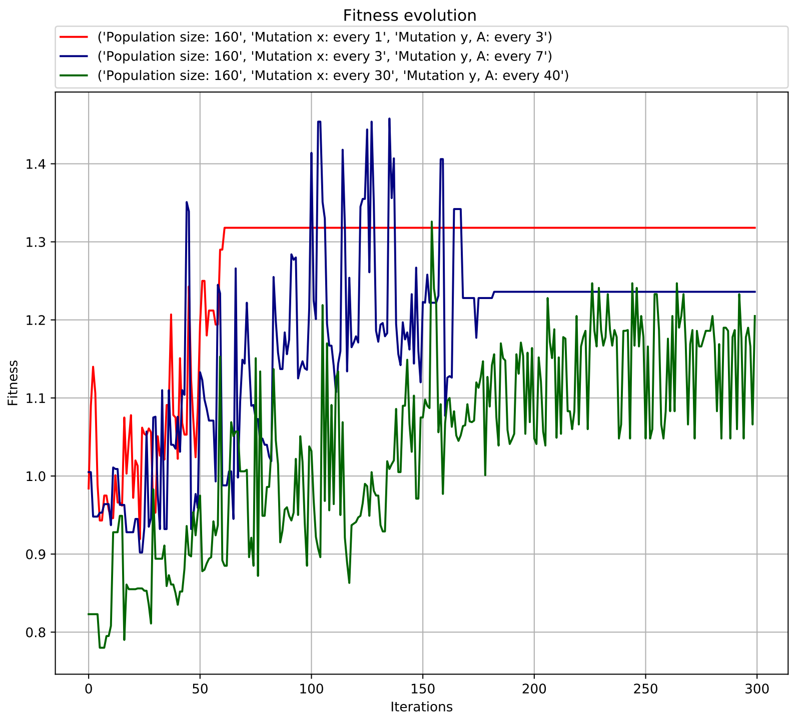

Figure 4.2: Fitness and mutation correlation, analysis performed on Tri truss 1

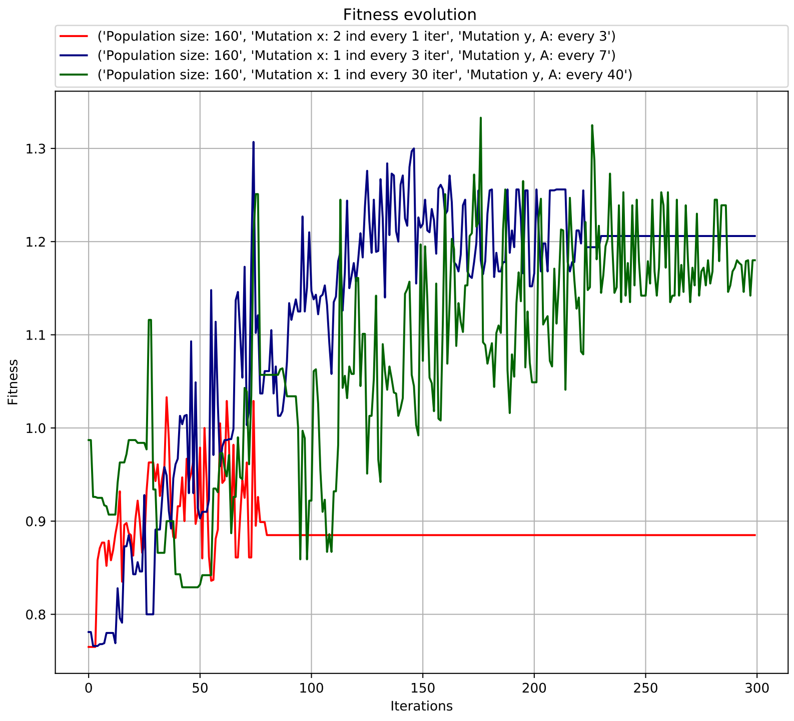

Figure 4.3: Fitness and mutation correlation, analysis performed on Tri truss 2

As we can see, in the second case (Figure 4.2), runs with less mutation were more successful. Although, on the Figure 4.3, the red population with highest mutation achieved its best result two times faster than the blue and more than four times faster than the green one. This can be explained by randomness that GA provides while applying mutation. A discussion in 4.4 Future work addresses an issue with elitism which, when solved, could lead to clearer, more coherent, results.

To analyze a slightly different type of results 2 runs with same ration of mutation to population were performed and one with doubled mutation of x-component of Tri truss. It is visible that both runs that had the same conditions got to very close results. The "red" member, which seems to have steeper incline but then rather slower further improvement is still pretty efficient.

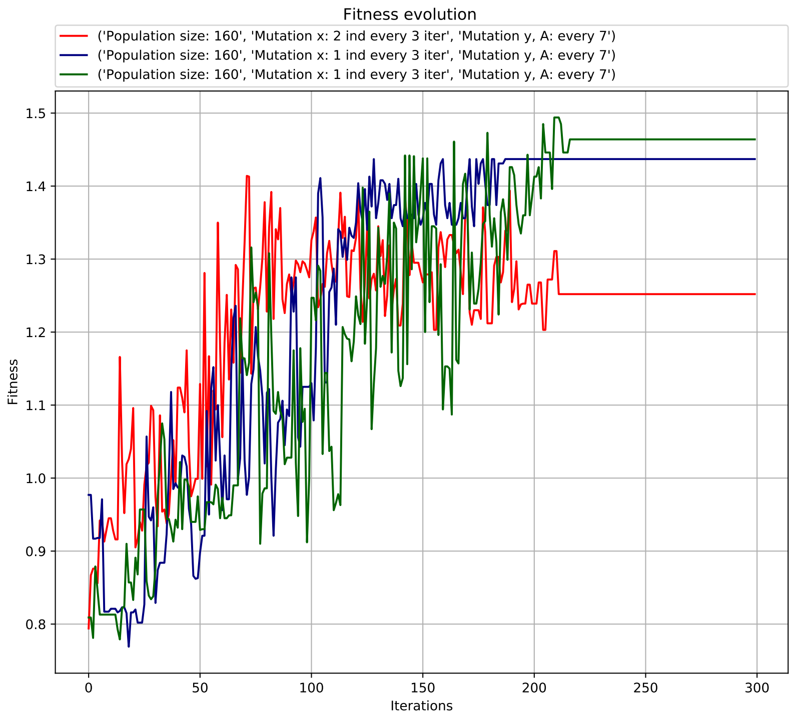

Figure 4.4: Fitness and mutation correlation, analysis performed on Tri truss 3

Overall further work can be done on analysis of mutation and fitness correlation, since current data had shown some inconsistency. Larger amount of algorithm runs should be performed to get good quantitative data for an analysis.

4.2 Case studies analysis

At first a more simple analysis was performed on a triangle to assure the code work-ability. Optimized were only top node's location in x-direction and cross-sections. Then the calculation and optimization experiment was extended to a, so called, 1x4 truss structure (name comes from the amount of sections it has) when still the same parameters were analyzed, two more nodes were optimized. And the last improvement was an enlargement to a Tri truss. This time x- and y- nodes coordinates, as well as, cross-section area were optimized.

4.2.1 Tri truss

Many analysis were performed on the Tri truss trying to find the best possible node locations for the structure. Below, you can find examples of the work. Blue are steel members in tension and red are concrete members in compression. These figures were mainly added to display a small section of the range the algorithm was going through. This is from the 20th iteration of the algorithm since towards the end of the calculation, all members with the best fitness were usually the same. The

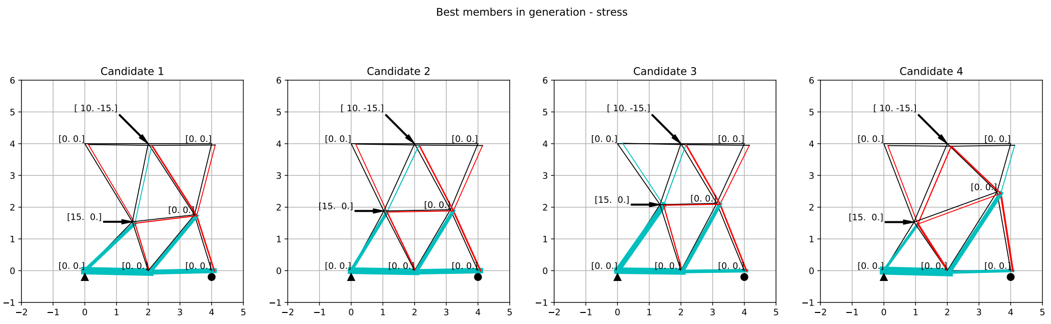

Figure 4.5: Example of node adjustments performed by the GA on Tri truss - stress distribution

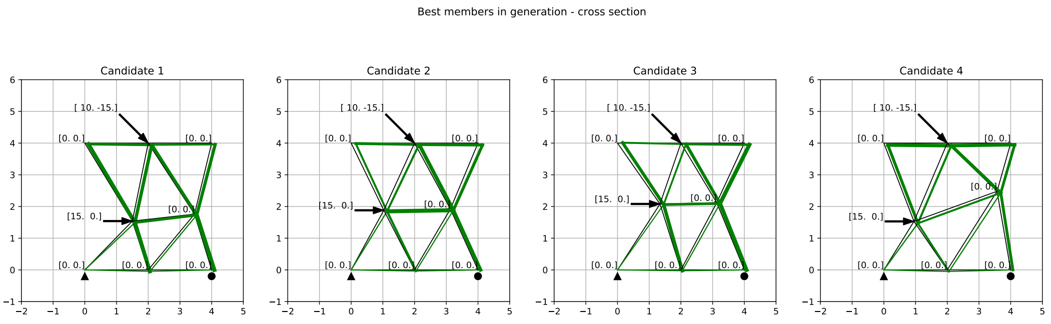

Figure 4.6: Example of node adjustments performed by the GA on Tri truss - cross-section

The best member from this current run was the following:

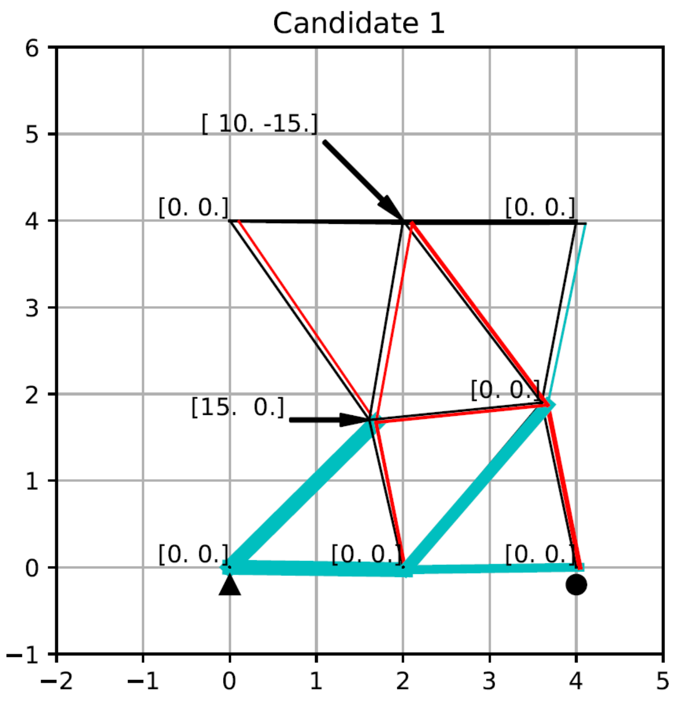

Figure 4.7: The best candidate from the population - stress distribution

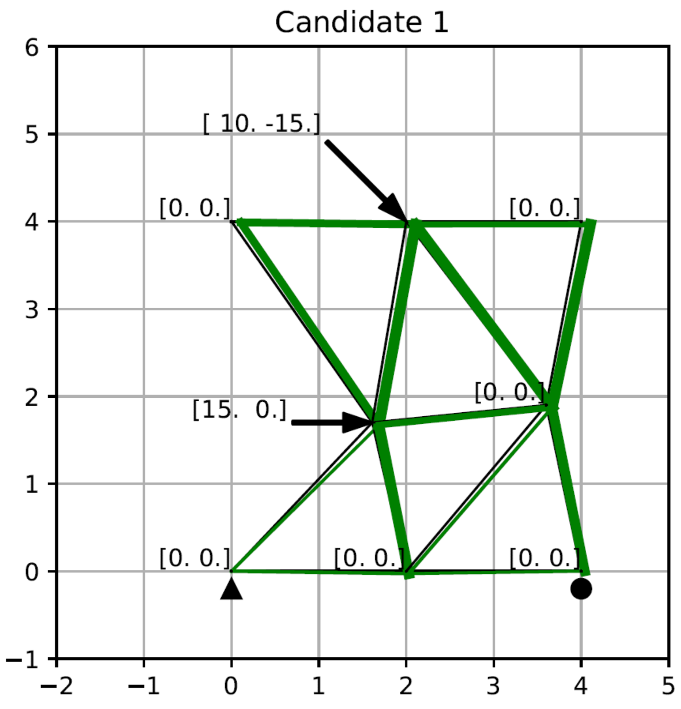

Figure 4.8: The best candidate from the population - cross-section

Overall the performance of the algorithm can be seen on four charts shown below:

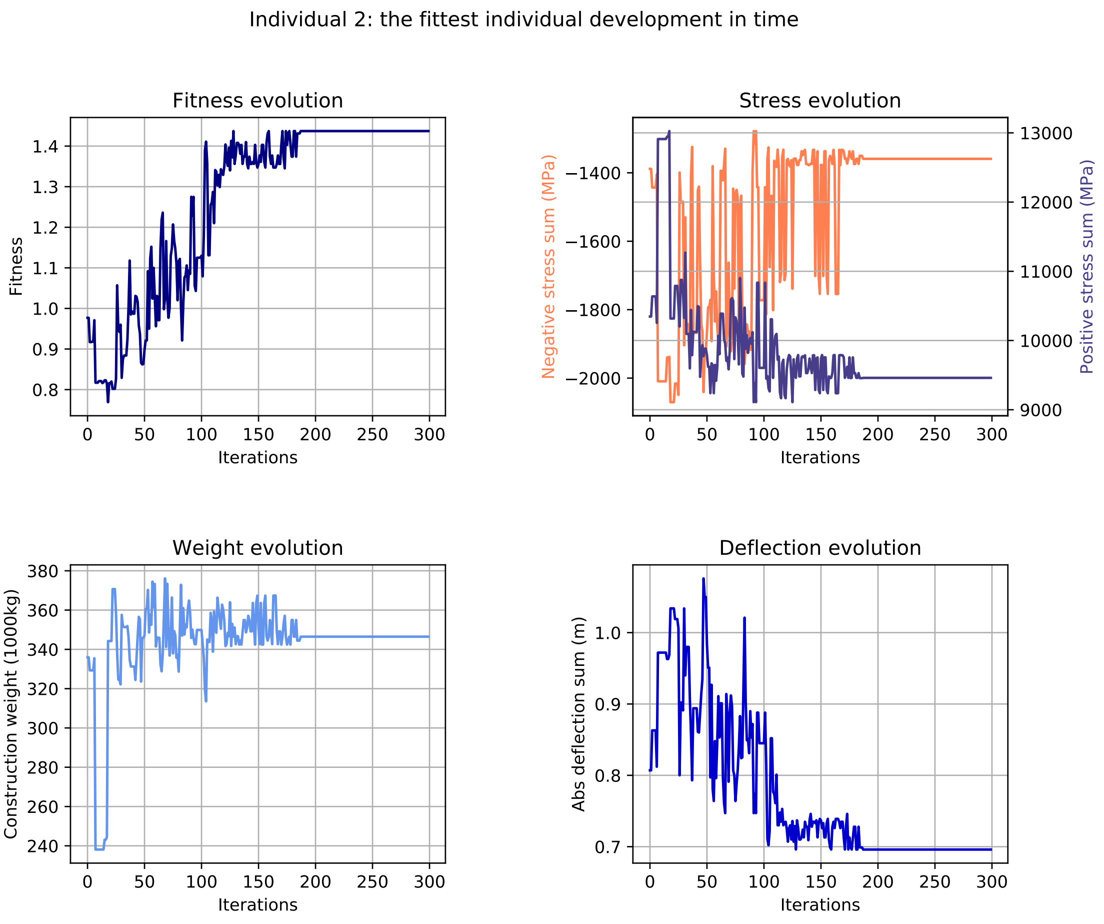

Figure 4.9: Development of the fittest individual in time

When observing the set of four graphs above on Figure 4.9, there is a clear fitness evolution visible as well as an overall deflection drop in the structure. Also stresses - positive and negative - have adjusted. On the other hand we can see that the weight of the structure has been increasing in time. This can be explained by looking into the coefficients of importance located in the fitness function. As stress and deflection were chosen to be 4 - 5x more important than the structural weight, and both stress and deflection are lower when the cross-section is larger, the algorithm has been increasing the cross-sections and hence, the weight.

4.3 Future work

Elitism implementation

Implementation of the elitism was originally a part of the work. At first I tried to implement a condition that was working only if the best individual was chosen for the crossover or a mutation.

Unfortunately it had not had a great impact on a fitness evolution. After exploring and implementing a much simpler solution when at the end of each iteration the worst member was chosen and replaced by the best one in a population, fitness function started to have steeper gradient. However, it could still be very clearly seen throughout all figures in this work that included fitness evolution, that the best member did not always stay in the population.

After a consultation I discovered that I could use a different logic. Creating a new list and from the old one always taking a few best individuals and the rest fill either with crossed-over ones or partially with completely newly generated individuals.

Greater focus on cross-section optimization

When it comes to the cross-section optimization, the correlation between stress and cross section could be thoroughly investigated and overall a greater priority can be given to it as a part of an efficient design. At this moment it also does not reflect possible shape of a cross-section and only takes it as an area that is filled with material. Changing this would require code extension and could also lead to finer results in optimization.

Fitness coefficients improving fitness function

As another aspect for greater detail work is fitness function. At first, right now it only takes three different variables. It can be considered to add more or, for example stress into negative and positive one and adding both to the fitness function calculation. That would give a possibility to adjust it separately for steel and concrete. Secondly, more time can be put into investigating what would be a better ration between either the coefficients or elements that influence the fitness function to get more precise scale of individuals.

Structural limitations observation

Currently overcoming the maximum structural capacity is assured by one condition working with modulus of elasticity. It would be interesting to investigate how would prestressed members influence the behaviour of the construction and adjust the function to new needs.

Another important aspect is that self weight of the structure is not included in the calculations right now. It was also not included in SCIA computations performed for this work. This does not correspond to the structural characteristics of the real world and for this reason it should be investigated to a greater extend.

Real world implementation

Lastly, I would like to state that this analysis and work can gain even more value if more effort is invested into a development of a real life implementation. By that I mean an investigation on how to build structures of unusual shapes and how they were built in the world already, as well as, an analysis of the connections between members.

This work could also be extended to a 3D model computing tool. In case it ever goes 3D the crossover type needs to be considered. How exactly a string splits and how to determine that point (types of crossover were briefly introduces in section 3. Analysis).

Other usage investigation

Are there alternative, more efficient methods to perform the calculation? What other problems can genetic algorithms solve? Based on these questions, more focus can be put on discovering the usage related to alternative approaches to construction. For example, while doing research on structural design, I discovered many books and articles on construction cost optimization via genetic algorithms.

Also performance of the algorithm can be addressed and further developed. Can focus on a cost optimization of a structure building (need a bit more info about it)

4.4 Closure

Among the stochastic direct search methods is the 'genetic algorithm' one based on the principles of natural selection and survival of the fittest.[Jenkins 1991] Maybe its connection to biology and nature is one of the reasons why I enjoyed so much working on this topic. But even though it is important to follow nature's patterns we shall always be aware that those systems were created for some particular reason which might not be in a correlation with our goal. By saying that I would like to emphasize that a great chunk of critical thinking should always be included while developing new engineering calculations or improving the current ones.

From project to project, day to day or even hour to hour, they tend to rebuild rather than reuse [Woodbury et al. 2010]. In this case is author contemplating about designers and their need to rebuild. From his observations of the programming community a parallel can be made related to the functioning of the world around us. And since by collaboration one can usually achieve greater results, all of the following code can be found on GitHub.

List of Figures

List of source codes

Bibliography

Ben-Ari, Monti (1996). *Understanding programming languages.* John Wiley & Sons, Inc.

Cazacu, Razvan and Lucian Grama (2014). "Steel truss optimization using genetic algorithms and FEA". In: *Procedia Technology* 12, pp. 339–346.

Champion, Edward R (1992). *Finite element analysis in manufacturing engineering.* McGraw-Hill Companies.

Ertas, Atila and Jesse C Jones (1993). *The Engineering Design Process.* Reproduced as Figure 6.3 in Ertas and Jones n.d.

— (n.d.). "The Engineering Design Process. 1993". In: *Wiley* 9, pp. 369–395.

Huebner, Kenneth H et al. (2008). *The finite element method for engineers.* John Wiley & Sons.

Hynek, Josef (2008). *Genetické algoritmy a genetické programovánı.* Grada Publishing as.

Jenkins, WM (1991). "Towards structural optimization via the genetic algorithm". In: *Computers & Structures* 40.5, pp. 1321–1327.

Lutz, Mark (2008). *Learning Python.* Sebastopol, California, USA: O'Reilly.

Miyata, T et al. (2002). "Full-scale measurement of Akashi–Kaikyo Bridge during typhoon". In: *Journal of wind engineering and industrial aerodynamics* 90.12-15, pp. 1517–1527.

Panchenko, TV (2007). "Geneticheskie algoritmy [Genetic algorithms]". In: *Astrakhan': AGU.*

Park, Kendall (2014). *Genetic-Algorithm.* `https://github.com/KendallPark/genetic-algorithm`.

Portela, Artur and Abdellatif Charafi (2012). *Finite elements using Maple: a symbolic programming approach.* Springer Science & Business Media.

Ruuska, Antti and Tarja Häkkinen (2014). "Material Efficiency of Building Construction". In: *Buildings*, p. 29.

Sanner, Michel F. (1998). "Python: A Programming Language for Software Integration and Development". In: *Journal of Molecular Graphics and Modelling.*

Sedgewick, Robert, Kevin Wayne, and Robert Dondero (2015). *Introduction to programming in Python: an interdisciplinay approach.* Ann Arbor, Michigan, USA: Pearson Education, Inc.

Sheppard, Clinton (2016). *Genetic Algorithms with Python.* Clinton Sheppard.

Shopova, Elisaveta G and Natasha G Vaklieva-Bancheva (2006). "BASIC—A genetic algorithm for engineering problems solution". In: *Computers & chemical engineering* 30.8, pp. 1293–1309.

Szmelter, Jan (1979). *Metoda elementów skończonych w statyce konstrukcji: przykłady obliczen; tablic 39.* Arkady.

Woodbury, Robert et al. (2010). *Elements of parametric design.* Taylor & Francis Group.