Structural Evolution:

A genetic algorithm method to generate structurally optimal Delaunay triangulated space frames for dynamic loads

Aikaterini (Kaiti) Papapavlou

This dissertation is submitted in partial fulfilment
of the requirements for the degree of Master of Science in
Adaptive Architecture & Computation from University College London

Bartlett School of Graduate Studies
University College of London
September 2008

I, Aikaterini (Kaiti) Papapavlou, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Signature:

Abstract

An important principle in the architectural design process is the quest for the optimum solution, a quest which is in this study structurally motivated and necessarily computationally oriented given its high complexity in nature. The present research project suggests an evolutionary algorithm that draws its power from the literal interpretation of the natural system's reproductive process at a microscopic scale with the scope of generating optimal Delaunay triangulated space frames for dynamic loads. The algorithm repositions a firm number of nodes within a space envelope, by establishing Delaunay tetrahedra and consequently creating adaptable optimised space frame topologies. The arbitrarily generated tetrahedralised structure is compared against a canonical designed one, whilst several experiments are conducted in order to investigate whether -and to what degree- the genetic algorithm method is appropriate for searching discontinuous and difficult solution spaces or not. The results of this comparison indicate that the method proposed has advantageous properties while being capable of generating an optimum structure that exceeds statically the performance of an engineered tetrahedralised space frame.

Word count: 9988

Acknowledgements

I would like to thank my supervisor:

Alasdair Turner for all his guidance, advice and encouragement

I would also like to thank:

Sean Hanna for his critical viewpoint and the inspirational discussions

Eva Friedrich for providing the script for the Delaunay triangulation

Chris Pappas for his endless support and patience

Table of contents

List of illustrations

1.0 Introduction

The research study presented here draws inspiration from the inner morphological mechanisms of nature and natural selection processes and seeks to experiment evolution on a space frame. Evolution entails alterations in the genetic makeup of a population, referred as the product of natural selection operating on the genetic variation between individuals. According to the method proposed, a tetrahedralised structure is coded as a genetic algorithm, namely as a computer-based optimization technique for modeling a system that exhibits ability for adaptation in the presence of local inner change, by encapsulating the characteristics of heredity of structural features from a parent space frame to offspring through genes. A Particle-Spring system is used as a tool for the assessment of the system during the optimization process of its structural performance.

1.1 Evolutionary Design and Evolutionary Techniques

Lately designers and researchers turned for inspiration to nature and its morphogenetic rules in order to improve the performance of their designs and introduced thus a new term for architecture. The so-called Evolutionary Architecture refers, according to John Frazer (1995), to the exploration of the form-generating processes in architecture, paralleling a wider scientific search for a theory of morphogenesis in the natural world.

Frazer threats architectural concept not anymore as the representation of shapes and forms but rather as the set of generative rules whose evolvement leads to the formation of space. The artificial structures are no longer fixed bodies but rather complex energy and material systems, the result of iterations of a long series of evolutionary processes - and such a system cannot be deduced from its components since it is more than the sum of its parts. According to D'Arcy Thompson (1961), what should be of great concern are the underlying rules and complexities of patterns that generate the end form (Thompson, 1961). In other words, by examining the rules and complexities that dominate in nature as well as the forces that compose the end form we can relate nature to design and architecture.
Evolutionary process in nature is capable of generating surprisingly original forms. This study will show how, by the means of computational power, similar innovation can be achieved. A mathematical model of morphogenesis used as a dynamic process from which

form will emerge, is capable to obtain optimised structural behavior by the repetition and interaction of selective rules. To this end, evolutionary algorithms were introduced to the problem, conceived as search algorithms that normally use an analogy with natural evolution to carry out search by evolving solutions to problems. Therefore instead of operating with one solution at a time within the search-space, these algorithms consider a large collection or population of solutions at once (Bentley, 1999).

The end result will be a mathematical model, grown from natural phenomena; the computer program will be directed to preserve populations of solutions, permit better solutions to 'have children' and let worse solutions to 'die'. The 'child solutions' will inherit their 'parents' characteristics with some small random variation, and then the better of these solutions are allowed to 'have children' themselves, while the worse ones 'die', and so forth (Bentley, 1999). By following these simple steps, massively parallel arrays of individual 'cell units' that have very simple processes in each unit do evolve over time and consequently, after a number of generations, the program produces solutions which will be considerably improved compared to their forebears.

1.2 Design Optimisation and Genetic Algorithms

The most well known computational technique based on the principles of evolution is Genetic Algorithms (GAs). They have been recently introduced in the engineering fields and in architecture as optimisation or form-generating tools, while their algorithmic processes more closely resemble natural evolution than other adaptive search algorithms. In this study, Genetic Algorithms will be used as an adaptive model for solving optimisation problems settled within the emergent space frame subjected to dynamic loads.

Genetic Algorithms were developed by John Holland in the 1960s within his effort to explain the adaptive processes of natural selection systems. In fact, this very kind of algorithm consists of a number of elements that describe the evolutionary process. To begin with, Holland (1975) proposed a population of chromosomes as the structure of genetics, representing the possible solution of the problem. Moreover, he introduced selection as a process that defines what part of the population will pass to the next generation for further evolvement. Selection is thus guided by a fitness function which evaluates every solution in the population; a score based on how well the solution fulfils the problem objectively.

Crossover and mutation are the main operators performing between two members of the chosen group defined by this very selection.

The genetic algorithm, unlike other optimisation techniques, commences its search of the design space from a number of randomly selected solutions and not from a solution based on an initial best guess. It uses two separate spaces: the search space and the solution space. The former is a space of coded solutions or genotypes to the problem, which are mapped on the phenotypes, while the latter is the space of actual solutions (Mitchell, 1996). The algorithm preserves a population of individuals, the genotypes, in a way that each one matches to a phenotype, namely the set of parameters or else the information needed for the structure to be evolved. In this study the phenotype corresponds to the Delaunay diagram, Delanda adding that in spite of the evolved form's realization at any time within the individual organisms, the population, and not the individual, is the matrix for the production of a form. This new form is also slowly synthesized within the larger reproductive community (Delanda, 2002).

What is more, evolutionary optimisation techniques, genetic algorithms being one of them, are mainly functional in cases when, although it is not hard to determine the quality of a single solution, it is nevertheless hard to examine all possible solutions separately in order to find the optimal one. In that sense, genetic algorithms are helpful because of their ability to search discontinuous and difficult fitness landscapes. If someone defines for every possible solution to the problem of interest a scalar-valued function, a function namely that entails how good a solution is compared to the others, then the continuous search commences along the fitness landscape, until a satisfying solution is found.

Genetic Algorithms have consequently the ability to evolve systems and produce solutions that the designer cannot possibly conceive, enhancing their creativity as well as solving problems whose detailed structure was not until now easy to perceive. And since the artificial forms or structures are lately of great complexity, GAs seems to be an appropriate and proficient computation method for treating problems that concern these structures by maintaining and improving their performance, recognizing changes in behaviors and loads, getting adapted into the goals set by the designer or even by adjusting past events in order to improve future performance.

1.3 The Delaunay Diagram

In Computational Geometry some of the most recognized names are those of the two Russian mathematicians: Georgy F. Voronoi (1868 – 1908) and Boris N. Delaunay (1890 - 1980), specialists in the fields of Geometry and the Number Theory acknowledging their significant contribution. 'Voronoi diagram' and its dual 'Delaunay triangulation' are important terms not only for Computational Geometry, but also for Geometric Modeling, CAD, Image Processing etc. and they became quite popular among researchers who used geometric applications. In particular, Delaunay triangulation is being exploited in numerous applications, in plane and 3D case. But why is this triangulation particularly better than others?

The advantages of the Delaunay triangulation can briefly get described by the max-min angle criterion, which requires that the diagonal of every convex quadrilateral occurring in the triangulation should be chosen in a way that replacement of the selected diagonal by another one must not increase the minimum of the six angles in the two triangles making up the quadrilateral. More specifically, the Delaunay triangulation of a planar point set maximizes the minimum angle in any triangle or else, it is all about a triangulation that satisfies the Delaunay property (Paul Chew, 1993). This property, also called the empty circle property according to Paul Chew, states that the circumcircle of a triangle formed by three points from the original point set is 'empty' if it does not contain vertices other than the three that define it (other points are permitted only on the very perimeter, not inside). An example of such a plane is shown in Figure 1.

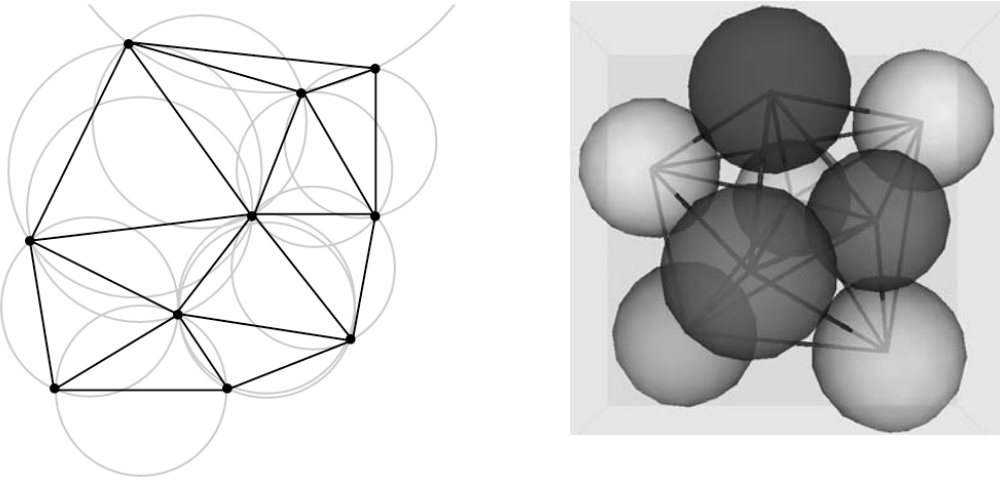


Figure 01. Examples of Delaunay Triangulations
Left: 2D Delaunay triangulation of 10 points in the plane with circumcircles shown (Source: <http://en.wikipedia.org/wiki/Delaunay_triangulation>)
Right: 3D Delaunay Triangulation of 8 points with circumspheres shown (Source: <http://roso.epfl.ch/jaf/3dwdt/triang.jpg>)

It is possible to use this kind of triangulation in tridimensional spaces by replacing the circumcircle with a circumscribed sphere. Over and above, Delaunay triangulation for a set P of points in the (d-dimensional) Euclidean space is by definition a triangulation DT (P), such that no point in P is inside the circum-hypersphere of any simplex in DT(P) (<http://en.wikipedia.org/wiki/Delaunay_triangulation>). Therefore, a Delaunay triangulation of a three-dimensional point set P is the collection of non-overlapping tetrahedra covering the convex hull of P such that each point of P is endpoint of at least one tetrahedron in DT(P). In computational geometry the three-dimensional Delaunay triangulations are the most common type of triangulations discussed and applied, since they are dense in the meaning that the min-containment sphere and the weighted average of the square of edge lengths are the smallest, while the circumspheres of the tetrahedral incident on an interior point is closest to the point. For all the aforementioned reasons, the Delaunay property will be applied in this study.

1.4 Thesis Objectives and Problem Definitions

This study aims to create a programmatic tool capable of optimizing a tetrahedralised space frame and is going to use an evolutionary algorithm for this purpose. Given a specific number of points randomly distributed in space and been triangulated by the means of the Delaunay property, it is not a straight-forward task to evaluate the structure according to its strain distribution and reposition the interconnected points in order to achieve a result with the minimum overall strain and the maximum angles between these members. There is not a regular recommended method or algorithm for accomplishing such a task taking into consideration both the geometrical (the positioning of the points) and the topological (the connectivity pattern) aspect in addition to builtability parameters.

The objective of this thesis is to evolve a complex adapting model produced by the sequence of structures taken from a set of genetic operators and eligible to different conditions throughout the design process. In this sense, a model is here meant to be more than the geometrical description of the system; it is rather the abstraction of the process, responsible for the system's emergence. The concept is materialised by the analysis of its geometrical and topological properties towards the morphogenesis of a space frame developed under the implementation of dynamic loads.

The before-mentioned proposed method of approach takes into consideration the following decisive factors:

a) The steering objective during the evolutionary process will be the development of an arbitrary space frame whose performance should be relatively good, just like the performance of an orthogonal canonical space grid.
b) Both space frames should consist of the same predefined, fixed number of particles for an accurate comparison. In other words, both of them should be tested under the same loads.
c) The distribution of the points in the evolved frame is arbitrary. A specific number of points should be fixed on the zero level, taken as ground for statically reasons.
d) The connections among the points should be established according to the Delaunay method.
e) The algorithm should take into account not only statically optimizing criteria but also the builtability factor which will affect the genetic description of the architectural concept.

1.5 Structure of the thesis

Completing a goal with the minimum of effort, either in terms of material and time, or of other expense, is of an essential profit in the engineering field. For this reason it is rather easy to understand the interest designers show in different optimization techniques and particularly in Genetic Algorithms. In the second section, a brief review of completed work will be presented, a one that summarises the positions of the present analysis. In section 3 the applied method will be thoroughly described by analyzing the proposed algorithm and reporting the results from the preliminary implementation of the experiment. The Genetic Algorithm will be tested with different variables in its genetic operators reporting differences in its performance in an attempt to reach the optimum solution. Following that, in section 4 the resulting optimized structure of the formal experiments will be compared to a canonical engineered non-optimised one and the findings of the measurements as well as the comparison of both cases will be exhibited. The final section will present and discuss an overall review of the previous investigation and some conclusions will be drawn, relied on the results.

2.0 Review of Related work

2.1 Structural Optimisation using Genetic Algorithms

The adaptive search technique known as Genetic Algorithm (GA) is not a new method for evaluating shapes or generating forms; the graphic art of Karl Sims and William Latham is quite well-known among researchers. In engineering terms, form and shape is closely related to performance in addition to the economics of the design solution. An example of a researcher who used Genetic Algorithms in engineering design is that of John Frazer (1995). In an application dealing with sailing yacht design, the GA optimization considered both objectives of engineering parameters (stability, center of buoyansy, wetted surface area, prismatic coefficient, blocking), and the designer's criteria such as aesthetic appearance, historic tradition, allusion of form, and so on (Frazer, 1996). The complication of such a search space is difficult to be described and therefore to evaluate and examine; genetic algorithms though, can surpass this complexity as been capable of searching discontinuous and difficult fitness landscapes.

As it is already mentioned the purpose of the research project conducted at the University of Ulster, was to optimize the performance of a racing yacht hull. That was achieved by using Genetic Algorithms by a combination of both natural and artificial selection. According to Frazer, a genetic code was first developed to control the fairing of the curves of the hull's profile. Afterwards, by mutating slightly the genetic code a new population was evolved. Each hull was then plotted out and several composite computations were carried out for displacement, trim, water plane, wetted surface and for block and prismatic coefficients which allowed for some indication of the likely performance of the hull. The more potentially coefficient hulls were selected through natural selection to reproduce new populations. On the other hand, artificial selection was also applied by selecting hulls for breeding on the basis of experience, institution or other considerations such as the ergonomics of the deck layout (Frazer, 1995).

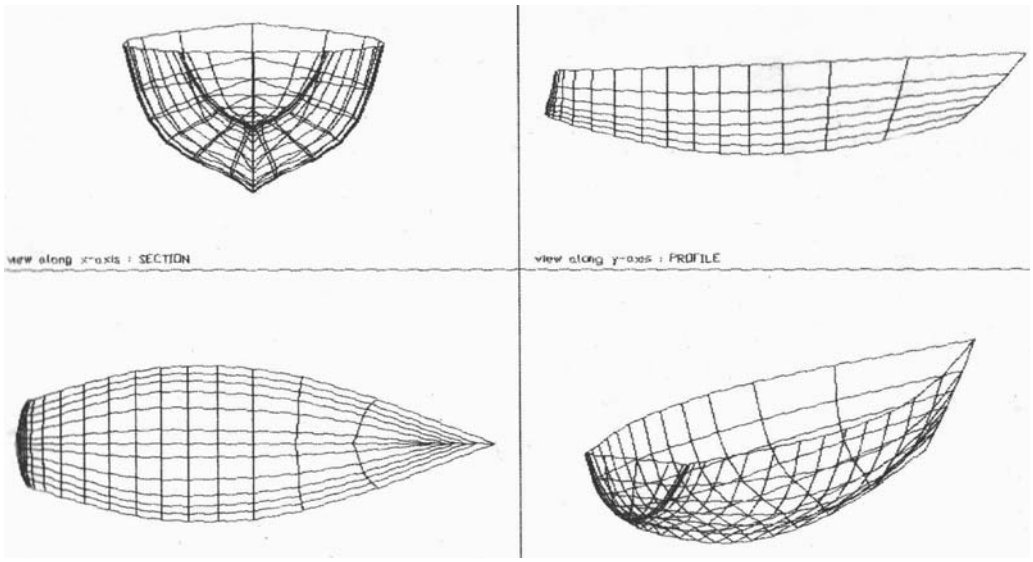


Figure 02. Development of yacht hull by genetic algorithms: John Frazer with Peter Graham, research assistant, 1993. (Source: Frazer, 1995, p.61)

Another attempt for structure optimization has also been undertaken by Peter von Buelow, who developed The Intelligent Genetic Design Tool (IGDT) and exploited many of Frazer ideas. The IGDT is more that an analysis tool, enhancing the designer to explore form-finding problems in a creative way; it guides the designer in such a way, letting him explore the design space without showing him the single "best" solution of the problem but suggests the directions that lead to the correct solution. This property lies to the basis of the structure of Genetic Algorithms which uses a population of solutions and not a single best one. GAs are very efficient at investigating the entire design space of a problem. It is possible for complex problems to have many regions of near-optimal solutions and with the aid of the genetic operators of random recombination and mutation GAs endeavor to explore a range of regions of potential solutions rather than concentrating on just one such region, adjusting a single potential solution. It is also in that way, Buelow states, that creative design is promoted and enhanced (Buelow, 2002). According to him, ideas play against each other, leading to further new ideas, as creative design is typically not a single-path, linear process, but a more complex, multipath exploration of many possible ideas.

A simple, flat cantilever truss was chosen as an example to point out the way IGDT operates given a set parameters. At the beginning the users defines the parameters for the program to start of a session. Constant criteria include material constants and properties, in this case steel, geometry constants like support positions or required load points, topology constants such as symmetry and required load cases and load combinations (Buelow, 2002).

The GA is steered by means of a fitness function which in the following example is limited for simplicity to the least weight. The Genetic Algorithm works with a given topology to find good geometry solutions. The geometry is defined by the Cartesian coordinates (real numbers) of each joint in the structure. According to Buelow, any joint coordinates can be set to appear positionally "frozen" to the GA or they can be given a pre-specified, set position in the x or y direction or both directions. In order to avoid extreme changes during crossover, the joint coordinates of a child are selected from points in a natural distribution about the parent points. In more detail, the node of the offspring's are selected from a normal distribution of random points within an elliptical range around nodes of parents. This can be seen in Figure 06 where it is graphically described the crossover process between two truss-parents. In this study a similar method to avoid abrupt topological alterations will be proposed applied in crossover as well as in mutation.

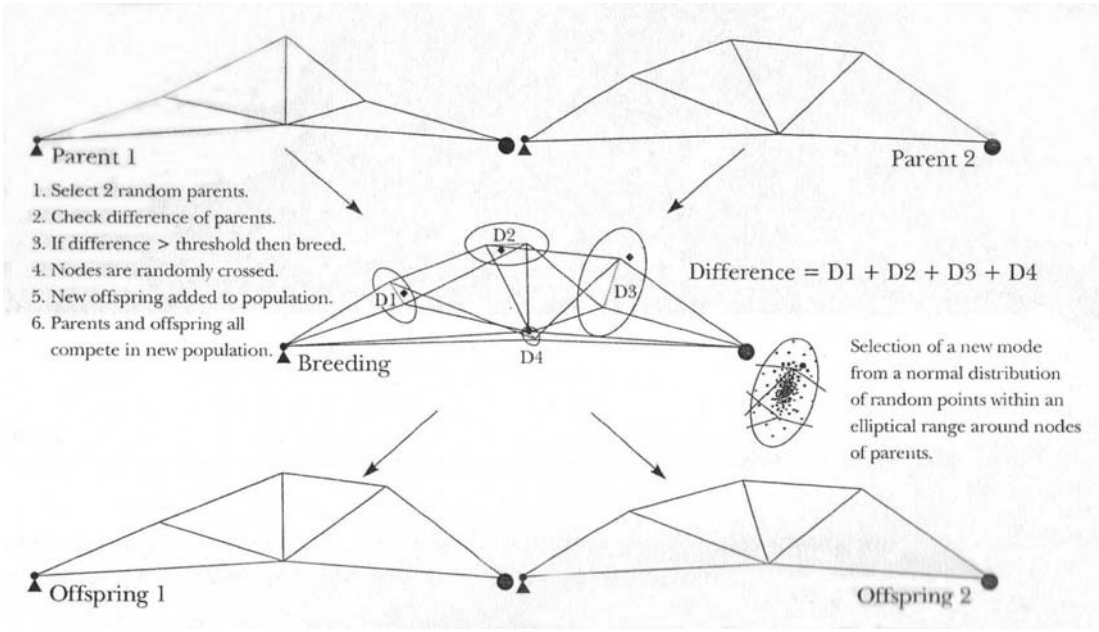


Figure 03. The breeding of two parents and the calculation of a different threshold. (Source: Buelow, 2002, p.328)

2.2 Delaunay and Voronoi Diagrams in load-bearing Structures

Because of the optimal properties of the Delaunay geometry and its geometric dual, the Voronoi diagram, both have been broadly used in structural engineering optimization as well as in structural topology optimization studies.

Pablo Miranda Carranza presented at the Generative Art International Conference in Milan, in December 2001 his study where he explored a load-bearing three-dimensional Delaunay structure. In Figure 03 is presented an example of the system with a simple "bridge" problem. Once the problem is presented, in the form of ten parallel loads and four supports, points are distributed randomly generating a random structure. After that, they are "tetrahedralised" through a Delaunay algorithm, creating the structure which will lead to an optimal structural configuration over-time. The algorithm evaluates the structure through a Finite Element Analysis and a score is given to each node according to the stresses of the bars converging in it. In order to achieve the desired, balanced condition the points with low scores will migrate to areas around points with high scores. What was observed from the resulting structure was that it was adaptively developed in the sense that it self-organized itself to form an arch and even if it was given four supports the program realized that it only needed two to support itself (Carranza, 2001). This project takes the approach to optimize the randomly distributed points that compose a Delaunay diagram with the aid of a Finite Element Analysis to reposition points accordingly to the stresses calculated. The information given to the program for the simulation is not explicitly coded but is rather an emergent property of the system.

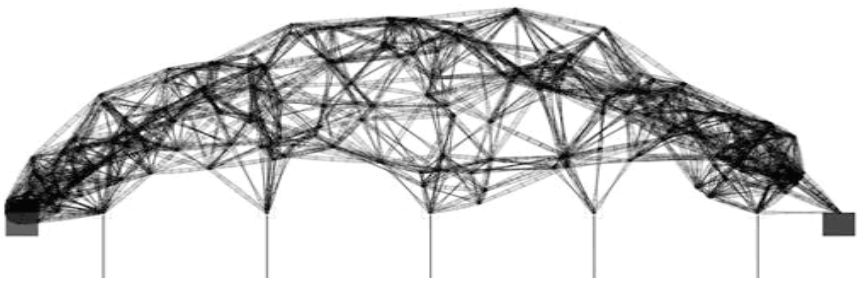


Figure 04. Self-designed Structures, a stable configuration of the 'bridge' problem. (Source: Carranza, http://www.armyofclerks.net/SelfDesign/SelfDesignedStructures.pdf)

The same conceptual approach it will be suggested in this research study where a Genetic Algorithm will perform as a reproductive and optimisation method, repositioning the random distributed points and thus letting the system to emerge. The genetic algorithm will be tested in relation to the Delaunay method, whether is capable to navigate the search space to the optimal solution. Trying to find the structure with such a geometry that comprises with the less strain in its members, the genetic algorithm will search continuously the discontinuous and difficult fitness landscapes containing all possible recombination of interlocking Delaunay tetrahedra.

Another project investigating the optimization of load-bearing structures but in this case by using the dual of the Delaunay diagram, the Voronoi diagram, as a geometry generator is the one developed by Eva Friedrich (2007). The Voronoi model here, is a static indeterminate system, evaluated statically using the structural analysis program Oasys GSA (www.oasys-software.com), considering the edges of the Voronoi polyhedral as structural members. The system intends to optimize in the same concept Miranda Carranza's structure, through methodically repositioning the points; the structural Voronoi points are moved systematically searching for a configuration which generates statically improved Voronoi polyhedral. Friedrich's structure is considered as a self-weight rigid system where the Voronoi edges are regarded as beams interconnected through rigid nodes. Several optimization methods were used to minimize the maximum displacement value of the nodes such as a gradient descent algorithm but they were proved inefficient as they were leading to local optima. The strategy that was applied after experimentation was grouping similar and neighboring points and then repositioning them in respect to each other in any possible direction, to evaluate the best combinational move (Friedrich, 2007). This method allowed the structure to evolve and optimize itself gradually by maintaining its topology where it was needed. Voronoi cells are particularly dependent on their adjacent ones; by adding or abstracting or even repositioning one can lead to a radical change of geometry. In this study the same behavior will be observed since the triangulation is dependent on the particular location of the points relative to one another and even a small change in position can cause abrupt changes to the topology of the structure.

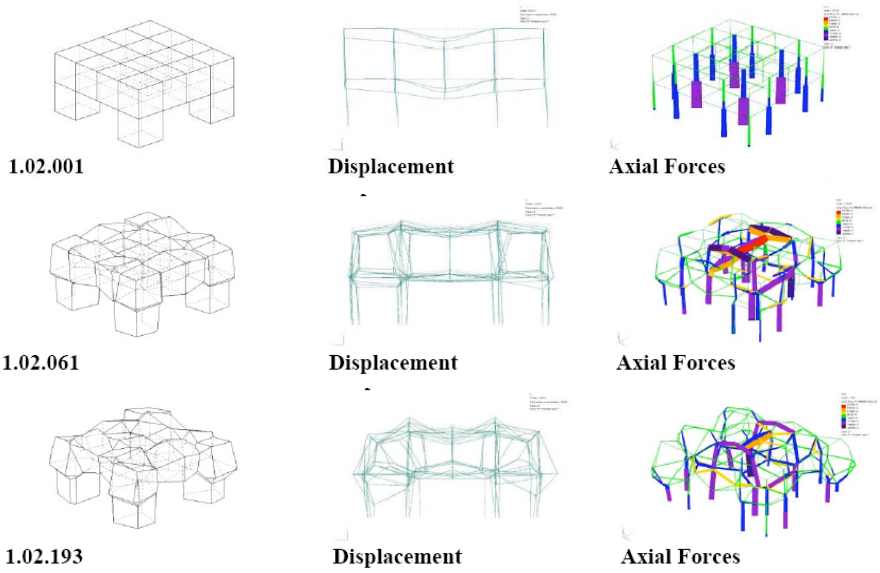


Figure 05. Stages of the structure's geometry, displacement and diagram of axial forces during its optimization process (Source: Friedrich, 2007)

2.3 Particle – Spring Systems

Particle-spring systems are based on lumped masses, called particles, which are connected by linear elastic springs. Quoting Bourke, a particle system is a collection of point masses in 3D space possibly connected together by springs and acted on by external forces (<http://ozviz.wasp.uwa.edu.au/~pbourke/modelling_rendering/particle>). To each spring it is given an axial stiffness, an initial length, and a damping coefficient. When a spring is shifted from its rest length to a new position generates a force. In addition, external forces can be applied to the particles, as in the case of gravitational acceleration.

The approach proposed in this research uses particle-spring system for structural form-finding optimization. Another example of a study that used this system for a structural form-finding was developed by Przemyslaw L. Jaworski for an MSc thesis entitled "Using simulations and artificial life algorithms to grow elements of construction" (Jaworski 2006). By distributing particles inside a specific volume and connecting them by springs in a way that they form a 3D triangulated geometry, he produced a support space frame structure. This structure grows from certain areas where "seeds" are scattered, towards an upward direction following the trails of simulated agents. By applying forces of attraction and repulsion to the particles he developed a static structure of tetrahedral compositions capable of supporting the weight of specific loads positioned upon it (Figure 08).

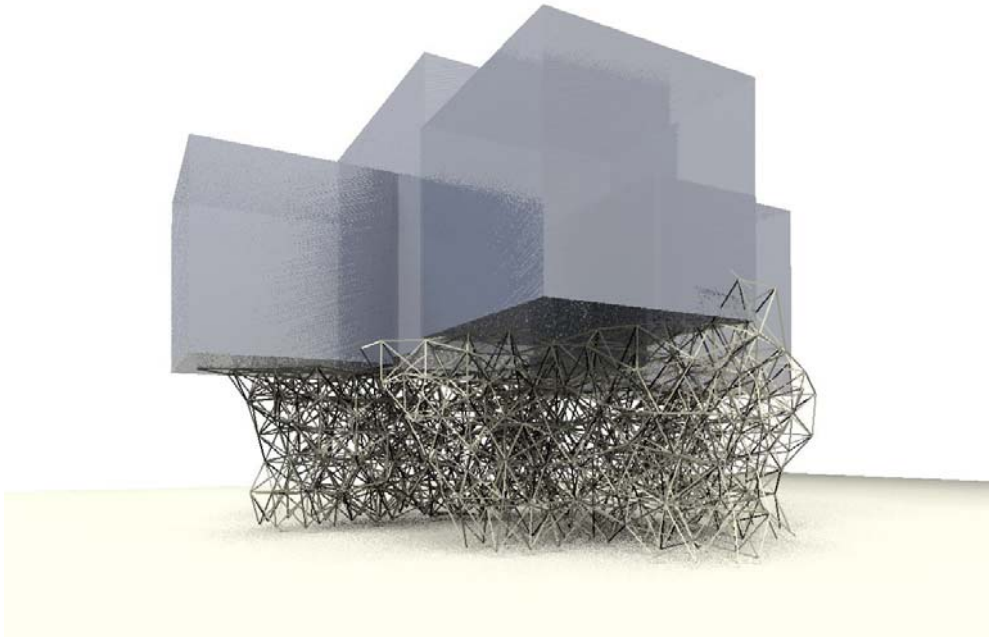


Figure 06. Support structure generated by a particle-spring system algorithm. (Source: Jaworski P. L., 2006, page 22)

3.0 Method

3.1 Overview of algorithm

The objective of this study was to develop an algorithm which would optimise a three-dimensional pin joint space frame under the application of dynamic forces by using a genetic algorithm. The algorithm itself is based on a bottom-up approach as there was no specific direction for the end topological result but the structure was rather established on the study of its local properties. The relationship of the finite point elements called particles which compose the structure characterizes the overall outcome. A piecemeal portrayal of the algorithm would incorporate a) a tetrahedra generating algorithm based on the properties of Delaunay diagram b) an algorithm that uses a Particle-Spring system to estimate statically the system and c) the setting of a genetic algorithm on it to evolve and optimise the resulting structure.

The algorithm was developed using Java (<http://www.java.com>)as well as the Processing programming language (<http://processing.org>) and was set to apply and test the performance of rules in individual parts of the space frame. It dynamically simulates a self organizing triangulated particle spring network, which evolves from generation to generation by small local alterations leading up to its global representation. The local adjustments that dynamically establish the end form are directed by a fitness function, which permits the structures with the overall smallest percentage of deflection (tension or compression) to move to the next population and feed the population with healthy solutions. To implement the genetic algorithm in this study case the Java Genetic Algorithms Package (<http://jgap.sourceforge.net>) was used and further modified to meet the objectives of this problem. The genetic operators by selecting the fittest, performing crossover and mutation between the virtual genes, mapped to particles-nodes of the structure, are seeking to optimise their relative and absolute position, establishing impermanent connections between them, aiming for a balanced distribution of weights, but always in the frame of the Delaunay diagram's properties.

3.2 Description of the Delaunay algorithm

Aforementioned, the aim of this thesis was to achieve an optimised structure composed of tetrahedra. For that reason a mesh Delaunay generating method was chosen which by

default encloses several properties such as the ability to compose a collection of non-overlapping tetrahedra avoiding 'skinny' angles. To that end, an algorithm was written which establishes a temporary initial position for the particles and later on during the evolutionary process will ensure the maintenance of the system's Delaunay properties. The first basic step of an iteration of the system is the performance of a Delaunay tetrahedralisation on a predetermined space volume filled with a predetermined number of points.

A straightforward technique of efficiently computing the Delaunay triangulation can be achieved with an incremental algorithm. This algorithm begins with four triangulated vertices (one tetrahedron) and then repetitively adds one vertex at a time re-triangulating the disturbed connections of the diagram. Each time a new point is added, the tetrahedra that contains this point splits in four parts and subsequently the algorithm is applied. In more detail the algorithm is searching through all the tetrahedra to find the one that encloses the randomly added point and after that potentially flips away every tetrahedron (Figure 09). In that way it is ensured that a sphere circumscribing any Delaunay tetrahedron does not contain any other input points in its interior and moreover each segment connecting two points is an edge of a Delaunay triangulation. Moreover taking into account the fact that the speed of execution is very important in such simulations, where a typical genetic algorithm must be iterated many times in order to produce the desirable result, the incremental algorithm is considered to be faster than a simple coded method of the definition of the Delaunay property.

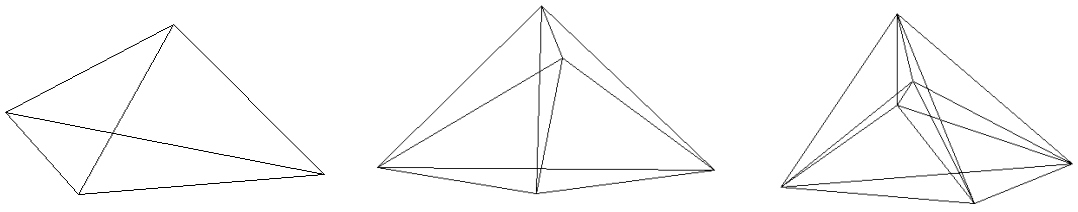


Figure 07. Establishment of a mesh Delaunay tetrahedra by adding one point at a time.

3.3 Describing the assessment of the structure by using a Particle-Spring system

Subsequently, the overall resulting tetrahedralised arbitrary structure is evaluated by calculating the deflection of each element of the frame. The evolved engineering artifact is

subjected to structural analysis which contains the set of physical laws required to study and calculate the behavior of the above-mentioned system; the reliability of this system is evaluated upon a)its ability to withstand its own weight and b)its ability to offset the effects of lateral loads. Hence the analysis of the system is the computation of the deformations and the stresses of its elements whose behavior encapsulates locally and globally the principal actions of the actual system.

The algorithm is taking the resultant mesh and subdivides it in finite one-dimensional elements (edges of the Delaunay tetrahedra), where each element is a straight, linear member with two nodes, one at each end. Physical properties are embedded in the system by using the characteristics of a Particle System where particles are attached to each node of the structure, randomly spread inside the bounding volume. Therefore each particle has a defined position and is supposed to have some mass. The structure is completed by assuming that every linear edge of the tetrahedra acts as a spring, connecting every two particles and keeping them a certain distance apart. The algorithm was based on traers.physics library engine for Processing (<http://www.cs.princeton.edu/~traer/physics>) where the inbuilt functions of the library made the manipulation of the system easier.

Firstly, the distance between the initial positions of the particles determined by the Delaunay topology is calculated and saved. Because of the downward forces acting on the system, the particles are affected by changing their position. The new resultant distance between the particles is calculated again and is subtracted from the initial one and the return indicates the deformation of each connection. If the distance between the two particles has a negative value then the spring is in tension and therefore exerts a force that takes the two connected particles away from each other. If the value of the distance is positive then the spring is in compression and therefore applies a force that brings the two connected particles closer to each other. The third case is that the resultant distance has the same value with the initial distance and consequently the spring has an ideal length and is in equilibrium (Figure 10). After each iteration of the algorithm, a new but yet temporary position for each node-particle is calculated and the node is drawn at its new position along with its new pairwise spring links that have been established. The algorithm by proceeding again through another iteration, a new topology for the structure is recalculated that acknowledges the updated positions of the particles and the spring connections. Topology therefore is not fixed but rather evolving taking into account always the Delaunay properties.

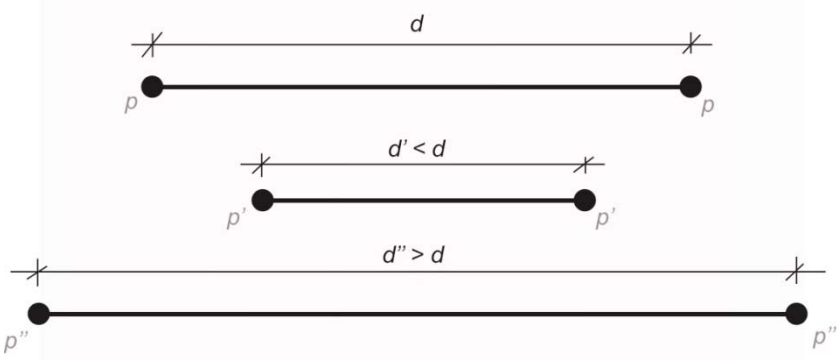
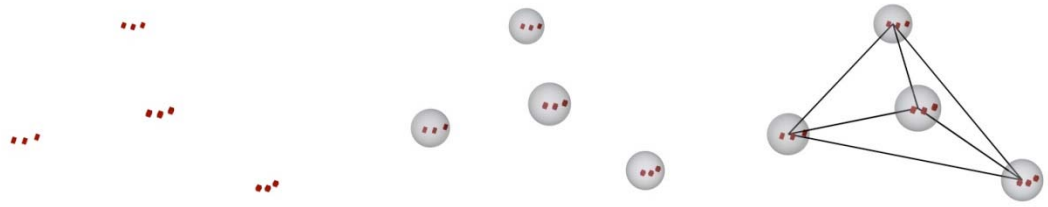


Figure 08. The three states of the member according to the spring force
(1): The member is in equilibrium
(2): The member is in compression
(3): The member is in tension

3.4 Description of the Genetic algorithm

So far the concept was described in terms of generative rules. The next step is to code it in genetic terms and describe the overall behavior of the system. In this subsection the developmental as well as the evaluation module will be explained by illustrating how the genetic algorithm evolves the structure over time and the procedure is mapped to the graphic output.

The genetic algorithm evolves by preserving a population of individuals that is composed of a genotype and a corresponding phenotype. The genotype comprises the coded version of the parameters that organise the tetrahedralised pin joint space frame. Such a parameter is referred to as a gene where in this study a gene takes the values (alleles) of a point corresponding to the position of a node of the system. Therefore, every gene represents the coordinates of a point in space, its x, y, z values in the Cartesian system, for example P_i (x_i, y_i, z_i) = $Gene_i$ (Figure 11).



(1) ***(2)*** ***(3)***

Figure 09. From genes to tetrahedral
(1): Genes are randomly initialized with double values (alleles)
(2): Points are mapped to the genes, using the double values as x,y,z coordinates.
(3): A Delaunay tetrahedron is generated from the point

After the genotype of an individual (a space frame) in the population of one hundred individuals is initialised with random alleles, the main loop of the algorithm commences with every phenotype being evaluated according to how well it completes the problem's objective given a certain fitness value. The fitness values or scores mapped on each structure decide the number of the copies of the structures that will be placed in the mating pool, an area that feeds the genetic operators with individuals and let them perform. There is several fitness proportionate selection methods (Mitchell, 1996) such as roulette wheel, sigma scaling, elitism, rank selection, steady-state selection as well as tournament selection which is the one used in this study. Melanie Mitchell (1996) comments that the aforementioned methods require one pass through the population at each generation to compute the mean fitness value and one second pass to compute the expected fitness value.

According to Mitchell the tournament selection method is computationally the more efficient and more amenable to parallel implementation since even though it shares similarities with the rank selection in terms of selection pressure, the rank scaling requires sorting the whole population by rank which is a time consuming procedure. More specifically, in the genetic algorithm used in this research project, at each generation, forty individuals-solutions are chosen randomly from the population for the tournament selection to proceed. Subsequently every two solutions among the forty chosen before are selected and an arbitrary number *n* between zero and one is picked to represent the two opponents of the tournament. The two solutions compete and the solution assigned with the biggest *n* wins with probability 0.9. A probability less than one was chosen so that variability is enhanced since it is believed that random fluctuations act as the seeds from which more efficient and organized patterns and structures can be developed (Resnick and Mitchel, 1997). Therefore it is a common approach in genetic algorithms to select not only the fittest but instead a random (or semi-random) selection with a weighting toward those that are fitter.

Following that, two individuals (two parents) are chosen from the mating pool and by the operating crossover a new solution (offspring) is generated by allocating genes from each parent's genotype to each offspring's genotype. The more simple form of crossover is a single point one where a single position is chosen randomly and the parts of two parents after the crossover position are exchanged to form two offsprings. To augment the complexity and maintain the genetic diversity between a 'parent' and its 'child' during recombination, and enhance the possibility of the 'child' to inherent genes from different parts of the parent's structure and not only by recombining genes between the top and the

bottom of the structure a uniform crossover was chosen where the genes are swapped with a fixed probability of 0.5.

Although crossover is the determinant factor for the variety in a genetic algorithm, mutation insures the population against permanent fixation (Mitchell, 1996); prevents the algorithm to reach local optima by reassuring that the chromosomes will not share too many similarities and therefore provoke the ending of the evolution. In this study the mutation operator changes the original state of a gene by a probability of 0.1 by altering the coordinates of a point randomly. A random variable for each gene is assigned which tells whether or not that gene will be modified.

With the two aforementioned operators, crossover and mutation, offsprings are reproduced and a new population will emerge permitting this process to evolve for a determined number of generations which in this research project is one thousand. It is shown that after one thousand of generations the structure does not evolve further reaching its stable optimized topology and geometry. It is essential to mention that the randomness in both operators is important for the efficiency of the genetic algorithm which is more than a parallel random search algorithm. Although the random element is ubiquitous in the entire process, the search is unquestionably directed by selection towards areas in the search space that contain better solutions and it is rare fooled by local optima, unlike many other search algorithms (Bentley, 1999).

3.5 Establishing the Fitness Function

So far, it was the breeding process that was described but not the evaluation. The genetic algorithm is guided along all its steps of its process by an objective function, the fitness function which depicts the dynamics of genotype frequencies in a population, for reproducing individuals, quantifying the prospective for survival of any individual. A fitness value for a chromosome determines its optimality in order to be ranked against all the other chromosomes in a population. The fittest chromosomes are those that are allowed to participate in the genetic process producing a new generation that will be better. The definition of the fitness function is not always a straightforward task and there are cases that are quite difficult to come up with an absolute fitness function that will lead to the optimal solution. In this research project the process was not always direct, but it was always goal oriented; to reach the point where the tetrahedralised space frame with

randomly distributed joints will perform as well as an engineered structure on an orthogonal canonical grid with the same number of joints. Towards that end a couple of different fitness functions were experimented endeavoring to achieve the best results.

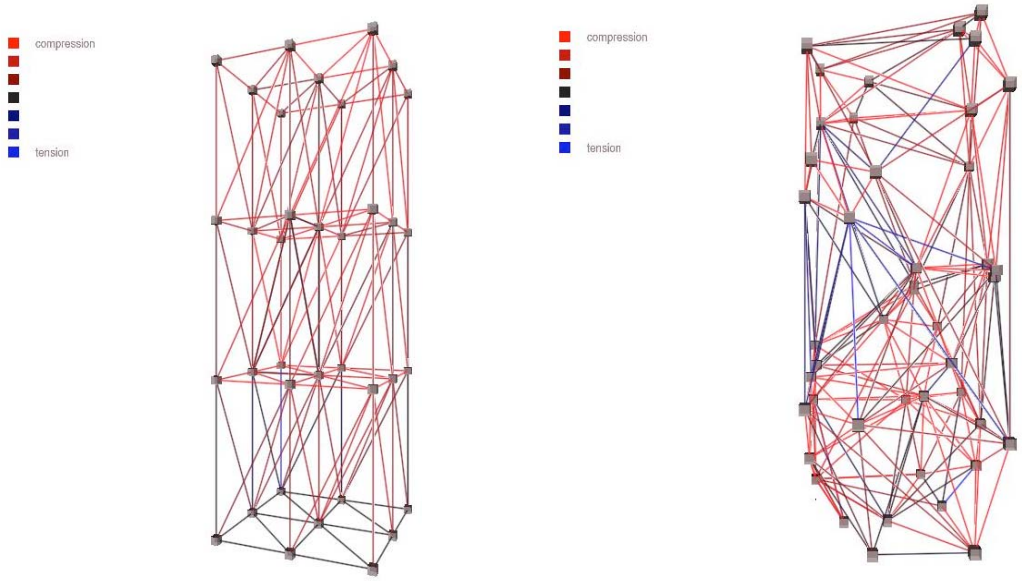


Figure 10. Indicative space frames under comparison. The edges are coloured according to the deformation; red for compression and blue for tension.
Left: A canonical engineered space frame of 45 nodes, tetrahedrilised using the Delaunay property
Right: A generated space frame of 45 nodes tetrahedrilised using the Delaunay property

As mentioned earlier, it's essential to be able to estimate how "good" in statically terms a potential space frame is relative to other potential space frames. In the algorithm the fitness function returns a positive integer number, the fitness value that reflects how optimal the solution is: the bigger or the smallest the number is according to the aspired result, the better the solution. It is worth mentioning that fitness function is defined by coded and non-coded objectives. Non-coded objectives are referred to design criteria such as aesthetics which plays an important role in engineering design. In this study for example, for aesthetically reasons, during the search for good solutions, it was decided the non-deformed connections not to be drawn and they were not taken into account during the geometrical analysis.

Aiming to optimise the space frame in terms of strain performance, the first experiments were held by assigning to the fitness value the biggest deflection calculated in the system. The deflections of all the members were calculated and the biggest absolute value was saved and used in the fitness function. It was observed that even though the structure was optimized it would not reach the scores of the engineered structure. Although the regular

one was characterized by high scores of tension or compression the overall score was less than the experimented structure since the stresses were better distributed and there were no extreme values recorded. Taking the former into account a different number was appointed to the fitness value. In this case the genetic algorithm was steered by the average of the sum of the absolute values of the deflections calculated in the system, but still the experiments delivered near optimal performance.

By comparing the geometry of the two structures, the regular one and the one under investigation it was noticed that the main distinction among them was the different range of angles between the tetrahedra. The first one, having the nodes distributed in predefined positions following the path of a regular three-dimensional grid, had uniformity as far as the initial string lengths and the scale of angles was concerned, in opposition to the arbitrarily generated structure which followed neither regularity nor symmetry topologically. Following this observation it was decided to modify the genetic algorithm from single to multiple-objective. There would be two fitness values steering the evolution; the first (fa) assigned with the average of the sum of the absolute values of the deformations of all individual springs of the system, and the second (fb) assigned with the value of the biggest angle calculated among all connections. According the estimated importance of each fitness values, weights were appointed to each of them in order to calculate the final value Fab. At the end the equation describing Fab was $Fab = weight \times fa + (1 - weight)fb$. Instrumentalising multiple-objective algorithms, running several experiments with different weights and observing the results, the final experiment was executed intending to maximise the minimum angle of all generated angles in the space frame with a weight of 0.95 assigned to fb.

3.6 Preliminary comparative testing

Having set the genetic operators as well as the fitness function, all the basic necessary rules that would contribute for the evolvement of the system were ready to be tested. At this point the system's behavior could be observed and conclusions for further modifications could be made. In Figure 13 it is shown a sample of the generated space frames. During the first generations the points are relocating constantly causing a continuous change of the structure's topology. When the fitness value is quite stable only few repositions are observed, which are also though responsible for abrupt changes to the topology, only in that

case the change is more local letting the structure to maintain its overall geometry. By comparing the results in Figure 14 it can be said that the algorithm succeeded in evolving good performances, analogous to the ones estimated at the engineered topology but not advantageous enough. The geometry produced each time with a given topology exhibited certain desired (fine distribution of points in the specified space envelope and sufficiently big angles between connections) as well as undesired (insufficiently small deflections) properties that would be beneficial for the further manipulation of the algorithm. A more detailed view of the results from the preliminary tests can be seen in Appendix II.

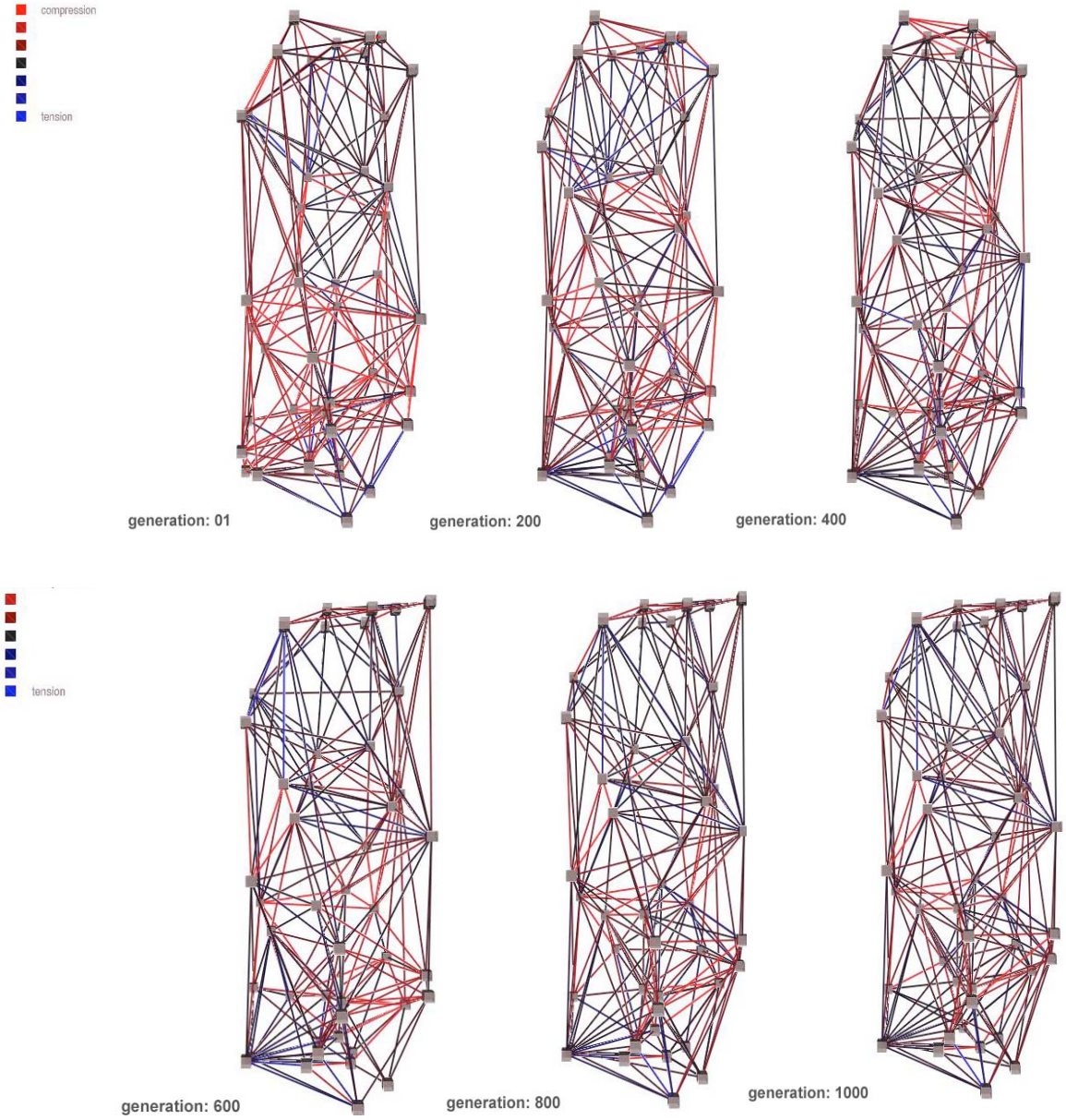
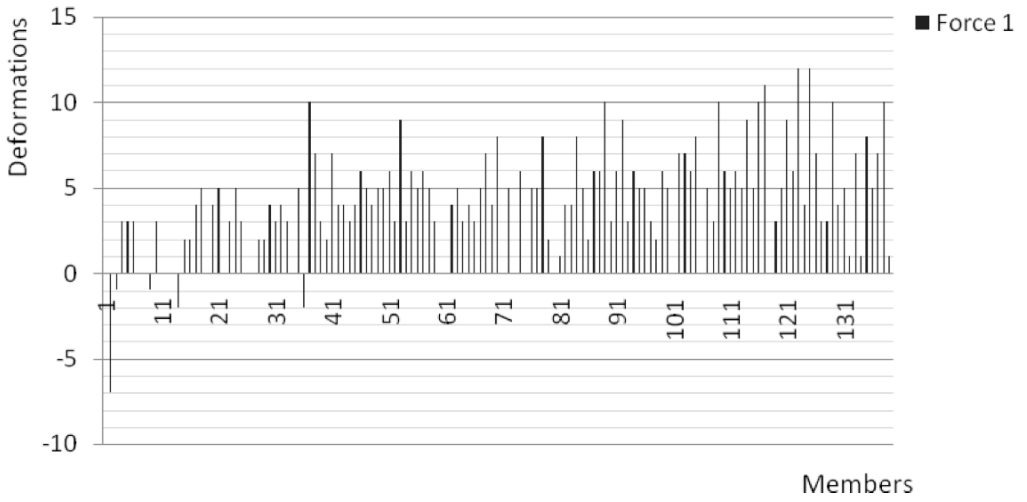
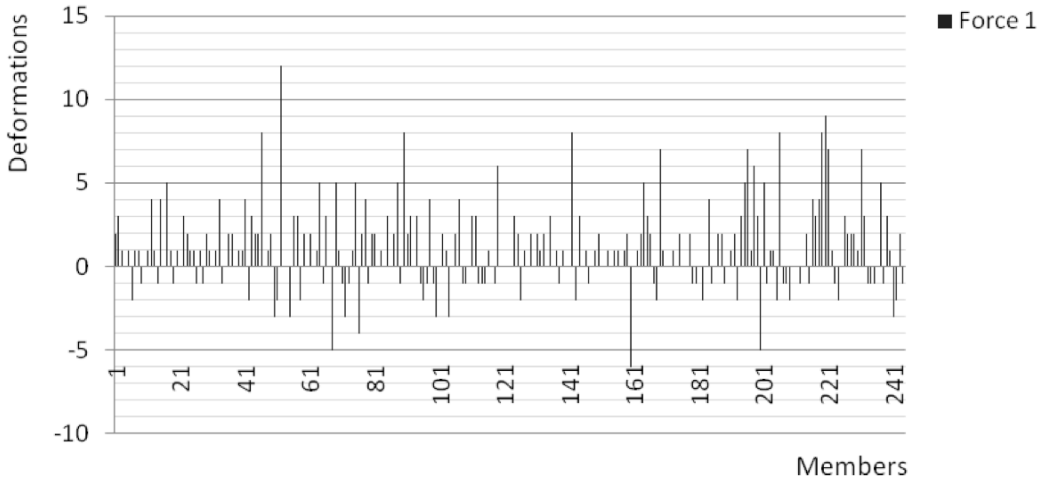


Figure 11. Indicative generated samples captured after every 200 generations. The edges are coloured according to the deformation. Red indicates compression whereas blue tension



(1)



(2)

Figure 12. Performance of the self-weight space frames
(1): Scores of the engineered space-frame
(2): Scores of the generated space-frame

4.0 Testing and Results

4.1 Adaptation of the code for the formal experiment

After running the first experiments and making the first observations of the breeding process, the script became a synthesis of all the coded ideas and was constantly modified and adapted in order to best satisfy the objectives of the research project. There were three features of the code that were further modified namely the order of the genes in the chromosome, the mutation percentage and the nature of the forces acting on the system.

In order to increase the chances of optimization a hypothesis was made that sorting the genes inside the predefined space envelop would lead to further better topologies as well as static performances of the system. By sorting the genes it is meant putting the composite genes, the nodes in order. The labeling of the composite genes is arbitrary; however by assigning them the coordinates of the nodes they can be set in order by their double values. They are sorted according to their y coordinate in descending order, from the highest to the lowest according to their distribution inside the orthogonal box. Having sorted the nodes, it means that during the crossover operation, the offsprings will have the same chance of inheriting the properties from the same areas of the structure from both the ‘mother’ and ‘father’.

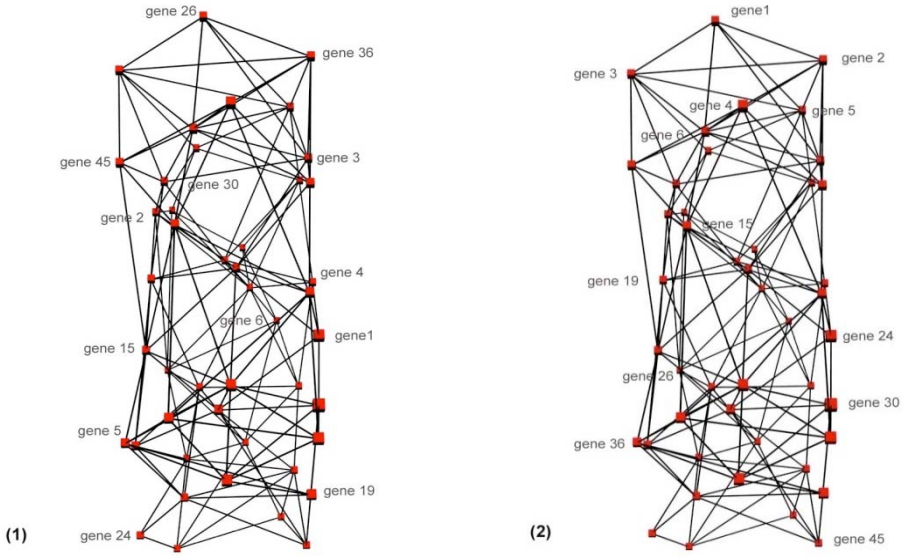


Figure 13.The order of the genes before (1) and after (2) been sorted

For further experimentation and because the coordinates of each pin joint of the system are real numbers with a meaningful special position, it was decided to include this attribute into the reproductive mechanism. The mutation was modified in such a way that during breeding, every time a node was selected to be 'disrupted' it was not allowed changing its coordinates randomly, but rather be assigned with a value distributed close to it, that is contained in a sphere drawn virtually around it as it is demonstrated in Figure 16. In that way the space of possible new values is narrowed but without reducing the innovation in the research space. Sometimes a small random change would cause better improvement than a radical change that could cause a sudden, not permanent though, instability in the structure. This observation is closely related to the nature of the Delaunay edges which are particularly dependent on their adjacent ones; by repositioning one can lead to a radical change of geometry.

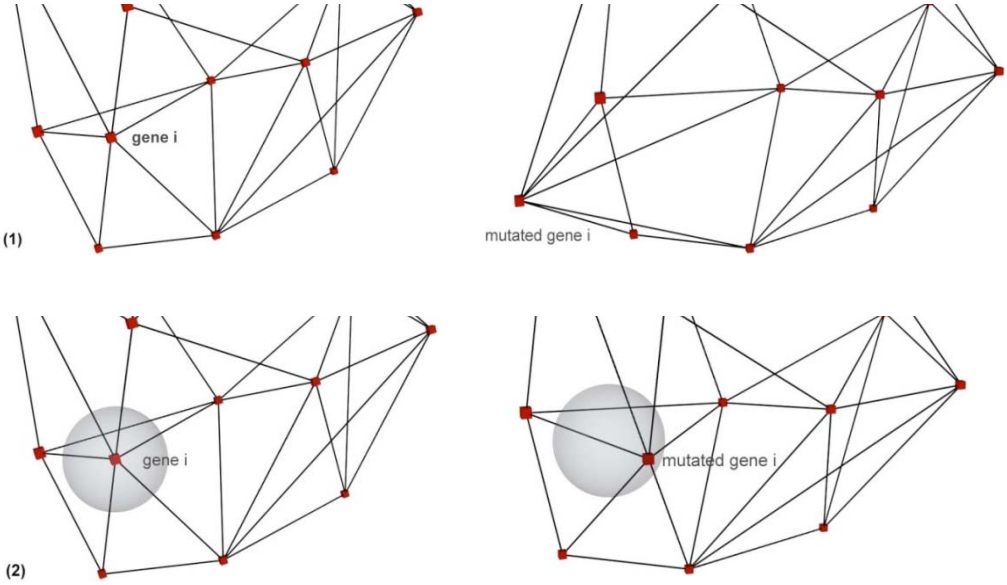


Figure 14. Mutating a gene
(1): A gene is mutated without limitation, the value is changing randomly
(2): A gene is mutated inside the virtual boundary

An aspect of the topologies that was also modified was the testing against lateral forces. It was speculated that changing the structure from self-weight to a structure under the appliance of side forces might alter the results. Following that it was decided to exert two forces of the same magnitude but with different direction as it is shown in Figure 17. The engineered space frame is very solid in terms of structural integrity as it is strengthened with diagonal bracing and therefore is difficult to defeat in terms of strain distribution. None the less, it was believed that under the shaking effect of two forces acting on both

sides the one after the other, making the structural components sway right and left could cause instability that would make the engineered structure insufficient.

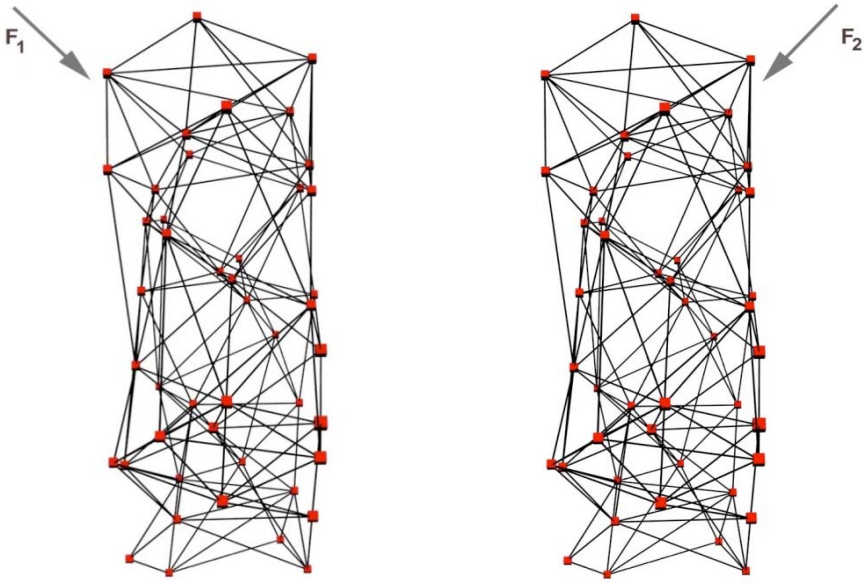


Figure 15. The two different exerted forces on the space frame

4.2 Structural Analysis of the System and Comparative Testing

4.2.1 First Experiment – Single Objective

After the algorithm's modification two formal experiments with different fitness function were executed. The first experiment that produced the test samples demonstrated in Figure 18 evolved with 45 nodes, spread in the volume of a solid with proportions 1x1x3. The samples were taken at an increment of the fitness function during evolution until 1000 generations were reached and the experiment was halted. A more complete list of illustrations of the generated space frames can be found in Appendix I. In each run of the algorithm log files were being recorded that kept track of several measurements. The results from one run create the solution that corresponds to a single member of the total population of generated solutions.

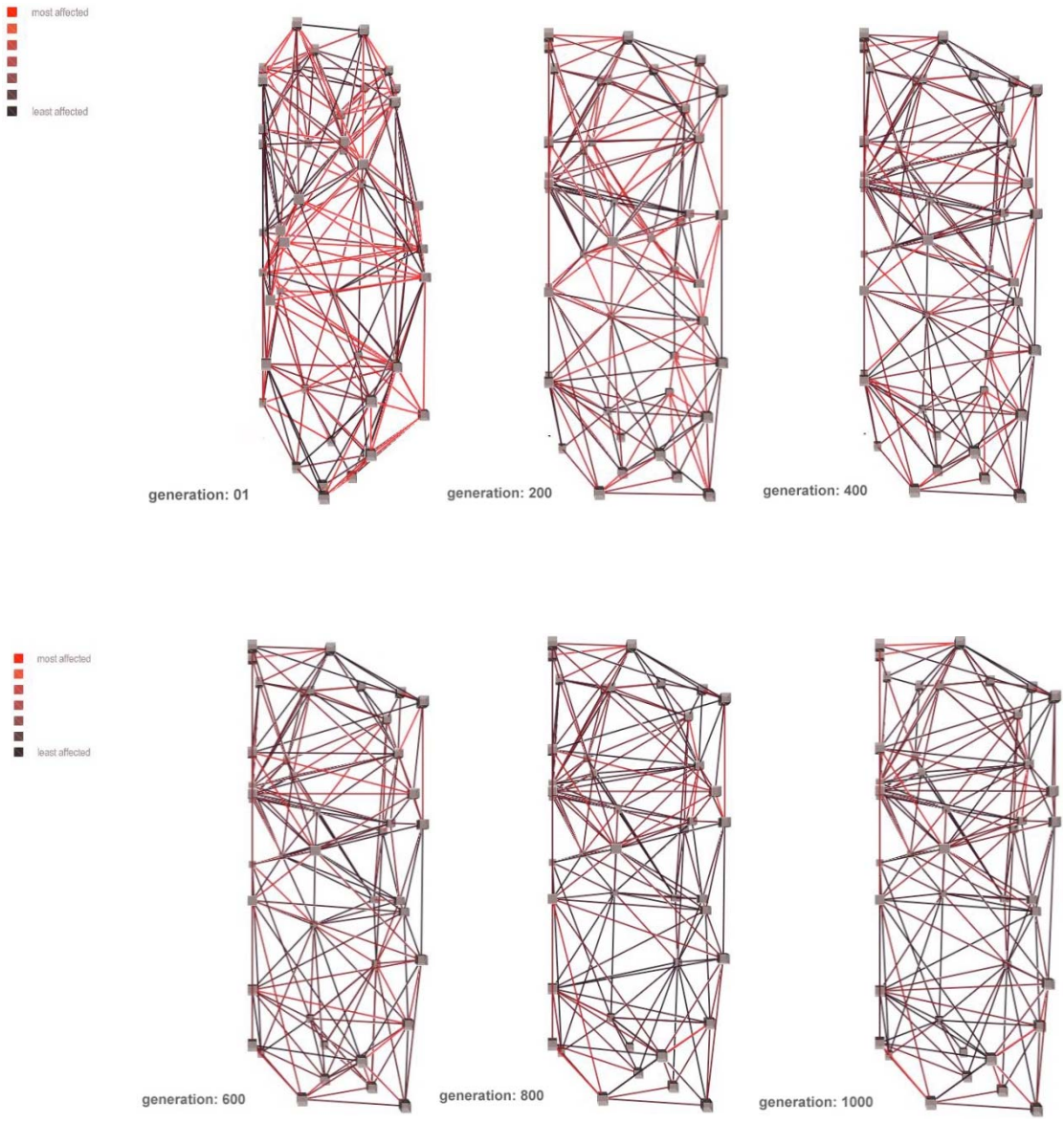


Figure 16. Indicative generated samples captured after every 200 generations during the first experiment. The edges are coloured according to the total deformation. The more red they are the more affected from the appliance of the two forces.

The fitness function of this experiment is single objective and tries to minimise the average of the sum of the absolute values of all members of the frame along time. In Figure 19 it is shown that during the evolutionary process the fitness value increases verifying the good performance of the algorithm. As it was expected at the beginning there is a fast evolution of the process, after a while it stabilises until it changes again and continues changes along time. It is worth observing that even after 1000 generations the values continue to increase meaning that the algorithm has not come to convergence and the structure keeps improving

its performance. Furthermore, since the topology of the system is fixed but not the geometry, there is a consequent adjustment of the members of the system. The quantity of connections varies, but it is an advantageous property of the optimisation process that, as it is depicted at Figure 20, at the end the space frame optimises its performance with less members.

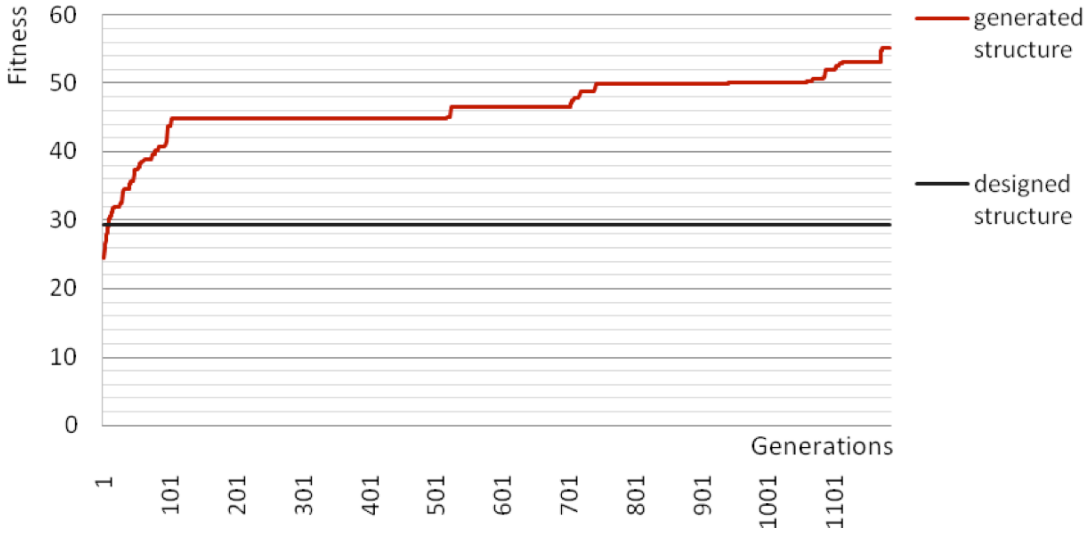


Figure 17. Fitness value activity during runtime of the algorithm for 45 nodes and two forces in comparison to the fitness of the designed structure

Figure 18. Number of members during runtime of the algorithm for 45 nodes and two forces

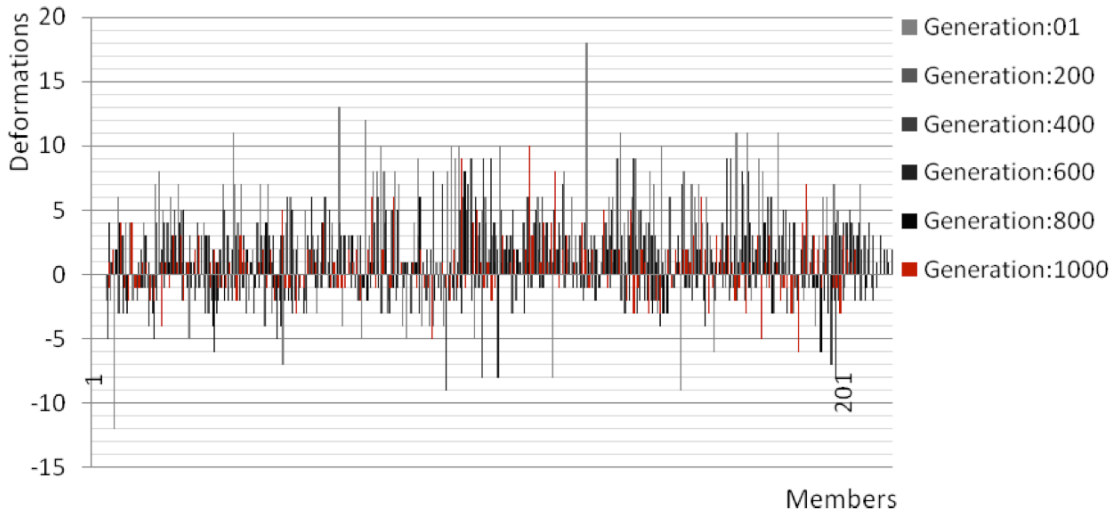


Figure 19. Deformations activity of all members during runtime of the algorithm for 45 nodes and two forces every 200 generations

Throughout the breeding process, as it was mentioned, the generated structures were assessed for stability under forces, applied proportionally to the number of nodes. The deflections of the connecting members of the joints played a significant factor for the forming of the resulting product. While the algorithm relocates the particles-nodes inside the defined space boundary, the springs react optimistically by reducing over time their deformation and therefore the total amount of strain is decreasing gradually. Figure 21 demonstrates this observation by showing the scores after the assessment of all members every 200 generations, where the scores with values below zero express the members behaving in tension whereas the scores with positive value express the members in compression. In this Figure it is clear that the individual strain of all members is optimised over time giving a more economical structure.

The members of the generated topology close to the outside boundary, in opposition to the inner members, appear to have suffered more strain, as depicted in Figure 22 on the left, where the axial forces diagram is shown. Moreover, the connections near the foundation as well as the top of the system seem to suffer much greater strain than the ones in between. In the same Figure, as well as in Figure 21, it is also shown that the overall structure relies more on compressive members to maintain structural integrity; the ratio of members behaving in compression to the one behaving in tension is bigger.

Despite the topologies generated by the genetic algorithm, another idealised engineered topology was purely geometrically designed and tested under the same forces, so that it could provide a measure of comparison. That was an orthogonal canonical space frame topology with the same analogies as the tested structure, the same number of nodes, and triangulated as well by using the Delaunay algorithm (Figure 22, right). Structurally the genetically evolved topology is more efficient than the orthogonal engineered one. This can be proved by observing the graphical representation of the strain distribution to the members of the system in both systems in Figure 23 and 24. Although the orthogonal topology is reinforced with diagonal bracing, was found to perform worse than other generated topology. Observing also Figure 19, the two different fitness values on the graph, the one of the generated and the other of the designed structure, it could be said that the generated designs improve dynamically exceeding the performance of a static one.

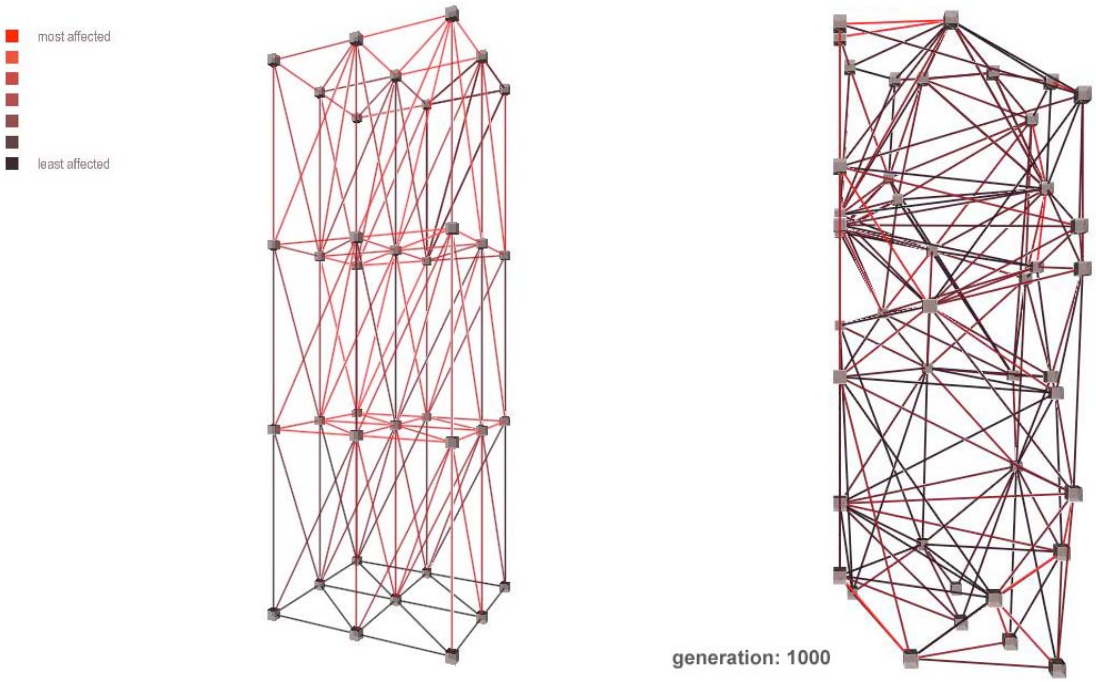


Figure 20. Samples of different geometries.
Left: Sample of the engineered topology that was used for comparing the results.
Right: Sample of a generated topology after 1000 generations using a multiple-objective evolutionary algorithm.

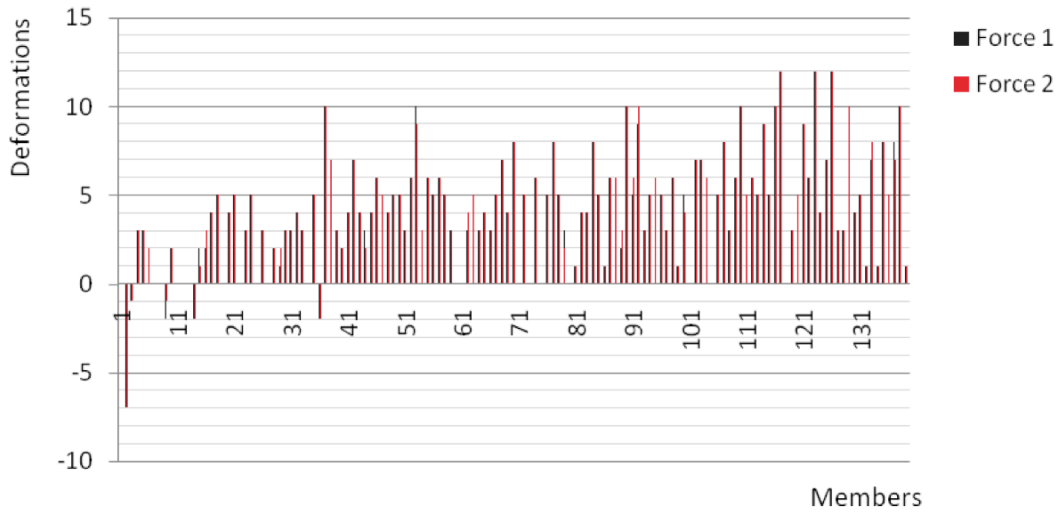


Figure 21. Strain distribution of the engineered space frame for 45 nodes and two forces

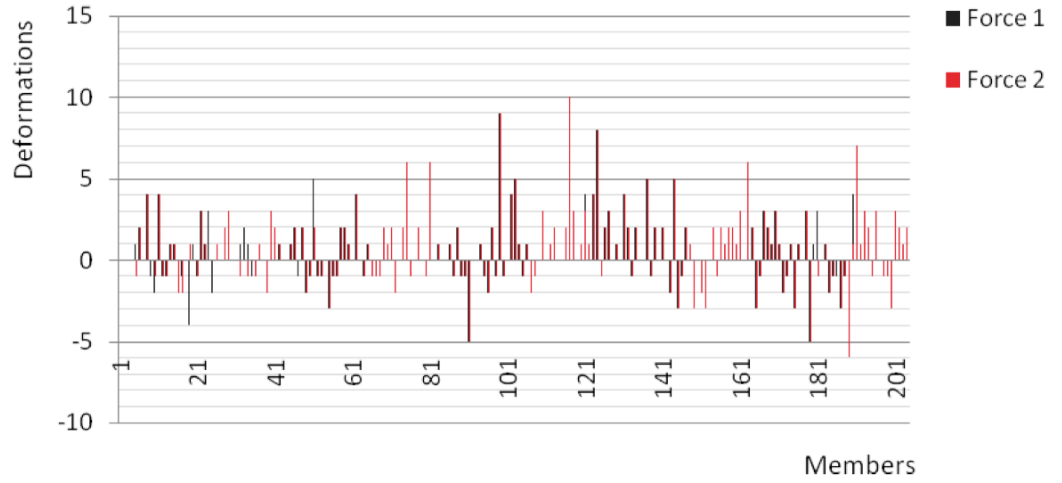


Figure 22. Strain distribution of the generated space frame for 45 nodes and two forces after 1000 generations

4.2.2 Second Experiment – Multiple Objectives

The second experiment produced the test samples demonstrated in Figure 25 using the same version of genetic algorithm with the same number of nodes (45) in the same space envelope. The samples were taken at an increment of the fitness function and for the same number of generations. A more complete list of illustrations of the generated space frames can be also found in Appendix I. The difference here was that the fitness function had two objectives; to minimise the average of the sum of the absolute values of all members as well as maximise the minimum angle calculated among all Delaunay tetrahedra with probability of 0.3. For that reason the equation $Fab = 0.3 \times fa + (1 - 0.3)fb$ was applied to determine the fitness value.

What was observed from the following Figures was that the multiple-objective optimisation gave rise to systems with the same structural capacity and performance but not better attributes considering builtability parameters. Skinny angles were again found but without leading to a disadvantageous frame in terms of strain distribution. Following closely the evolutionary process there were detected very abrupt changes of the topology. During the first experiment after a number of generations, someone could see only few repositions of the structure's nodes and when that was happening it was changing more locally the topology and therefore the geometry of the system, until it reaches a point where the changes were infrequent. During the second experiment's process, almost at every change of the fitness value the alterations in topology were efficient enough to generate a totally different geometry. The structure only after the 800^{th} generation reached a more stable topology allowing after that small local alterations that would optimise further the structure. This characteristic of the evolutionary process is closely related to the restricted properties of the Delaunay triangulation; even a small change in an angle of the system may be responsible for the generation of new tetrahedra instead of only one transformation of the respective one.

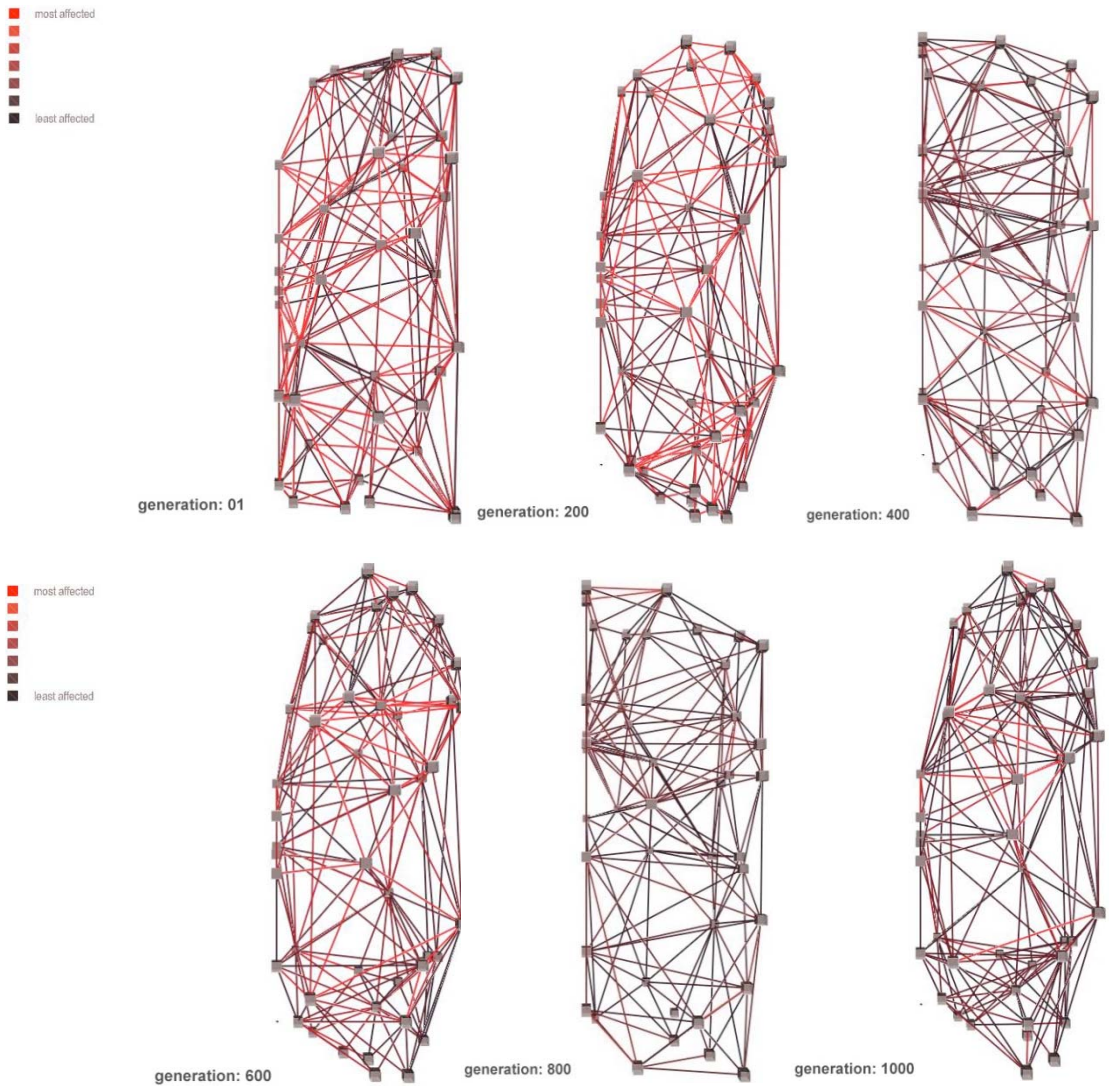


Figure 23. Indicative generated samples captured after every 200 generations during the first experiment. The edges are coloured according to the total deformation. The more red they are the more affected from the appliance of the two forces.

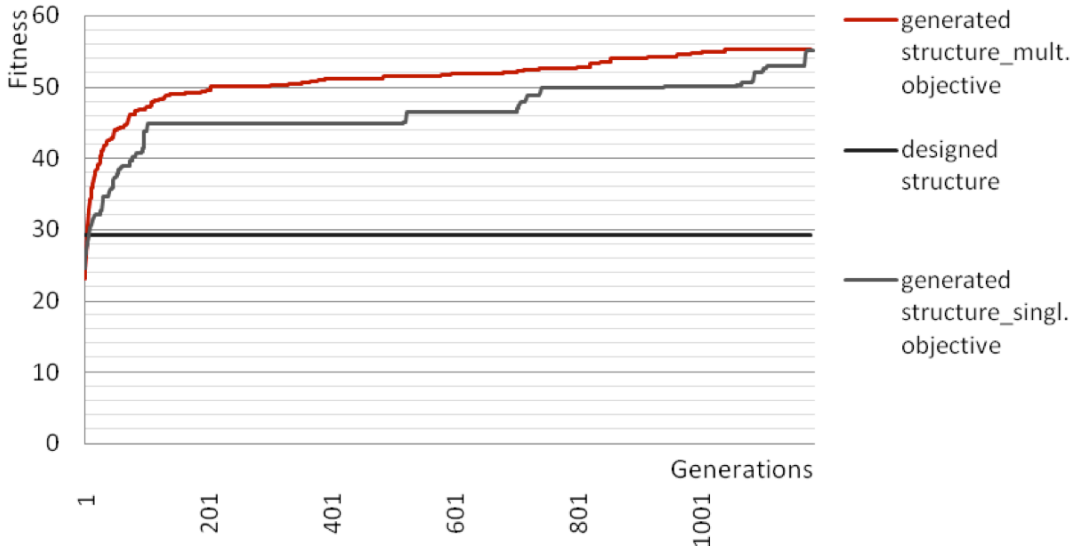


Figure 24. Fitness value activity during runtime of the algorithm for 45 nodes and two forces

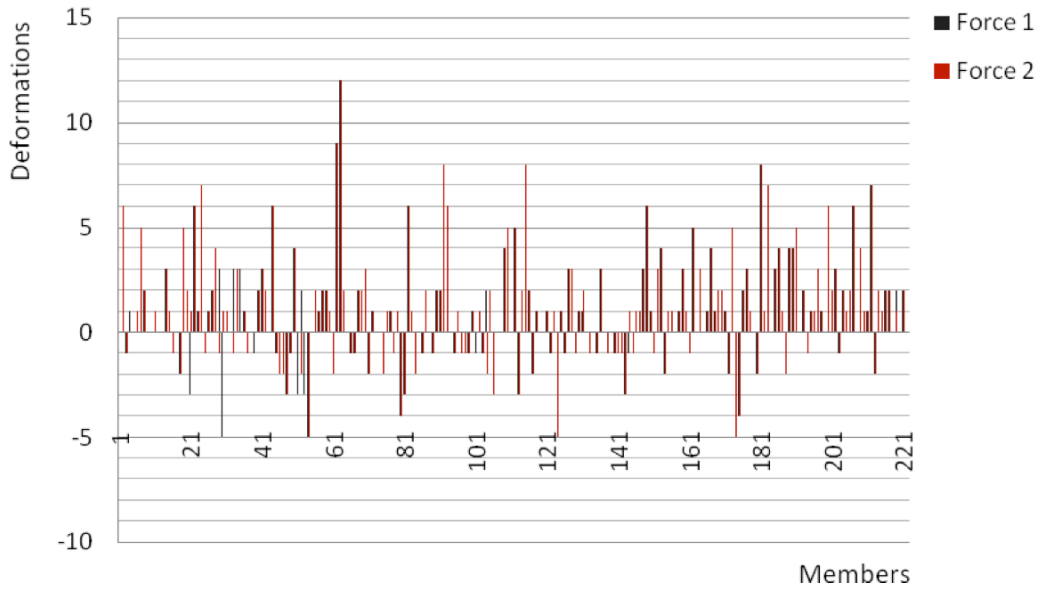


Figure 25. Strain distribution of the generated space frame for 45 nodes and two forces after 1000 generations

5.0 Discussion

5.1 Overview of findings

According to the results listed above, the algorithm produces topologies by presenting a combination of optimal attributes found also in the regular engineered topology. It has been shown that the algorithm has been capable of evolving a multiplicity of Delaunay tetrahedralised space frame structures with various topologies which encompass a certain amount of good properties better than the ones of the orthogonal space frame. This idea was based on the fact that for several measurements the values corresponding to the generated topologies turned out to be better than the engineered topologies' respective values. One of the essential findings referred to the genetic algorithm as a more advisable and yet powerful tool for statically optimising systems under the effect of multiple dynamic loads acting anti-diametrically and producing high scores of deformation.

The structural analysis confirmed that the algorithm performs optimally in terms of minimising the strain on individual strings, namely by better distributing (in a more even way, so to say) the strain on the members by constantly rearranging the position of the joints. Testing the structure against lateral forces altered significantly the results compared with those of the initial testing on self-weighted systems. In terms of average member strain, the generated space frames performed not only better than the orthogonal engineered one but also better in terms of individual member strain. When dividing the aggregate strain with the number of nodes, the generated topologies seemed to perform actually better than all other topologies. The application of side forces on the generative space frame as well as the engineered one was thus found to have a transformational effect on the topology of the structures, as it was shown by the captures taken during the process. Last but not least, the hypothesis that the application of lateral loads would cause the optimised topologies to become more economic in engineering terms than the one under comparison, seems to be confirmed when considering their structural behaviour in terms of strain displacement.

Geometrically, even though the number of nodes remained firm along the optimisation process, the number of connections varied. To begin with, the total connections were found to be more than those of the orthogonal structure, what produced a disadvantageous property, taking into account that both share the same number of nodes. Considering separately the generated geometry, the connections found at the end of the evolutionary

process were less than the ones emerging at the beginning of the process, which consists of a property evaluated as advantageous since with fewer members, a better performance was achieved in terms of structural integrity. Furthermore, the property of the system to be generated with non-overlapping tetrahedra can be considered as a very helpful attribute for builtability parameters, given that the resulting optimised space frame can break down to a locking of a different tetrahedra process. Finally, it is believed that a more advanced analysis of the topologies and geometries is required to extract more precise observations and hence those that would lead to a more efficient algorithm that would tackle with all unfavourable properties of the generated space frame.

5.2 Critical assessment

The method portrayed in this study is assessed with reference to its accomplishment by addressing the research questions and its capacity relatively to the systems it was compared with. With the hypothesis of creating a tool for optimising arbitrary tetrahedralised space frames that would improve dynamically, the approach was evaluated as an efficient bottom-up tactic. In this sense, one of the most significant elements of the proposed method could be that it effectively optimises any arbitrarily triangulated space frame. Moreover, the builtability parameter gets successfully incorporated, when employing certain variables which are qualified in attaching specific project characteristics to the code, such as the maximum possible angles between members and non overlapping tetrahedra.

The synergy of the hypothesises been taken after the preliminary testing seemed to cause the system to converge to topologies with an optimised performance, better than the engineered one, and was sufficiently and not excessively connected, a property that can be interpreted in manufacturing terms as the minimum use of material. However the genetic algorithm was significantly challenged while accomplishing the objective with a minimum of effort in engineering terms. The advantage of the Delaunay property to create non-overlapping tetrahedra facilitates the designer with a sort of geometry more easily analysed and later to be fabricated. On the other hand, the Delaunay diagram encompasses certain characteristics that are restricted during the optimised process. Although the algorithm avoids ‘skinny’ angles, the empty circle property creates limitations to the structure in a way that it then appears responsible for the non-optimal results. It is believed that the use of an algorithm that could generate more adaptable interlocking tetrahedra and thus more flexible triangulations where the edges would not have to be restrictively inside the

circumsphere could lead to a geometry with more optimal features. An algorithm that could take as an input a desirable range of angles as well as a desirable range of distances between the edges could generate a structure that could not only be more optimal in static terms but also be easier to be fabricated.

It should be noted at this point that the conclusions are based on the results after the measurements of a single run of the genetic algorithm and for 1000 generations. Due to lack of time it was not possible to run the experiments for more than once but it was also not considered necessary, as the objective was to optimise any arbitrary space frame and not to seek for the most optimised one. However, it is believed that after the execution of several experiments with different parameters, a more intensive investigation among the resulting solutions could have been carried out, leading to better forms of optimised systems. Genetic algorithms are very complex processes and being extremely difficult to predict the outcome, only through constant experimentation an absolute consciously strategy could ever get programmed.

5.3 Further development

All the observations, remarks and criticism reported before, lead to a re-consideration and evaluation of the decisions taken during the instrumentalising of the genetic algorithm. Therefore, it was judged profitable for a future simulation of the algorithm to conceive the topological and geometrical expression of the system as two individual, separate subjects of the algorithm's description. It can be conjectured that if the geometry of a tetrahedralised system was maintained but not obtained with the realisation of a Delaunay topology, better results could have become feasible with a more optimised node positioning, and therefore better structural behaviour in terms of strain displacement.

As natural systems grow under load, their inner structure being the result of an evolutionary reproduction, the structure under exploration in this thesis tries to imitate this process and evolves borrowing nature's rules. A further investigation of the system under question would regard the application of forces with different magnitude and/or direction allowing for more complex and difficult to evaluate force-deformation behaviour of the members to take place. The system shows capacity of adaptation in the presence of change and responds thus to stimuli from a more dynamic environment. Following that, a path for a more complex geometry could be designed, namely one in which the structure could be evolved,

optimised and be analysed with respect to its behaviour. Through new temporal thresholds, a more intelligent system could be materialised as well, adapting its geometry to the changing circumstances throughout the design process.

Although the generated samples were found to acquire several optimal features in static terms, the investigation on the whole did not consider any architectural extensions; there were no spatial or functional parameters included in the research's process. Consequently, an open question for future deliberation would be how this method of optimal design could get impeccably incorporated into an architectural design process of finding frameworks that could serve structural purposes of specific members on real projects of architectural dimensions.

6. Conclusions

The problem addressed at this research study concerned, given a number of nodes, the statical optimisation of a three-dimensional system composed by Delaunay tetrahedra, under the application of two anti-diametrical forces. The method chosen for implementing the problem set regards a genetic algorithm incorporating a physical dynamic simulation of a particle-spring system. Along its entire process, this project aimed to uncover whether the suggested method was appropriate or not and tried to fulfil the author's objectives. By conducting the initial experiments, the capacity of the algorithm was proved successful in generating optimised space frames and consequently the basis for the formal experiments was set. Observations from the preliminary tests in addition to several hypotheses were responsible for synthesising the final parameters as well as for the ultimate adaptation of the algorithm. Subsequent to that, the generated topology was further analysed mainly in structural terms but also in topological and geometrical ones. During its structural analysis, this very topology was compared to an engineered structure sharing the same characteristics and bearing the same load, while the results were documented. The estimation of those results verified that the algorithm is capable of optimising space frames with beneficial properties, overcoming the statical performance of the engineered one. When describing the restrictions of the recommended method, various approaches to overcome them were discussed and different ways for further exploration were referenced.

This investigation reinforces the argument that engineering and architectural design can benefit when drawing inspiration from the innate morphological mechanisms of nature and natural selection processes. Selecting the fittest, recombining, mutating solutions of a problem will lead to the optimum one. By breaking down the process to the end result in small steps expressed in a genetic language, namely in simple generative rules that interact with each other, this algorithmic method actually produces the direction for shape evolution. This bottom-up approach results to an intelligent system; a system that is no more a fixed body, but rather a complex energy and material system impossible to get separated from its components; a system which is not conceptualisable in a conventional way. Other than that, it could be further claimed that designs generated in a computational and dynamical environment can exceed in performance static figures. Not only the proposed method facilitates the designer to find a solution that fulfils the performance standards and provides good economy, but it also intends to help him exploring structures that are difficult to assess and less likely to get optimised. In this context, it might well be the case that the suggested method is the suitable one to form a mechanism for the research

of composite space frames and structure relationships under the appliance of complex dynamic forces.

Appendix I

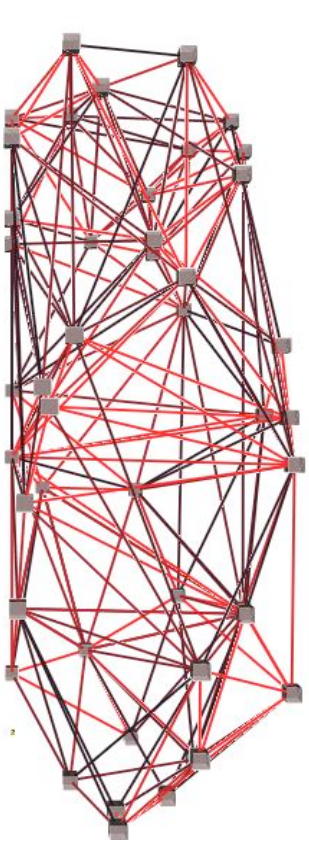
Illustrations of generated Delaunay space frames

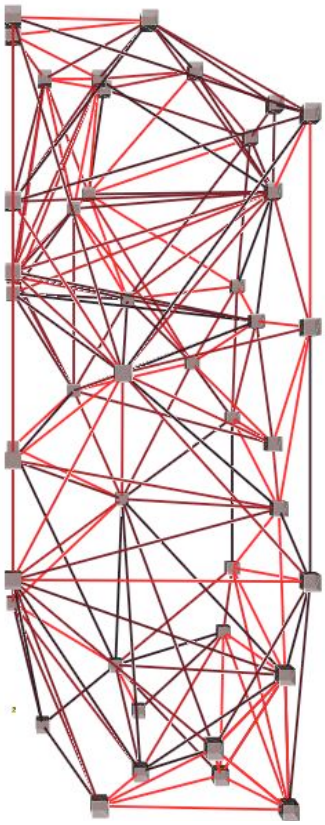
Detailed set of illustrations of the first formal experiment (pages 49-51)
(Frames were captured every 80 generations)

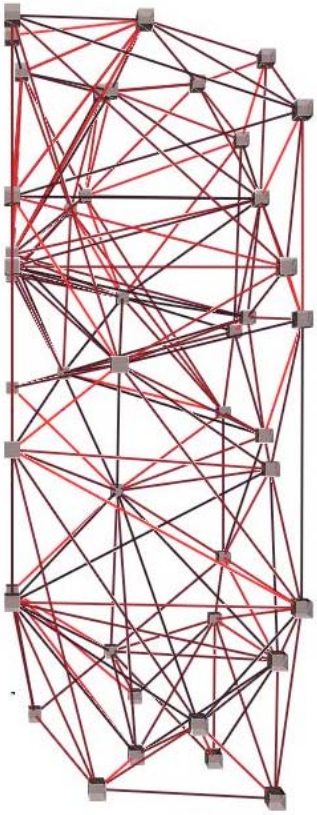
This list includes the generated Delaunay space frames from one run of the genetic algorithm when according to the fitness function it was minimising the average of the sum of the absolute values of all strains. The more red a spring is the more affected is from the application of the loads.

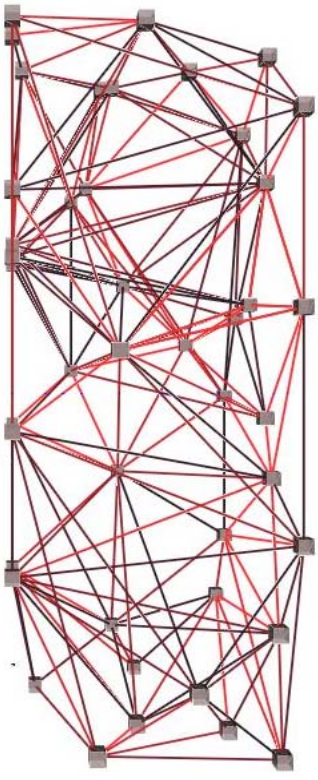
Detailed set of illustrations of the second formal experiment (pages 52-55)
(Frames were captured every 80 generations)

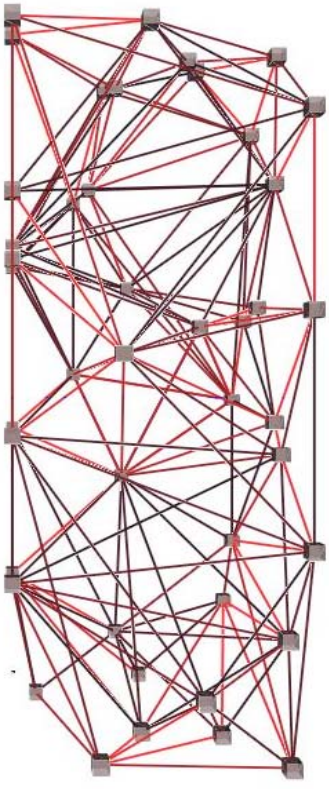
This list includes the generated Delaunay space frames from one run of the genetic algorithm when according to the fitness function it was maximising the minimum angle of the generated tetrahedra with a 30 percentage weight and at the same time minimising the average of the sum of the absolute values of all strains. The more red a spring is the more affected is from the application of the loads.

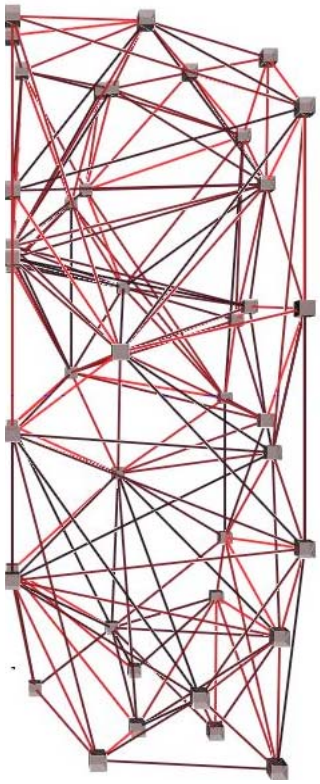


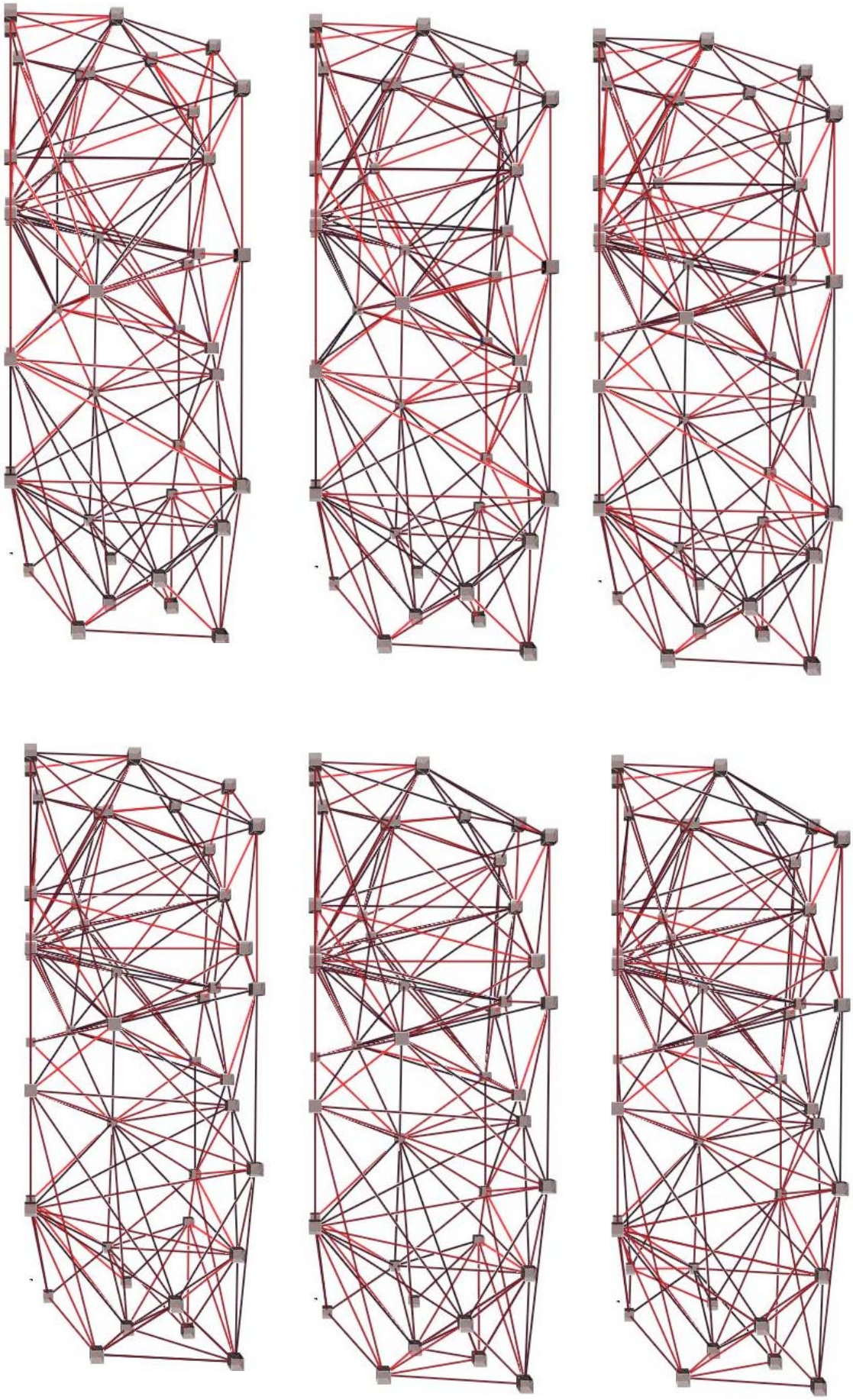


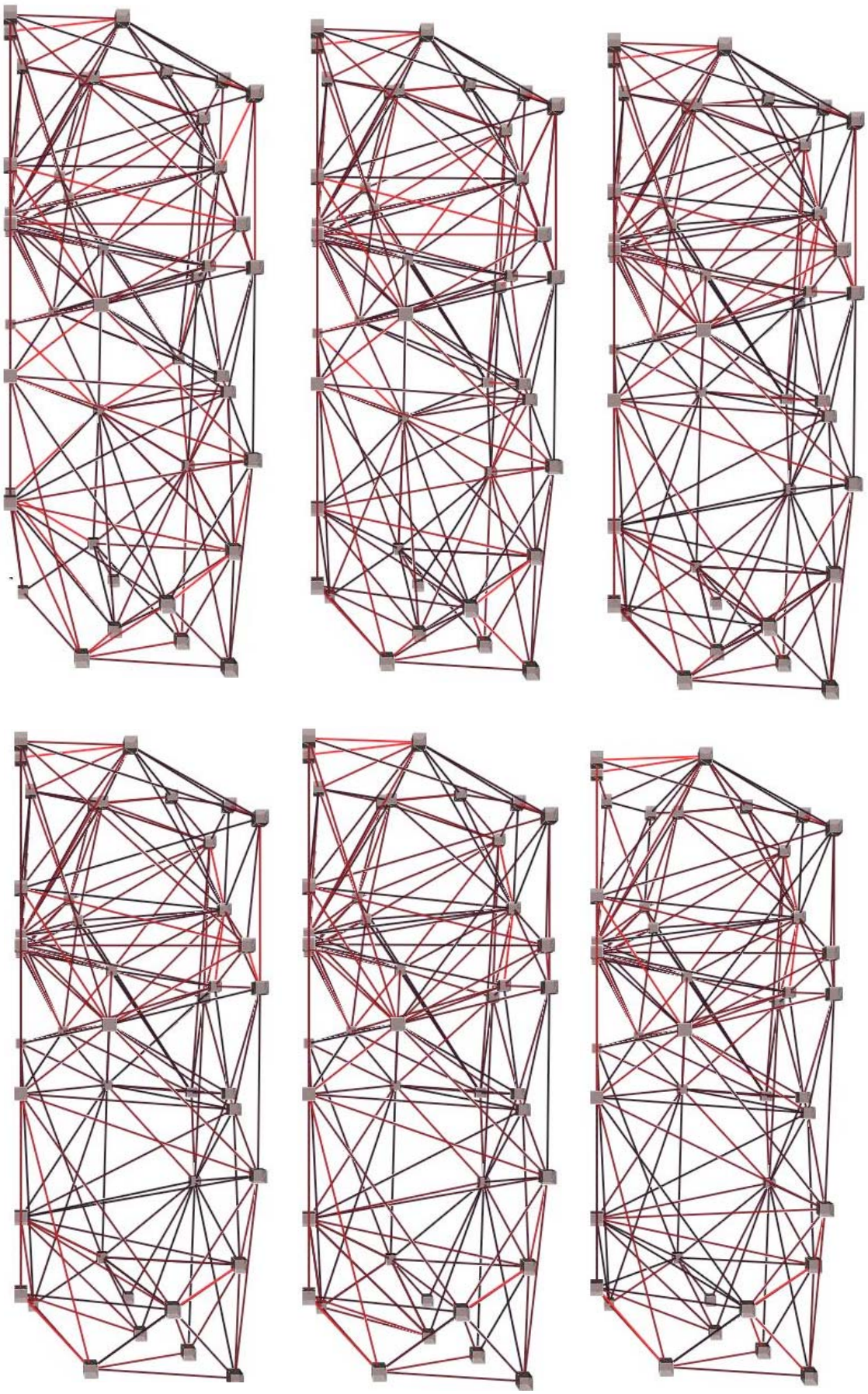






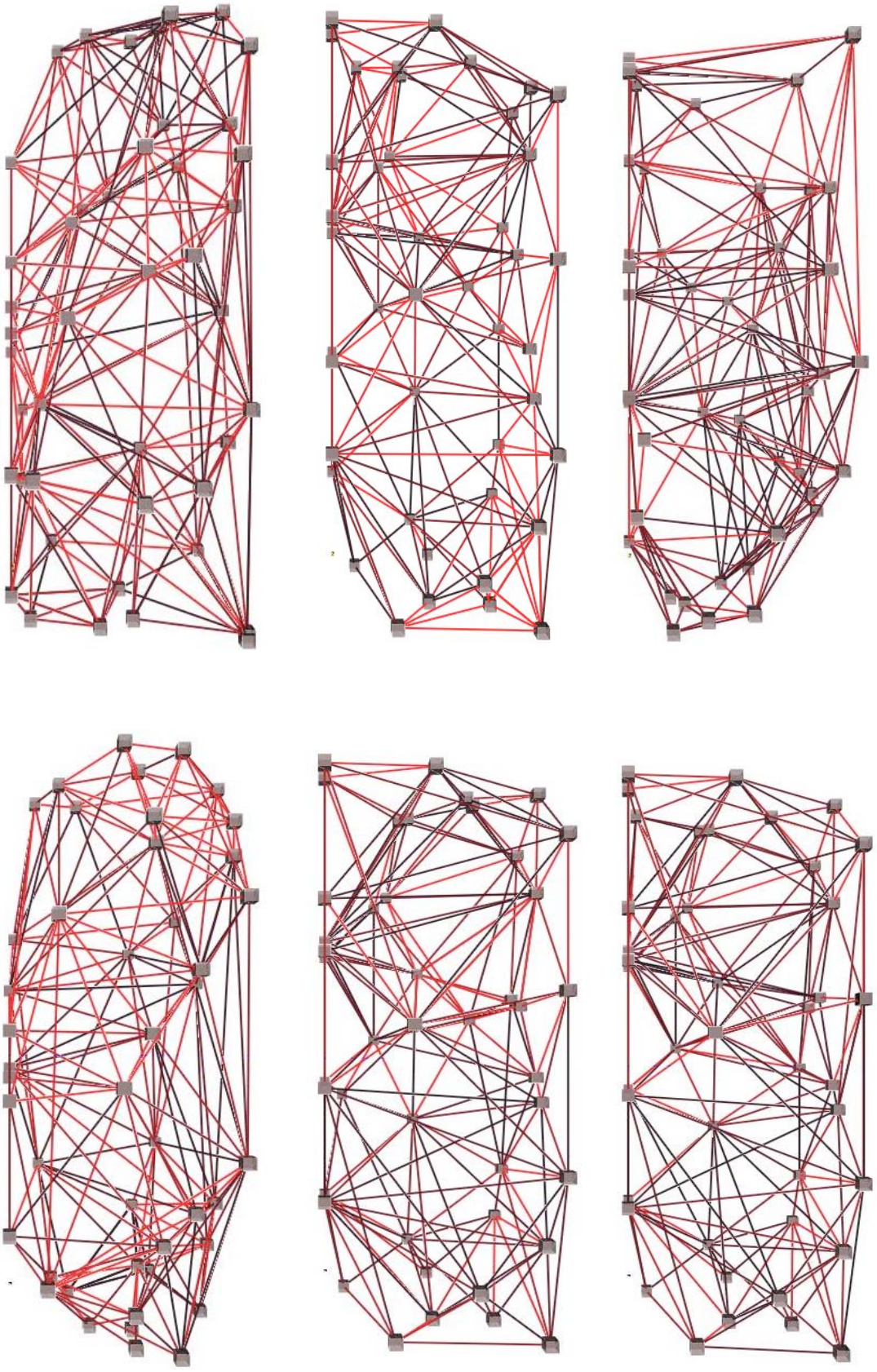


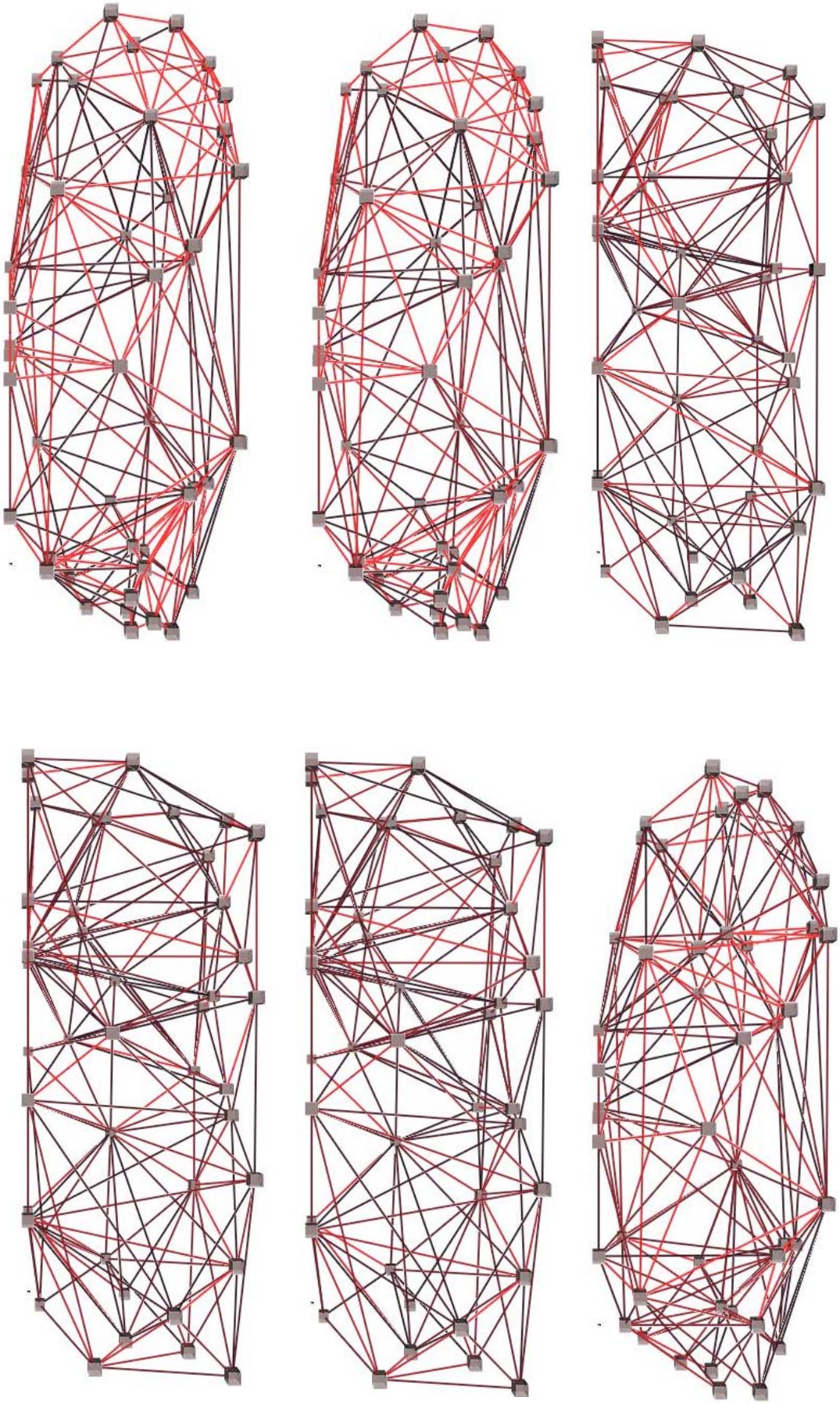


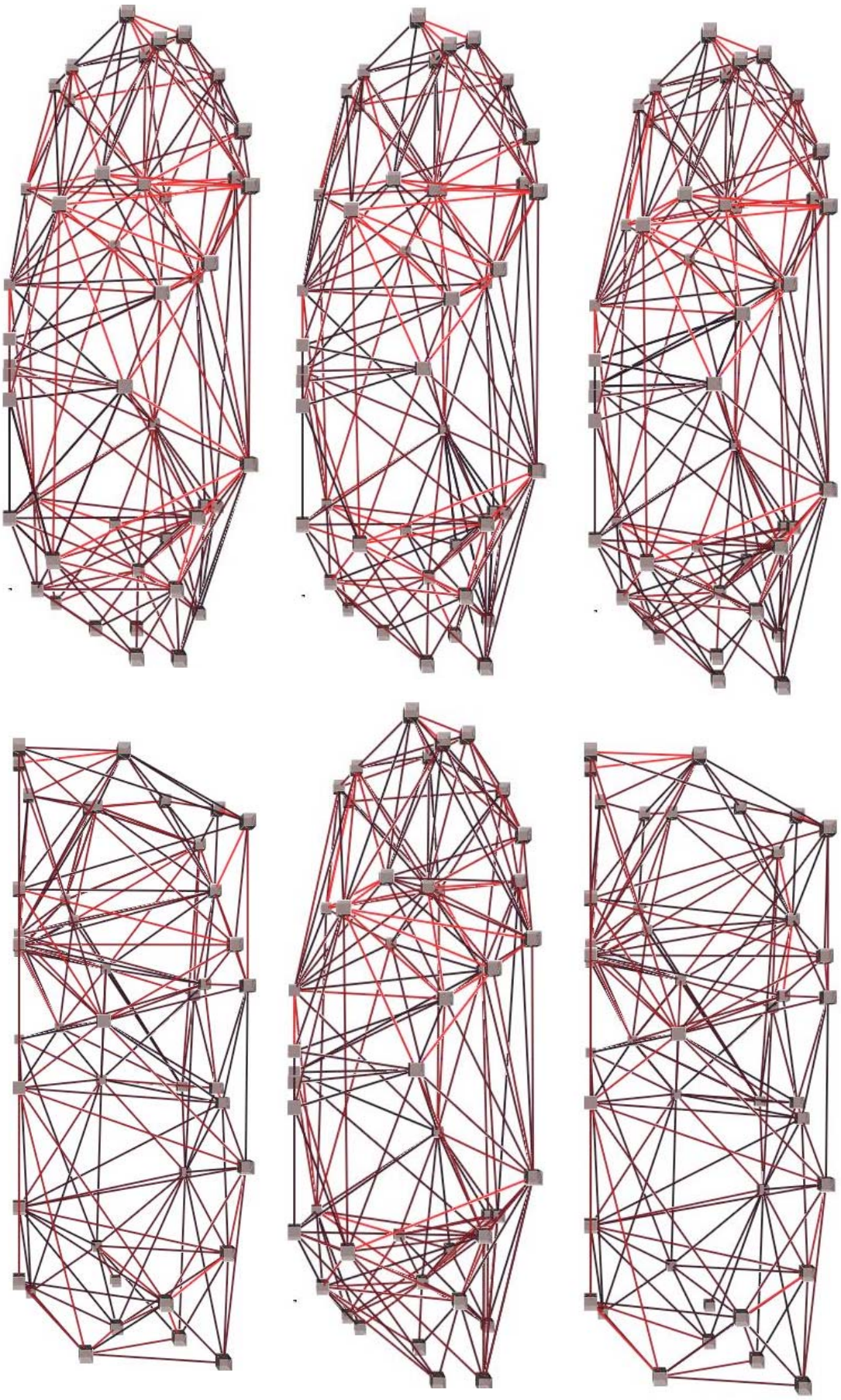


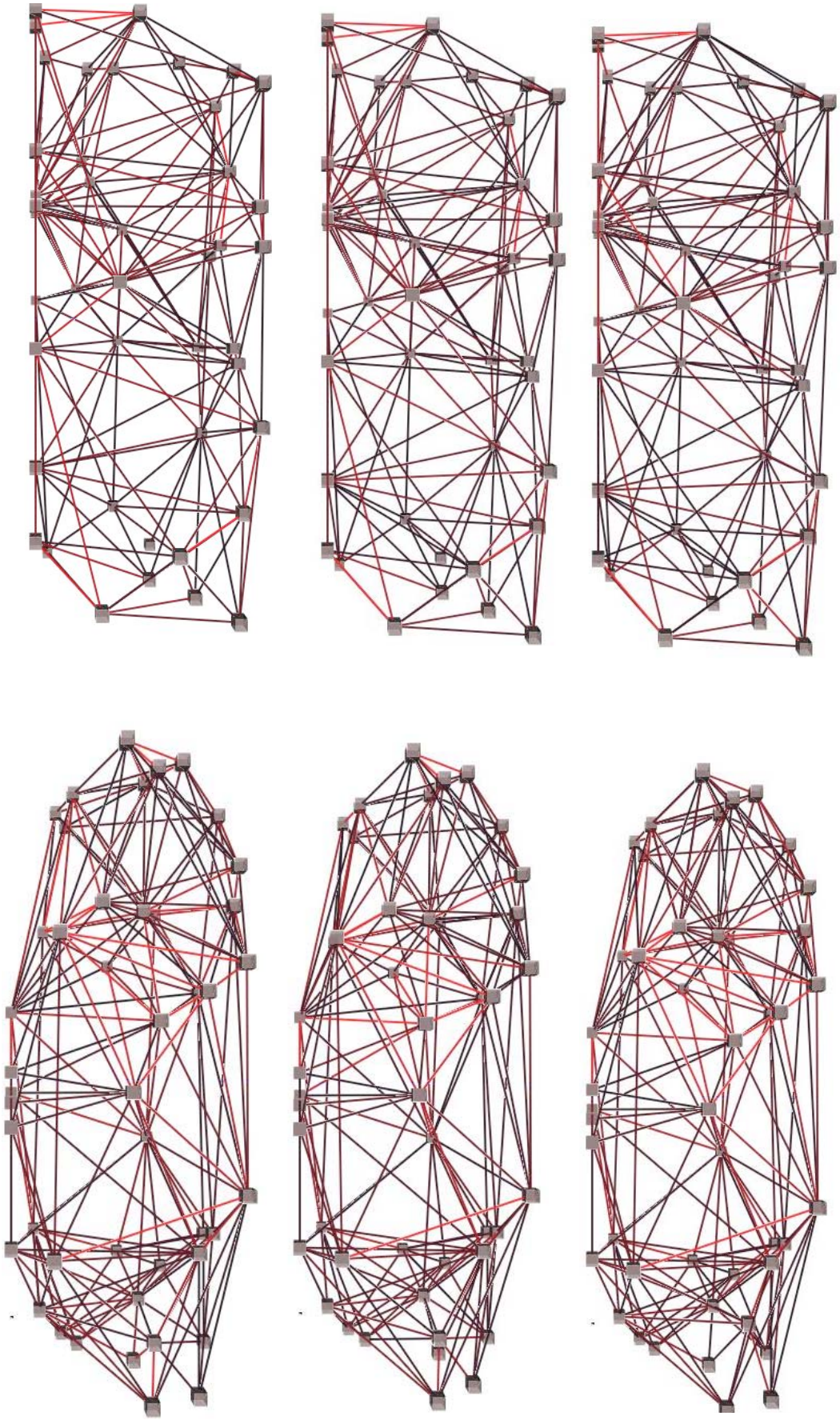
Second Experiment

Figure 27. Indicative samples every 80 generations









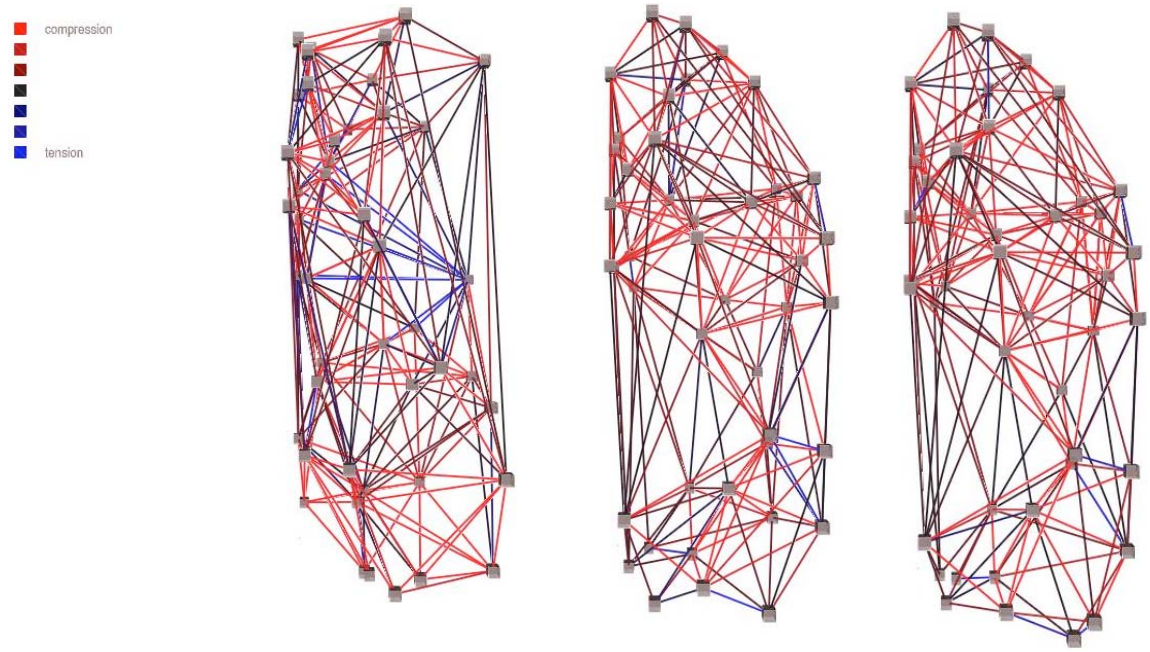
Appendix II

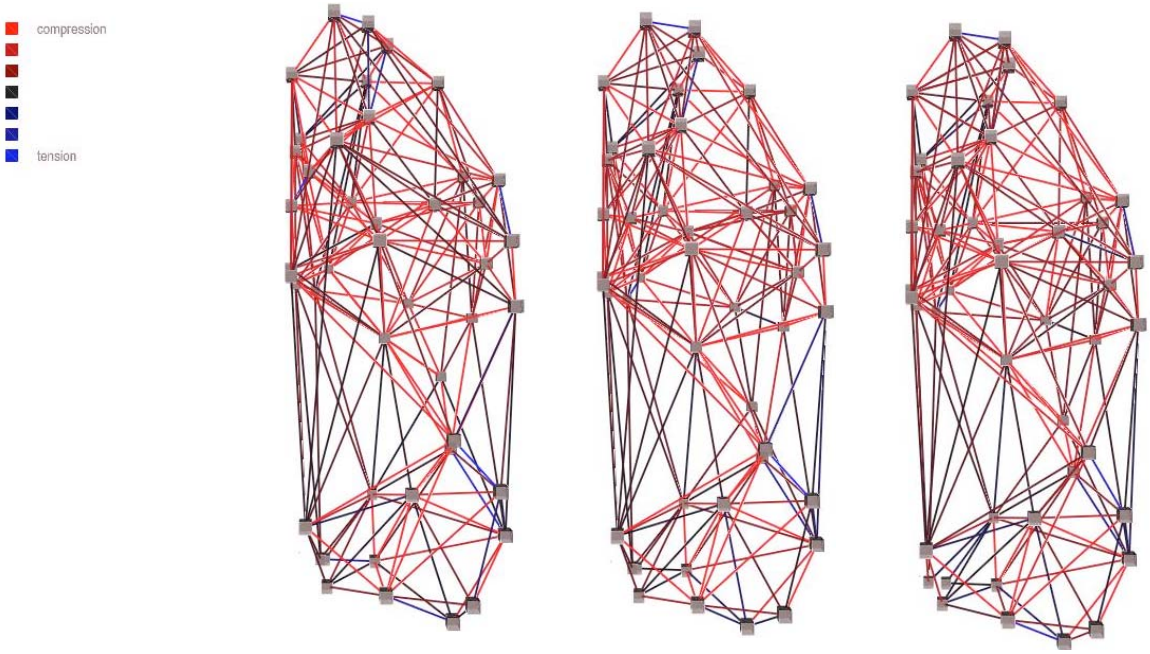
Self-weight Structures

Illustrations and results from several preliminary experiments executed on self-weighted space frames. Each of the followings experiment was executed with a different fitness function.

Experiment_3

Single Objective: Minimising the maximum deflection





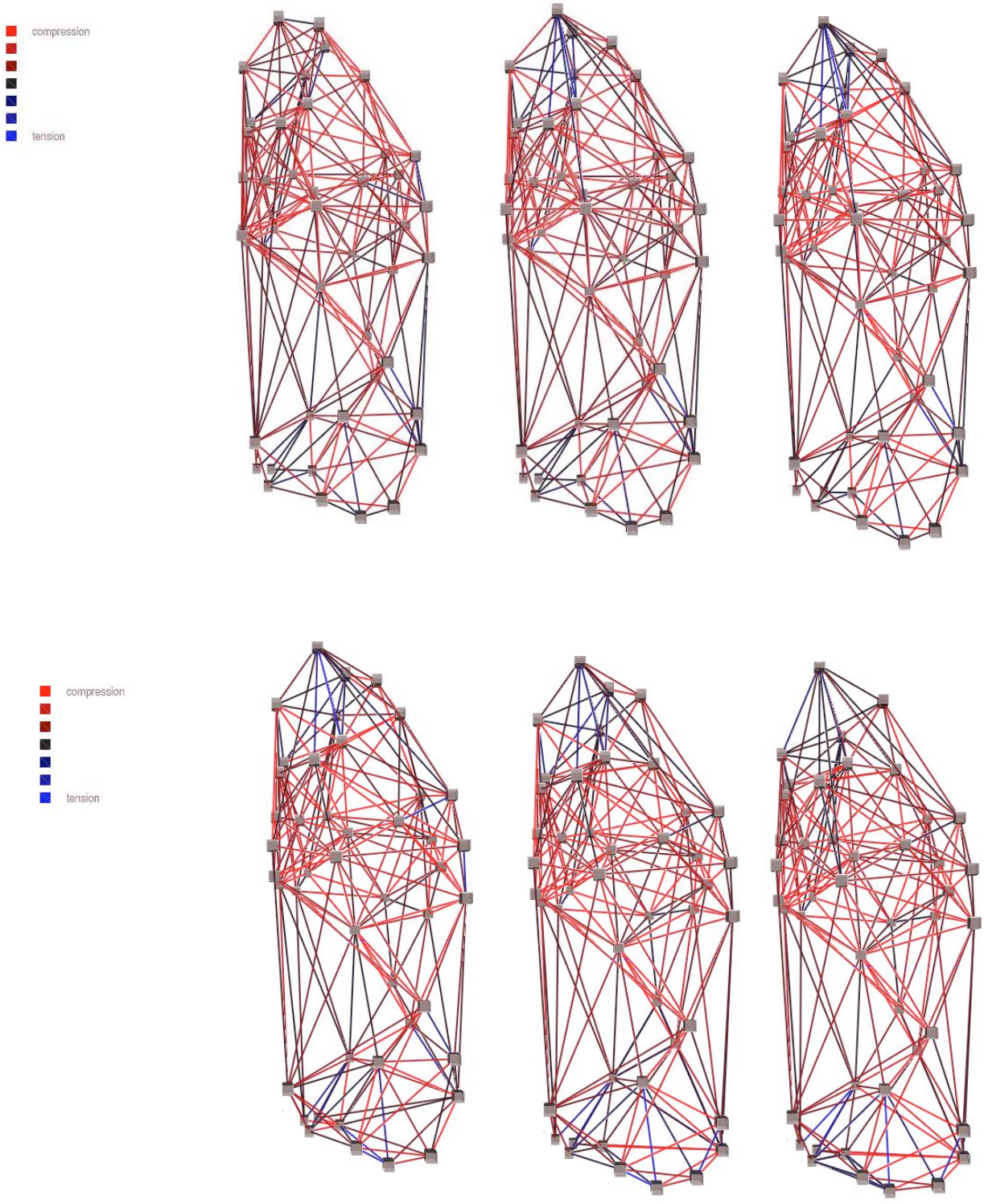


Figure 28. Experiment_3. Indicative generated samples captured after every 200 generations. The edges are coloured according to the nature of deformation. Red indicates compression whereas blue tension

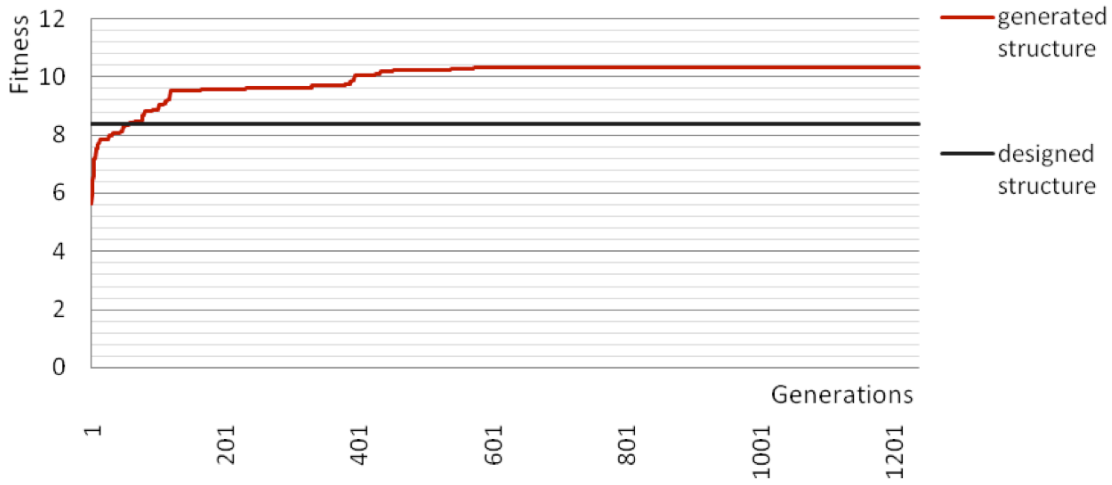


Figure 29. Experiment_3. Fitness value activity during runtime of the algorithm for 45 nodes

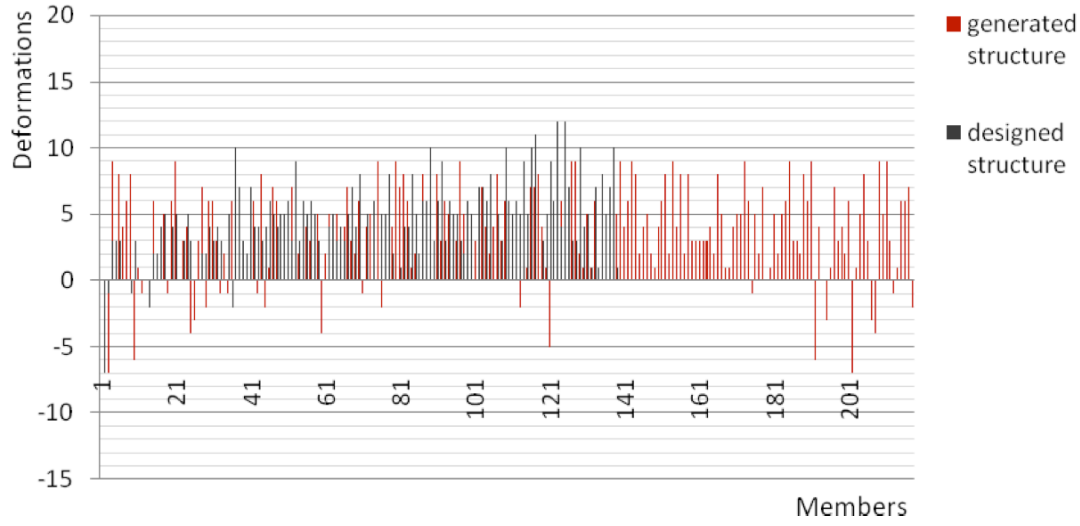
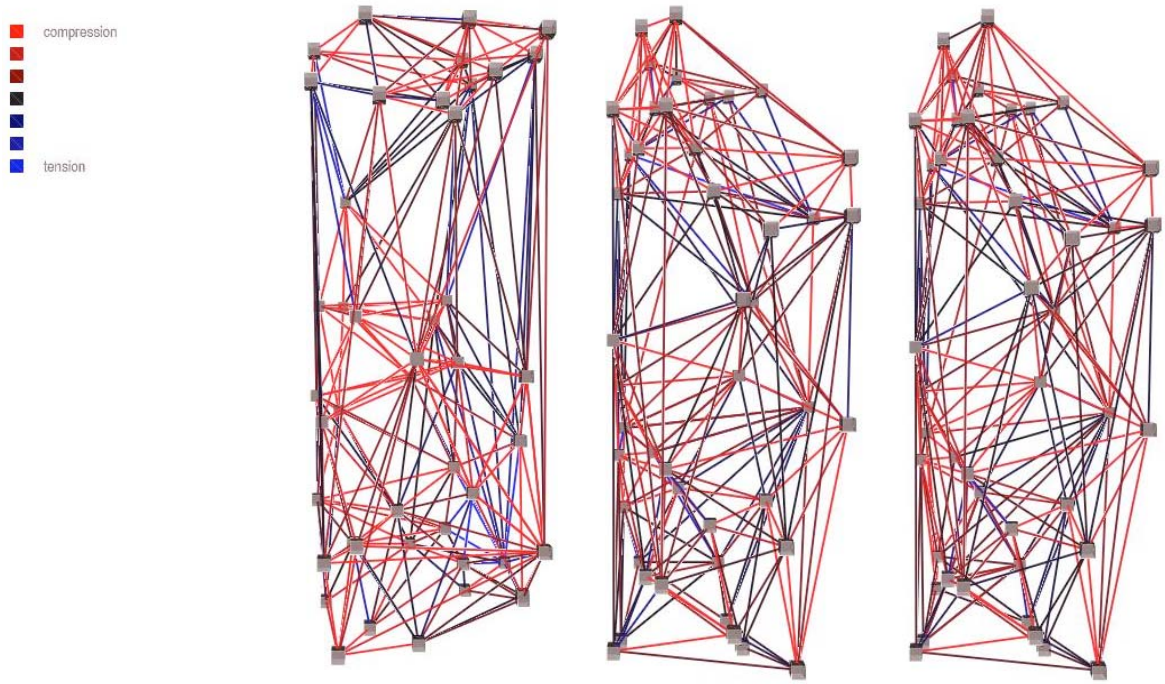
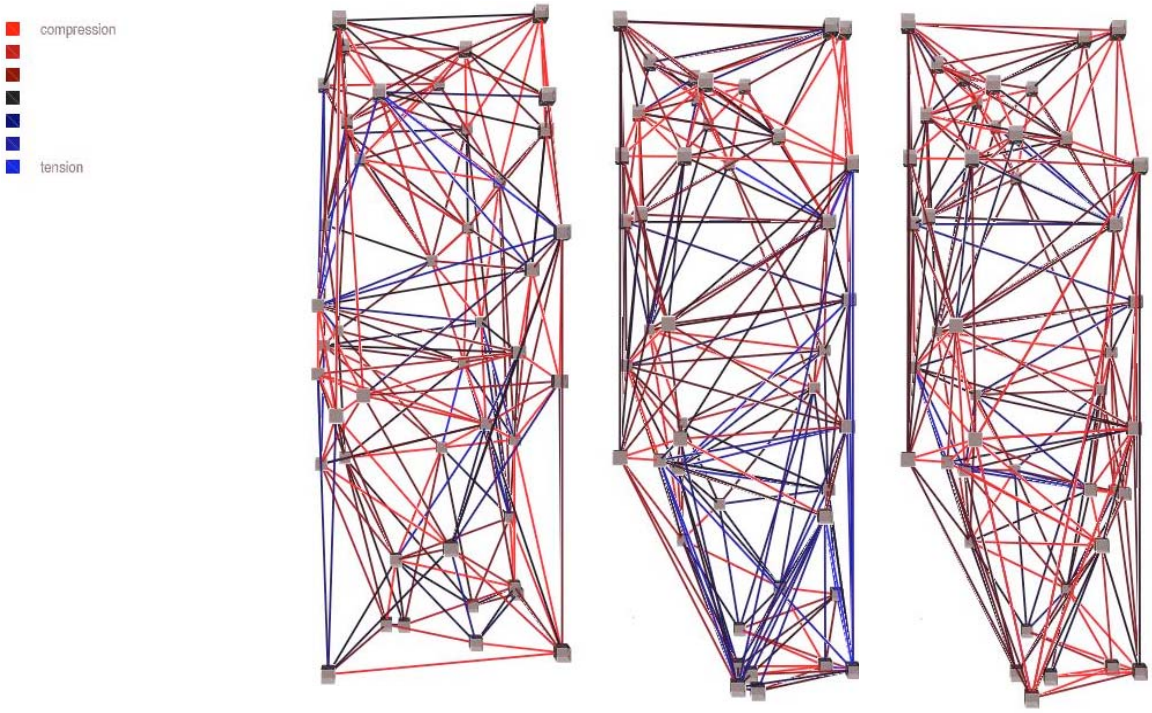


Figure 30. Experiment_3. Strain distribution of the generated space frame for 45 nodes after 1000 generations

Experiment_4

Single Objective: Minimising the average of all deflections





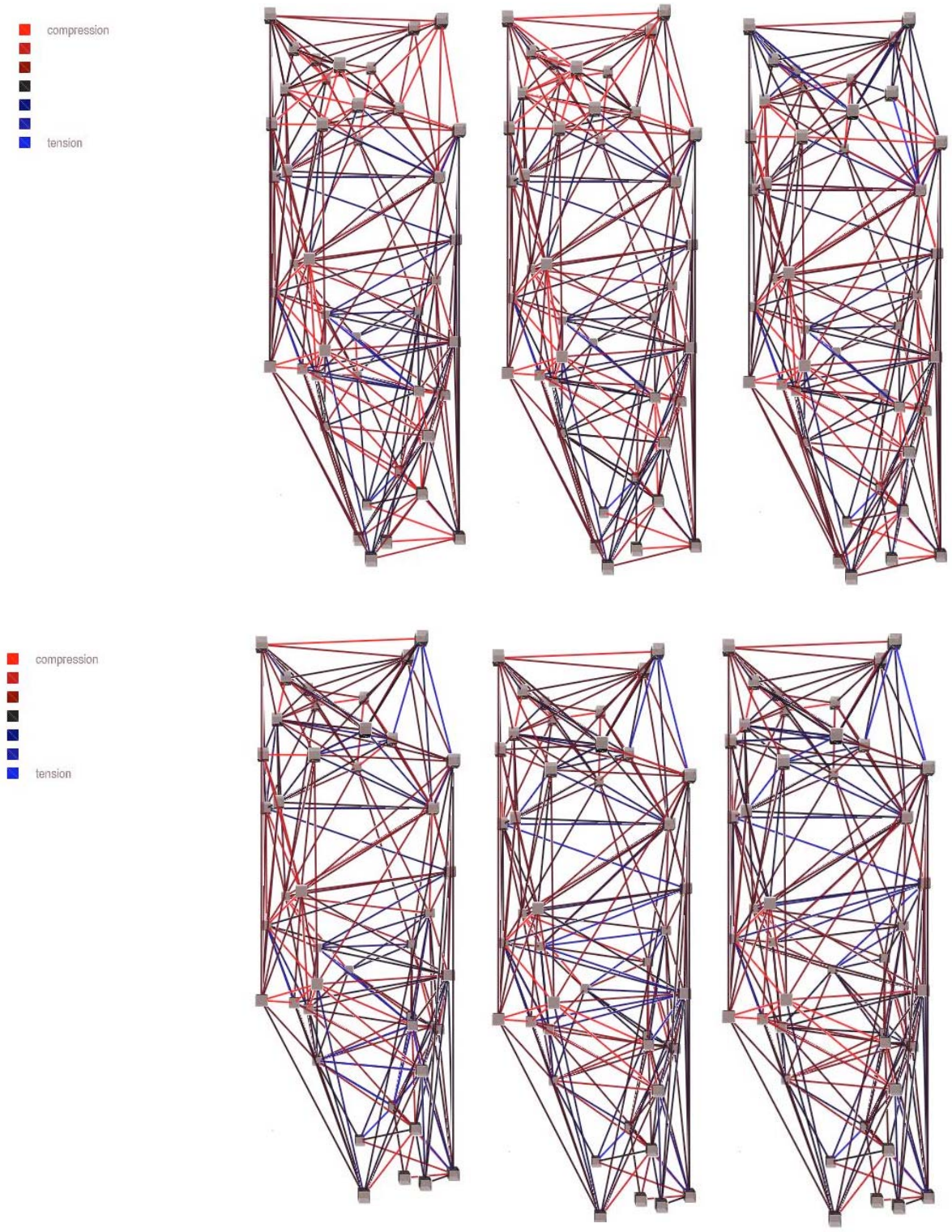


Figure 31. Experiment_4. Indicative generated samples captured after every 80 generations. The edges are coloured according to the nature of deformation. Red indicates compression whereas blue tension

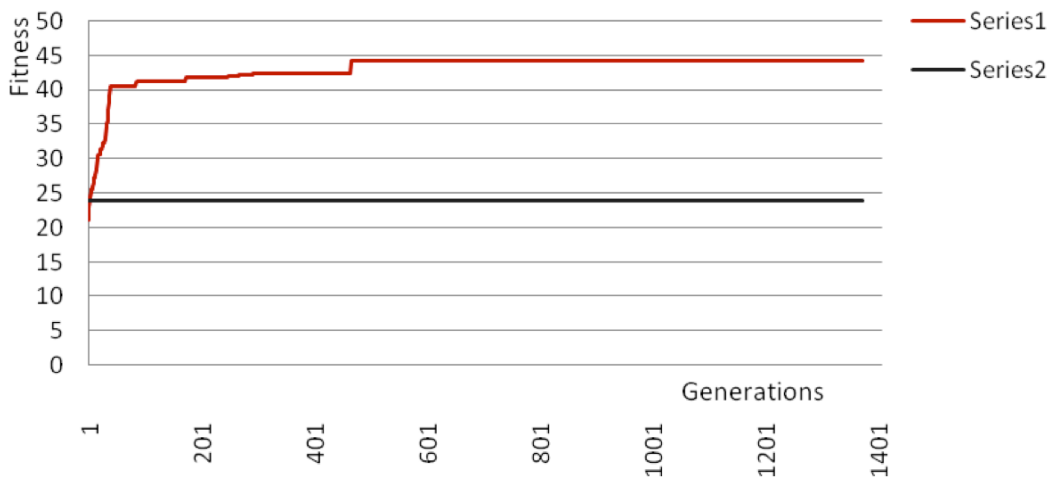


Figure 32. Experiment_4. Fitness value activity during runtime of the algorithm for 45 nodes

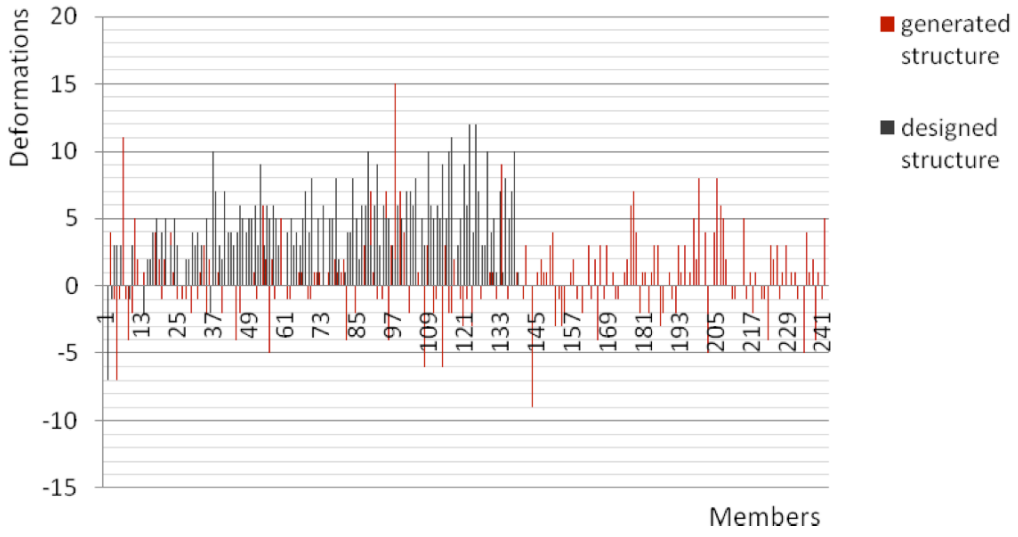
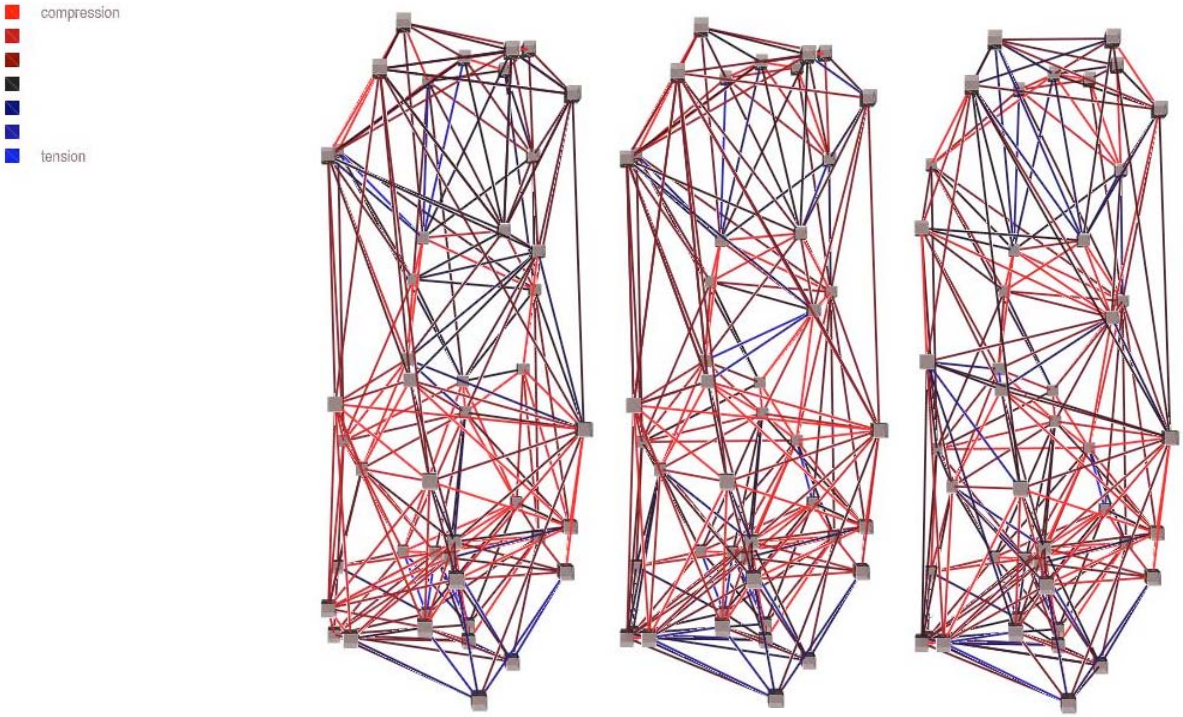
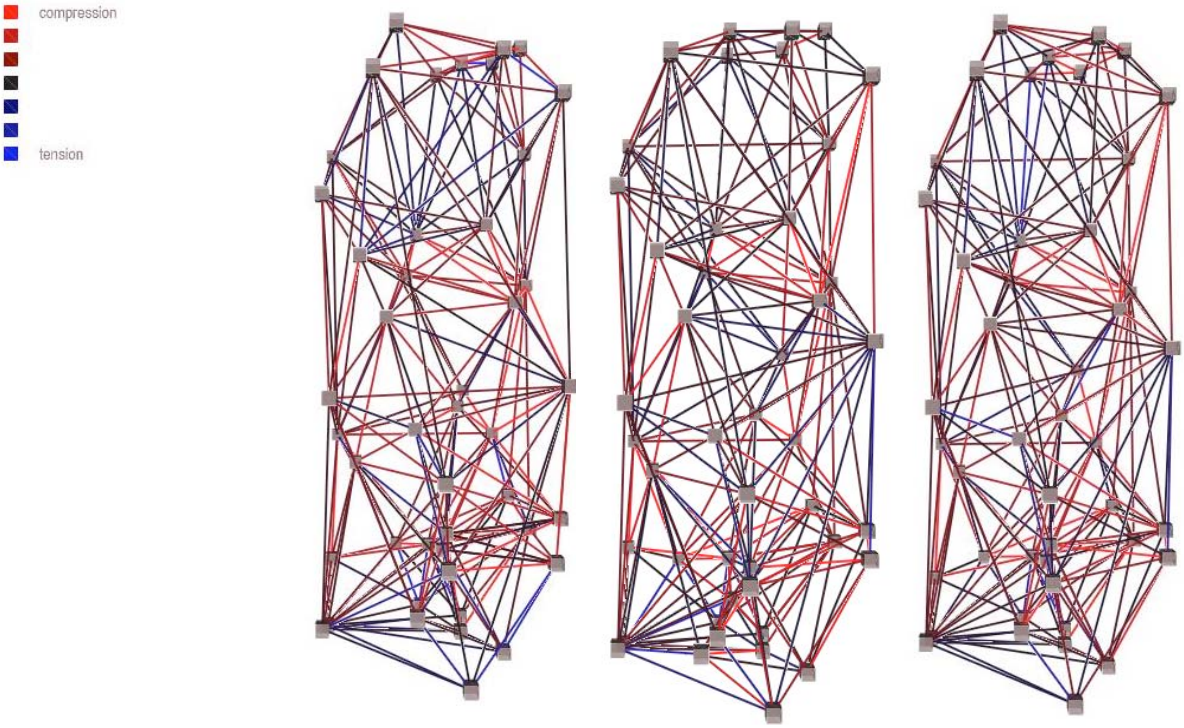


Figure 33. Experiment_4. Strain distribution of the generated space frame for 45 nodes after 1000 generations

Experiment_5

Multiple Objectives: Minimising the maximum angle and minimise the average of all deflections





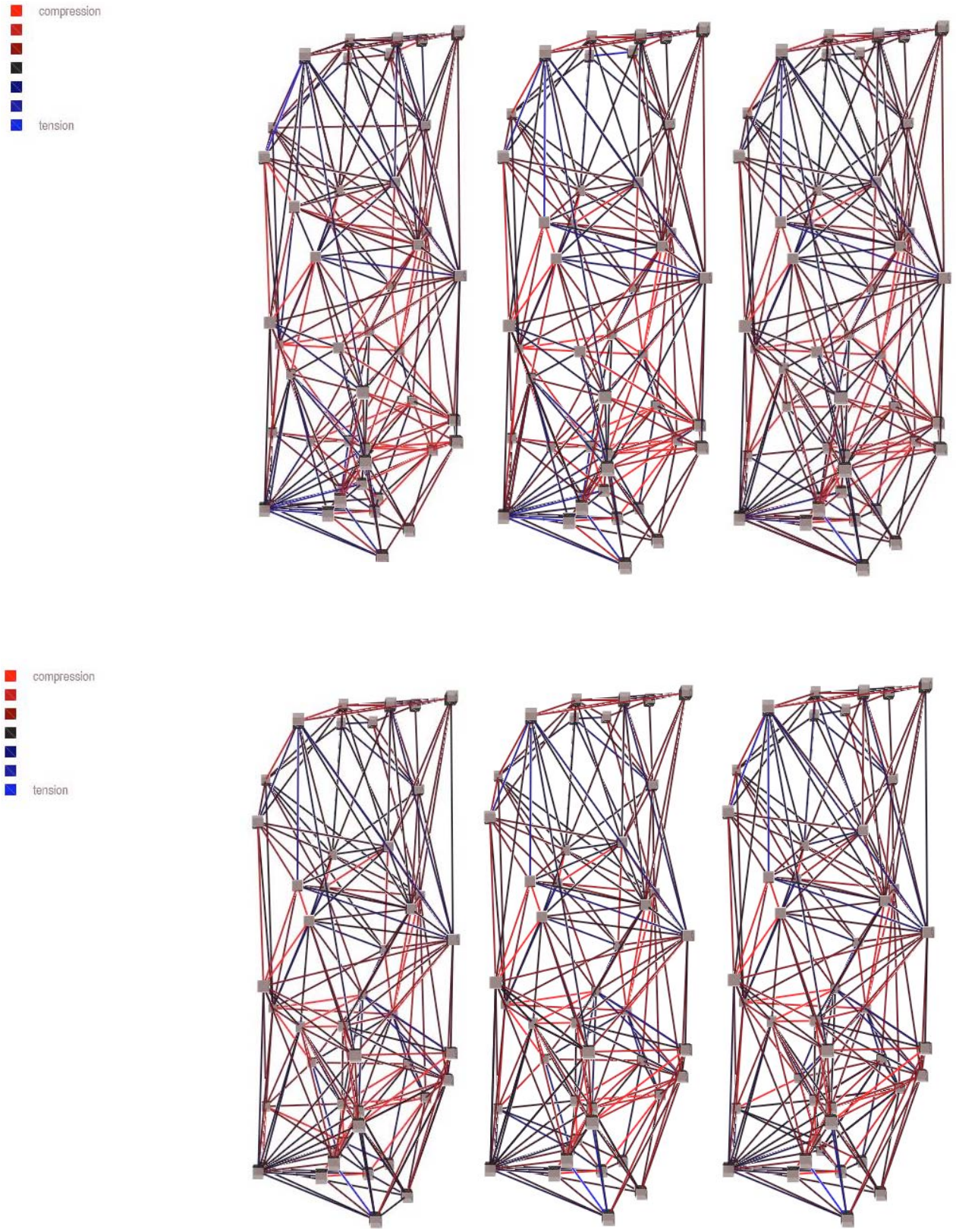


Figure 34. Experiment_5. Indicative generated samples captured after every 200 generations. The edges are coloured according to the nature of deformation. Red indicates compression whereas blue tension

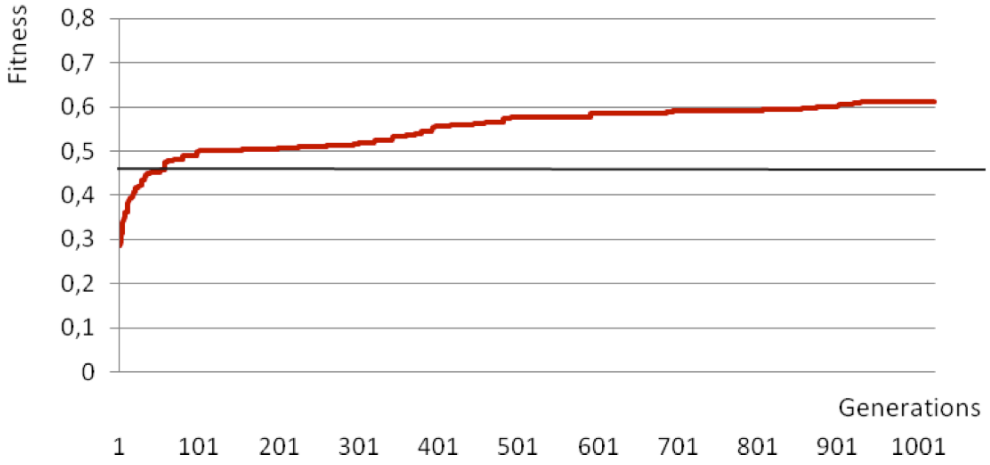


Figure 35. Experiment_5. Fitness value activity during runtime of the algorithm for 45 nodes

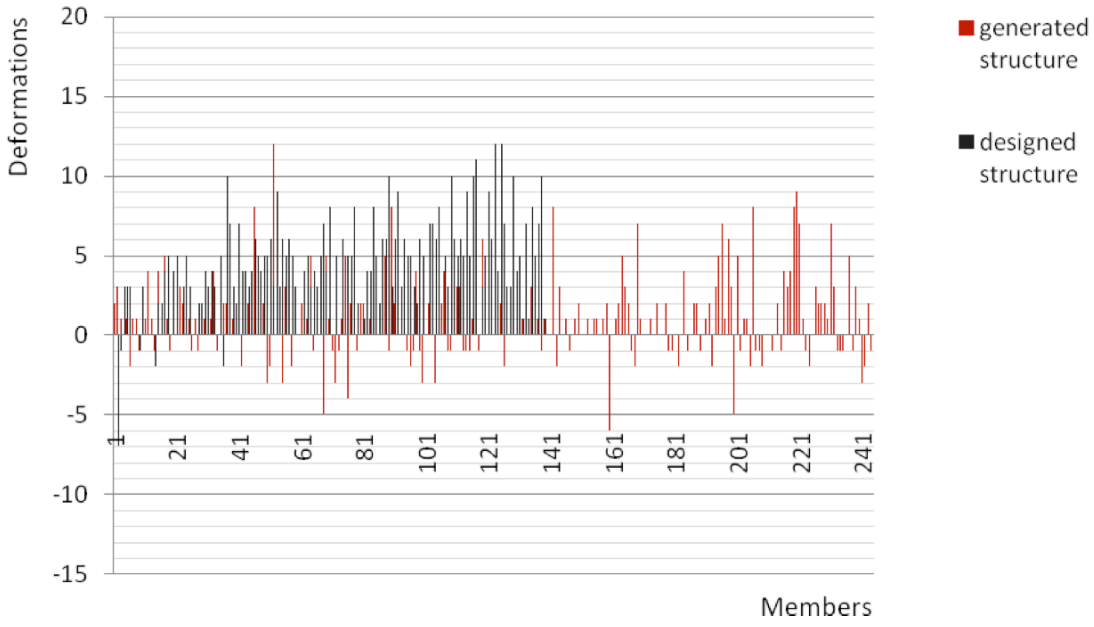


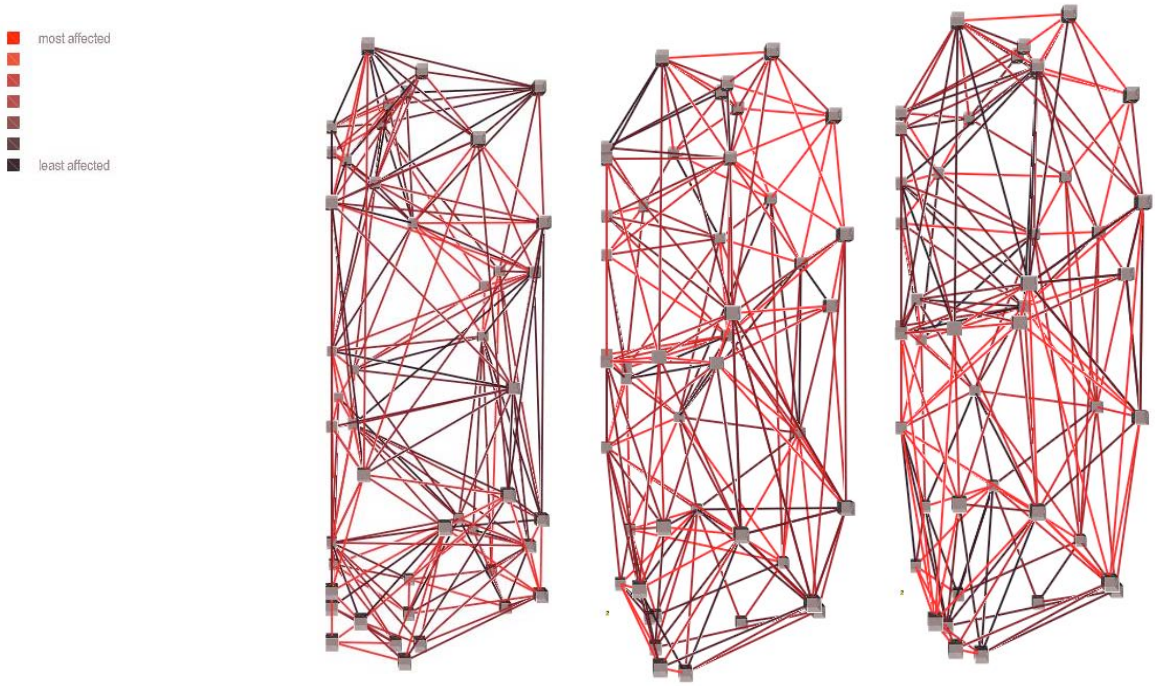
Figure 36. Experiment_5. Strain distribution of the generated space frame for 45 nodes after 1000 generations

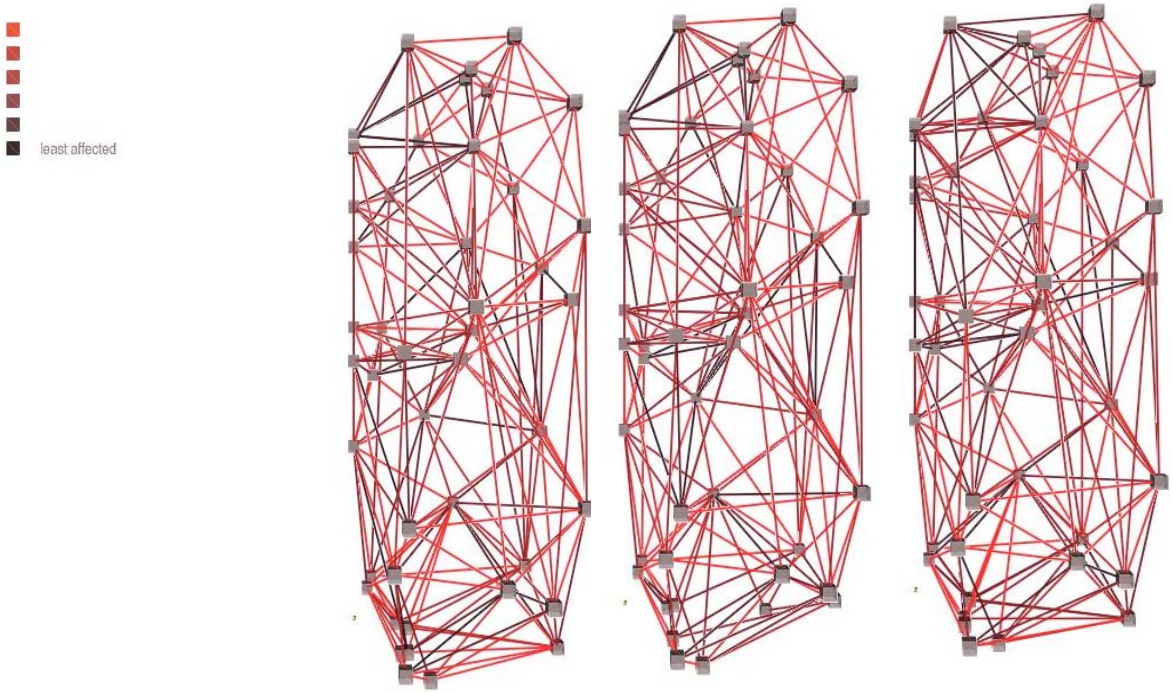
Application of two loads on the Structure

Illustrations and results from several experiments executed on space frames with two loads applied. Each of the followings experiment was executed with a different fitness function.

Experiment_6

Single Objective: To minimise the maximum deflection





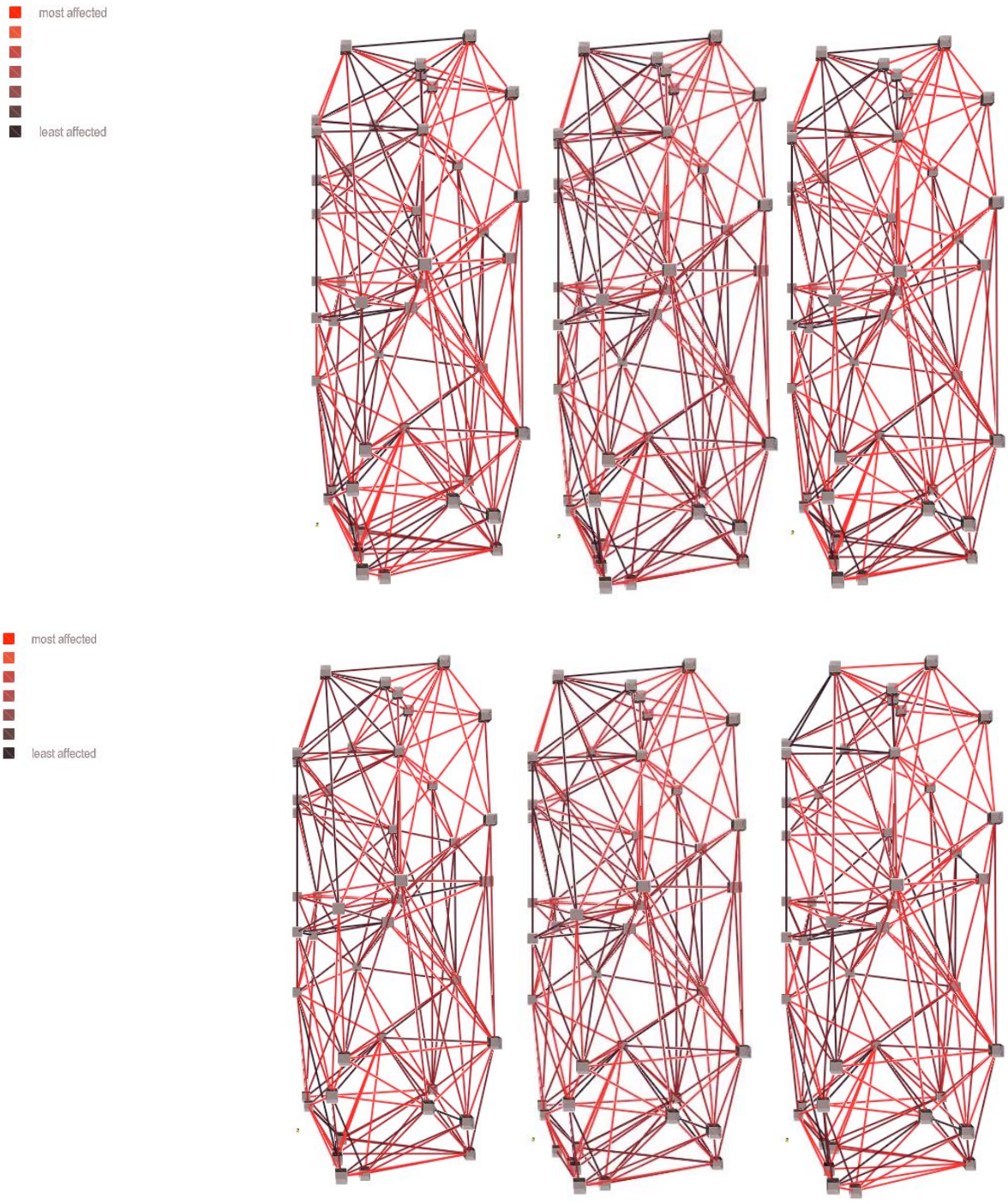


Figure 37. Indicative generated samples captured after every 200 generations. The edges are coloured according to the total deformation. The more red they are the more affected from the appliance of the two forces.

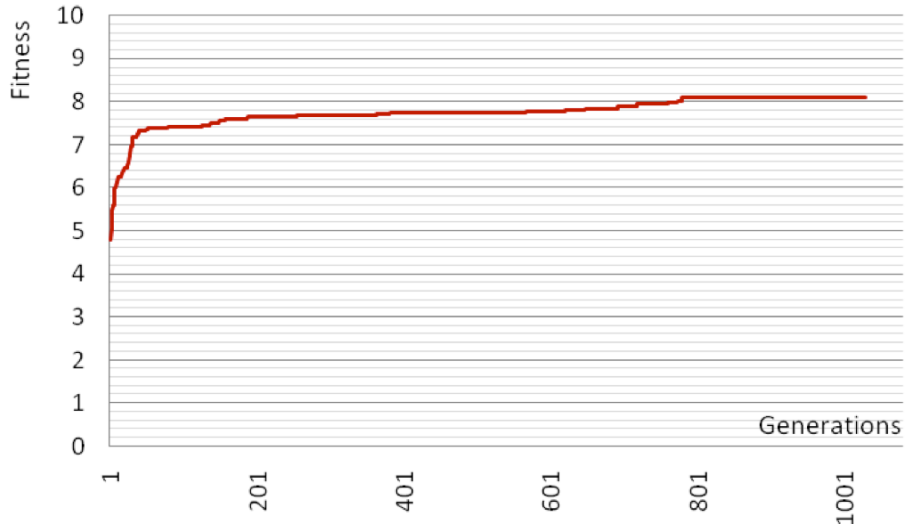


Figure 38. Fitness value activity during runtime of the algorithm for 45 nodes and two forces

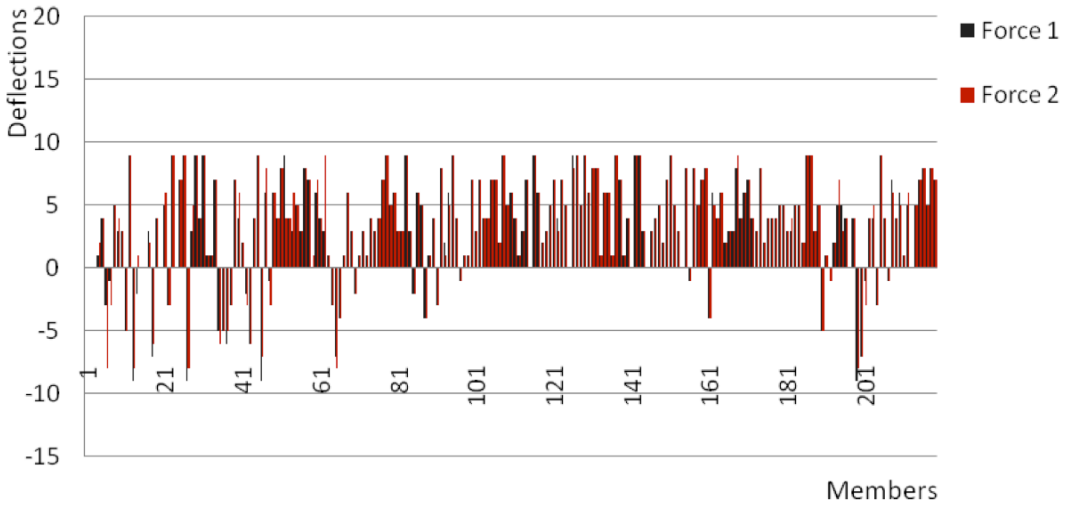


Figure 39. Experiment_6. Strain distribution of the generated space frame for 45 nodes after 1000 generations

Appendix III

Pseudocode (after JAVA)

Indicative Classes from Test.package

Basic functions of the Optimisation Class (main class)

```
public class Optimization  {

int numOfPoints = 45; //number of points
int popSize = 100;    //the size of the population
double x_min = 0;     //set the size of the solid
double x_max = 220;
double y_min = 0;
double y_max = 600;
double z_min = 0;
double z_max = 220;
int g = 0;
...

public Optimization() {
...
        IncrementalDel.createBigTetra(biggest);
//create the big tetrahedron inside of which all tetrahedra are generated
        Configuration conf = new MyConfiguration();
        conf.setPreservFittestIndividual(true);
        myFunc = new Fitness(this);
        CompositeGene[] sampleGenes = new CompositeGene[numOfPoints];
// Array of the composite genes

//set the points-genes inside the solid
try {
        for (int i = 0; i < numOfPoints - 40; i++) { //middle_randomm
                sampleGenes[i]= new CompositeGene(conf);
                sampleGenes[i].addGene(new DoubleGene(conf, x_min, x_max));
                sampleGenes[i].addGene(new DoubleGene(conf, y_min, y_max));
                sampleGenes[i].addGene(new DoubleGene(conf, z_min, z_max));
        }
...
for (int i = numOfPoints - 31; i < numOfPoints - 24; i++) { //middle_z.fixed
                sampleGenes[i]= new CompositeGene(conf);
                sampleGenes[i].addGene(new DoubleGene(conf, x_min, x_max));
                sampleGenes[i].addGene(new DoubleGene(conf, y_min, y_max));
                sampleGenes[i].addGene(new DoubleGene(conf, z_max, z_max));
        }
for (int i = numOfPoints - 24; i < numOfPoints - 17; i++) { //middle_x.fixed
                sampleGenes[i]= new CompositeGene(conf);
                sampleGenes[i].addGene(new DoubleGene(conf, x_min, x_min));
                sampleGenes[i].addGene(new DoubleGene(conf, y_min, y_max));
                sampleGenes[i].addGene(new DoubleGene(conf, z_min, z_max));
        }
...
for (int i = numOfPoints - 10; i < numOfPoints - 5; i++) { // down
                sampleGenes[i]= new CompositeGene(conf);
                sampleGenes[i].addGene(new DoubleGene(conf, x_min, x_max));
                sampleGenes[i].addGene(new DoubleGene(conf, y_min, y_min));
                sampleGenes[i].addGene(new DoubleGene(conf, z_min, z_max));
        }
for (int i = numOfPoints - 5; i < numOfPoints; i++) { // up_5
                sampleGenes[i]= new CompositeGene(conf);
```

```
                    sampleGenes[i].addGene(new DoubleGene(conf, x_min, x_max));
                    sampleGenes[i].addGene(new DoubleGene(conf, y_max, y_max));
                    sampleGenes[i].addGene(new DoubleGene(conf, z_min, z_max));
            }
Chromosome sampleChromosome = new Chromosome(conf, sampleGenes);
conf.setSampleChromosome(sampleChromosome);
conf.setPopulationSize(popSize);
conf.setFitnessFunction(myFunc);
population = Genotype.randomInitialGenotype(conf);
//initialise randomly the composite genes inside the solid
...
}
...
while (true) {
            tick(); //evolve
            }
}

public void tick() {
            g++;
            population.evolve();
            ...
}

...
public void sendConstruction(Construction2 construction, double fitness) {
            if (fitness>bestFitness){
                        bestConstruction=construction;
                        bestFitness=fitness;
                        ...
                        }
            }

public Construction2 getBestConstruction() {
            return bestConstruction;
            }
            ...
}
```

Basic functions of the Fitness Class

```
public class Fitness extends FitnessFunction {
double weight= 0.9;
private Optimization opt;

public Fitness(Optimization opt2) {
            opt = opt2;
}

public double evaluate(IChromosome chromosome) { //set the fitness values
            Construction2 construction = new Construction2(chromosome);
            double fitness1 = mo(construction);
            //double fitness2=maximum(construction);
            fitness1 = 1 / fitness1;
            //double fitness3 = minAngle(construction);
            //double fitness= weight*fitness3 + ((1-weight)*fitness1);
            double fitness=fitness1;
            opt.sendConstruction(construction, fitness);
            return fitness;
}

private double mo(Construction2 construction) {//average of deformations
```

```
		double sum = athroisma (construction);
		int lengths=construction.getD_a().length+construction.getD_b().length;
		return (sum/lengths);
		}

private double athroisma(Construction2 construction) { //sum of all deformations from the two forces
		float[] d_a = construction.getD_a();
		float[] d_b = construction.getD_b();
...
		for (int i = 0; i < d_a.length; i++) {
		athroisma_a  +=Math.abs(d_a[i]);
		}
		for (int i = 0; i < d_b.length; i++) {
			athroisma_b  +=Math.abs(d_b[i]);
		}
		athroisma= athroisma_a + athroisma_b;
		return athroisma;
		}

private double maximum(Construction2 construction) {
//saves the maximum of the deformations
		float[] d_a = construction.getD_a();
		float[] d_b = construction.getD_b();
		double maximum_a = 0;
		double maximum_b = 0;
		double maximum = 0;
		for (int i = 0; i < d_a.length; i++) {
			if (Math.abs(d_a[i]) > maximum_a)
				maximum_a = Math.abs(d_a[i]);
		}
		for (int i = 0; i < d_b.length; i++) {
			if (Math.abs(d_b[i]) > maximum_b)
			maximum_b = Math.abs(d_b[i]);
			}
		maximum = Math.max(maximum_a, maximum_b);
		return maximum;
}

private double minAngle(Construction2 construction) {
//calculates and returns the minimum angle
		double minAngle = Double.MAX_VALUE;
		for (int i = 0; i < construction.getPoints().length; i++) {
		Vector<Line> connected = new Vector<Line>();
		for (int j = 0; j < construction.getLines().length; j++) {
		if (construction.getPoints()[i].pos.getDist(construction.getLines()[j].getStart().pos) ==0
                                                        ||
construction.getPoints()[i].pos.getDist(construction.getLines()[j].getEnd().pos)==0)
                                                    connected.add(construction.getLines()[j]);
}
		for (int k = 0; k < connected.size(); k++) {
			for (int l = k+1; l < connected.size(); l++){
				double angle= angle(connected.get(k), connected.get(l));
				if( angle< minAngle)
				minAngle= angle;
			}
		}
		}
		return minAngle;
		}
...
}
```

Indicative Classes from Basic.package

Basic functions of the Construction2 Class

```
public class Construction2 {

public Boid[] points; // all points
public Line[] lines; // all lines
public float[] d_a; // deformation_a
public float[] d_b; // deformation_b
private Simplex[] tetrahedra; // tetrahedra

public Construction2(IChromosome chromosome) {
        Gene[] genes = chromosome.getGenes();
        points = makePoints(genes);
        ParticleSystem ps1 = null;
        ParticleSystem ps2 = null;

if (true) {
        IncrementalDel triangulation = new IncrementalDel(points);
        lines = triangulation.getLines();
        tetrahedra = triangulation.getTetra();
        }

ps1 = makeParticleSystem_1(points, lines);
ps2 = makeParticleSystem_2(points, lines);
//deflections from force 1
float d1[] = new float[ps1.numberOfSprings()];
float d2[] = new float[ps1.numberOfSprings()];

for (int i = 0; i < ps1.numberOfSprings(); i++) {
        d1[i] = ps1.getSpring(i).currentLength();
}

ps1.advanceTime(0.5f);

for (int i = 0; i < ps1.numberOfSprings(); i++) {
        d2[i] = ps1.getSpring(i).currentLength();
}
d_a = new float[ps1.numberOfSprings()];
for (int i = 0; i < ps1.numberOfSprings(); i++) {
        d_a[i] = (d1[i] - d2[i]) * 100 / d1[i];
}

ps2 = makeParticleSystem_2(points, lines);
//deflections from force 2
float d3[] = new float[ps2.numberOfSprings()];
float d4[] = new float[ps2.numberOfSprings()];

for (int i = 0; i < ps2.numberOfSprings(); i++) {
...
        }
//Force 1
private ParticleSystem makeParticleSystem_1(Boid[] points, Line[] lines) {
Particle[] particles = new Particle[points.length];
ParticleSystem makesystem = new ParticleSystem(0.2f, 0.1f, 0.0f, 0.01f);
        for (int i = 0; i < points.length; i++) {
particles[i] = makesystem.makeParticle((float) 0.5,
(float) points[i].getX(), (float) points[i].getY(),(float)points[i].getZ());
        if (points[i].getY() == 0)
                particles[i].makeFixed();
                else
                particles[i].makeFree();
                }
```

```
for (int i = 0; i < lines.length; i++) {
                        Line line = lines[i];
Boid start = line.getStart();
                        Boid end = line.getEnd();
                        int startID = start.getId();
                        int endID = end.getId();
                        Particle startParticle = particles[startID];
                        Particle endParticle = particles[endID];
float dist = startParticle.position().distanceTo( endParticle.position());
makesystem.makeSpring(startParticle, endParticle, 2, 0.25f, dist);
            }
            return makesystem;
}

//Force 2
private ParticleSystem makeParticleSystem_2(Boid[] points, Line[] lines) {
...
return makesystem;
            }

//make points out of genes
public Boid[] makePoints(Gene[] genes) {
int numOfPoints = genes.length;
Boid[] points = new Boid[numOfPoints];
for (int i = 0; i < numOfPoints; i++) {
            CompositeGene gene = (CompositeGene) genes[i];
            List<Gene> doubleGenes = gene.getGenes();
            double x = ((DoubleGene) doubleGenes.get(0)).doubleValue();
            double y = ((DoubleGene) doubleGenes.get(1)).doubleValue();
            double z = ((DoubleGene) doubleGenes.get(2)).doubleValue();
            points[i] = new Boid(i, x, y, z);
            }
            return points;
}
...
}
```

Indicative Classes from MyGenetic.package

Basic functions of the MyConfiguration Class

```
//This file is part of JGAP.
//authors Neil Rotstan, Klaus Meffert

public class MyConfiguration extends Configuration implements ICloneable {
...
//Constructs a new Configuration instance with a number of configuration settings.
  public MyConfiguration(String a_id, String a_name) {
    super(a_id, a_name);
    try {
      setBreeder(new GABreeder());
      setRandomGenerator(new StockRandomGenerator());
      setEventManager(new EventManager());
      TournamentSelector selector = new TournamentSelector(this, 40, 0.90d);
// sets the tournament selection,selects 40 individuals where the fittest //wins with probalility 0.9
selector.setDoubletteChromosomesAllowed(true);
       addNaturalSelector(selector, false);
       setMinimumPopSizePercent(0);
       setSelectFromPrevGen(1.0d);
       setKeepPopulationSizeConstant(true);
       setFitnessEvaluator(new DefaultFitnessEvaluator());
       setChromosomePool(new ChromosomePool());
addGeneticOperator(new MyCrossoverOperator(this, 0.7d));
// 0.7= a_crossoverRatePercentage which is the desired rate of crossover in percentage of the population
addGeneticOperator(new MyMutationOperator(this, 10)
//mutates 10 individuals
    }
...
}
```

Basic functions of the MyCrossoverOperator Class

```
//This file is part of JGAP.
//authors Neil Rotstan, Klaus Meffert, Chris Knowles

public class MyCrossoverOperator extends BaseGeneticOperator implements
                        Comparable {
...
//The crossover operator randomly selects two Chromosomes from the //population and "mates" them by
randomly picking a gene and then swapping //that gene and all subsequent genes between the two
Chromosomes. The two //modified Chromosomes are then added to the list of candidate Chromosomes
//Uniform crossover is applied here

public void operate(final Population a_population, final List a_candidateChromosomes) {

// Work out the number of crossovers that should be performed.
int size = Math.min(getConfiguration().getPopulationSize(),
                                            a_population.size());
int numCrossovers = 0;
if (m_crossoverRate >= 0) {
numCrossovers = size / m_crossoverRate;
} else if (m_crossoverRateCalc != null) {
            numCrossovers = size / m_crossoverRateCalc.calculateCurrentRate();
} else {
            numCrossovers = (int) (size * m_crossoverRatePercent);
            }
RandomGenerator generator = getConfiguration().getRandomGenerator();
```

```
IGeneticOperatorConstraint constraint = getConfiguration()
                                        .getJGAPFactory().getGeneticOperatorConstraint();
// For each crossover, grab two random chromosomes, pick a random
// locus (gene location), and then swap that gene and all genes
// to the "right" (those with greater loci) of that gene between
// the two chromosomes.
int index1, index2;
        for (int i = 0; i < numCrossovers; i++) {
        index1 = generator.nextInt(size);
        index2 = generator.nextInt(size);
        IChromosome chrom1 = a_population.getChromosome(index1);
        IChromosome chrom2 = a_population.getChromosome(index2);
// Verify that crossover is allowed.
        if (!isXoverNewAge() && chrom1.getAge() < 1 && chrom2.getAge() < 1) {
// Crossing over two newly created chromosomes is not seen as
// helpful here
        continue;
}
        if (constraint != null) {
                List v = new Vector();
                v.add(chrom1);
                v.add(chrom2);
        if (!constraint.isValid(a_population, v, this)) {
// Constraint forbids crossing over.
        continue;
}
}
// Clone the chromosomes.
        IChromosome firstMate = (IChromosome) chrom1.clone();
        IChromosome secondMate = (IChromosome) chrom2.clone();
// Cross over the chromosomes after you have sorted them
        sort(firstMate);
        sort(secondMate);
doCrossover(firstMate, secondMate, a_candidateChromosomes,
                                                generator);
                }
        }
//sorting the composite genes
private void sort (IChromosome in){
HashMap<Double, CompositeGene> map= new HashMap<Double, CompositeGene>();
PriorityQueue<Double> pq=new PriorityQueue<Double>();
Gene[] genesin = in.getGenes();

for (int ii=0 ; ii<genesin.length;ii++){
        CompositeGene compositeG =(CompositeGene) genesin[ii];
        double x = ((DoubleGene)compositeG.getGenes().get(1)).doubleValue();
//x
        double y = ((DoubleGene)compositeG.getGenes().get(1)).doubleValue();
//y
        double z = ((DoubleGene)compositeG.getGenes().get(2)).doubleValue();
//z
        double key=y+z*0.00001+x*0.000000001;
        while (map.containsKey(key)){
        ...
        key+=0.000000001;
                }
        map.put(key,compositeG);
        pq.add(key);
}
Gene[] geneout=new Gene[genesin.length];
        for (int ii=0 ; ii<genesin.length;ii++){
        Double y = pq.poll();
        geneout[ii]=map.remove(y);
}
...
        }
```

```
private void doCrossover(IChromosome firstMate, IChromosome secondMate,
        List a_candidateChromosomes, RandomGenerator generator) {
        Gene[] firstGenes = firstMate.getGenes()
Gene[] secondGenes = secondMate.getGenes();
        int locus = generator.nextInt(firstGenes.length);
// Swap the genes.
        Gene gene1;
Gene gene2;
        Object firstAllele;
//operate Uniform crossover with probability 0.5
for (int j = 0; j < firstGenes.length; j++) {
                if (Math.random() >= 0.5) {
                gene1 = firstGenes[j];
                gene2 = secondGenes[j];
firstAllele = gene1.getAllele();
                gene1.setAllele(gene2.getAllele());
                gene2.setAllele(firstAllele);
                }
        }
// Add the modified chromosomes to the candidate pool so that
// they'll be considered for natural selection during the next
// phase of evolution.
        a_candidateChromosomes.add(firstMate);
        a_candidateChromosomes.add(secondMate);
}
...
}
```

Basic functions of the MyMutationOperator Class

```
//This file is part of JGAP.
//authors Neil Rotstan, author Klaus Meffert

//The mutation operator runs through the genes in each of the Chromosomes in
//the population and mutates them in statistical accordance to the given
//mutation rate. Mutated Chromosomes are then added to the list of candidate
//Chromosomes destined for the natural selection process.

private void mutateCompositeGene(final ICompositeGene compositeGene, final RandomGenerator
a_generator) {
        for (int k = 0; k < compositeGene.size(); k++) {
        double percentage = -1 + a_generator.nextDouble() * 2;
        percentage = percentage/10
//local mutation
                for (int l = 0; l < compositeGene.size(); l++) {
                Gene a_gene = compositeGene.geneAt(k);
                a_gene.applyMutation(k, percentage);
                }
        }
}
```

Indicative Classes from Incremental_Delaunay.package

Basic functions of the IncrementalDel Class

```
public class IncrementalDel extends Triangulation {

private Simplex mostRecent = null; // Most recently inserted triangle
public boolean debug = false; // Used for debugging
private int tricounter;
public static Simplex bigTetra = null;

//Constructor 1. All sites must fall within the initial triangle(bigTetra)
public IncrementalDel(Simplex triangle, int tricounter) {
        if (triangle == null)
        init(triangle);
        mostRecent = triangle;
        this.tricounter = tricounter;
}

//Constructor 2.
public IncrementalDel(Boid[] points) {
        this(bigTetra, 1);
        for (int i = 0; i < points.length; i++) {
Boid b = new Boid(i, points[i].getX(), points[i].getY(), points[i].getZ());
                this.delaunayPlace(b);
        }
}
...

//Place a new point site into the DT.
public Set<Simplex> delaunayPlace(Boid site) {
        Set<Simplex> newTriangles = new HashSet<Simplex>();
        Set<Simplex> oldTriangles = new HashSet<Simplex>();
        Set<Simplex> doneSet = new HashSet<Simplex>();
        LinkedList<Simplex> waitingQ = new LinkedList<Simplex>();
// Locate containing triangle
        if (debug)
        System.out.println("Locate");
        Simplex triangle = locate(site.pos, site.getId());
// Give up if no containing triangle or if site is already in DT
        if (triangle != null && triangle.contains(site))
        return newTriangles;
// Find Delaunay cavity (those triangles with site in their circumcircles)
        if (debug)
        System.out.println("Cavity");
        waitingQ.add(triangle);
        if (triangle != null)
        while (!waitingQ.isEmpty()) {
        triangle = (Simplex) waitingQ.removeFirst();
        Boid[] b = (Boid[]) triangle.toArray(new Boid[0]);
        Pnt[] p = new Pnt[b.length];
        for (int i = 0; i < p.length; i++) {
                p[i] = b[i].pos;
        }
                if (site.pos.vsCircumcircle(p) == 1)
                continue;
                oldTriangles.add(triangle);
                Iterator<Simplex> it = this.neighbors(triangle).iterator();
                for (; it.hasNext();) {
                        Simplex tri = (Simplex) it.next();
                        if (doneSet.contains(tri))
                        continue;
                        doneSet.add(tri);
                        aitingQ.add(tri);
                }
```

```
            }
// Create the new triangles
            if (debug)
            System.out.println("Create");
for (Iterator<Set<Boid>> it = Simplex.boundary(oldTriangles).iterator();
it.hasNext();) {
Set<Boid> facet = it.next();
            facet.add(site);
            newTriangles.add(new Simplex(facet, tricounter));
            tricounter++;
}
// Replace old triangles with new triangles
            if (debug)
            System.out.println("Update");
            this.update(oldTriangles, newTriangles);
// Update mostRecent triangle
            if (!newTriangles.isEmpty())
            mostRecent = (Simplex) newTriangles.iterator().next();
            return newTriangles;
}
...
}
```

Basic functions of the Simplex Class

```
public class Simplex extends AbstractSet<Boid> implements Set<Boid> {

public List<Boid> vertices;// The simplex's vertices
public long idNumber;// The id number
public static boolean moreInfo = false;//True if more info in toString

//Constructor 1
public Simplex(Collection<Boid> collection, int id) {
this.vertices = Collections.unmodifiableList(new        ArrayList<Boid>(collection));
this.idNumber = id;
}

//Constructor 2
public Simplex(Boid[] vertices, int id) {
            this(Arrays.asList(vertices), id);
}

//True iff simplices are neighbors. Two simplices are neighbors if they are the same dimension and they share a
facet.
public boolean isNeighbor(Simplex simplex) {
            HashSet<Boid> h = new HashSet<Boid>(this);
            h.removeAll(simplex);
            return (this.size() == simplex.size()) && (h.size() == 1);
}

//Report the facets of this Simplex. Each facet is a set of vertices.
public List<Set<Boid>> facets() {
            List <Set<Boid>> theFacets = new LinkedList<Set<Boid>>();
            for (Iterator<Boid> it = this.iterator(); it.hasNext();) {
                        Boid v = it.next();
                        Set<Boid> facet = new HashSet<Boid>(this);
                        facet.remove(v);
                        theFacets.add(facet);
            }
            return theFacets;
```

```
}

public List  <Set<Boid>> facets2() {
        List <Set<Boid>> theFacets = new LinkedList <Set<Boid>> ();
        Boid old = null;
        Boid first = null;
        for (Iterator<Boid> it = this.iterator(); it.hasNext();) {
                Boid v = it.next();
                if (old != null) {
                        Set<Boid> facet = new HashSet<Boid>();
                        facet.add(old);
                        facet.add(v);
                        theFacets.add(facet);
                } else {
                        first = v;
                }
                old = v;
        }
        Set<Boid> facet = new HashSet<Boid>();
        facet.add(old);
        facet.add(first);
        theFacets.add(facet);
        return theFacets;
}
...
}
```

Basic functions of the Triangulation Class

```
//author L. Paul Chew
//A Triangulation is a set of Simplices. For efficiency, we keep track of //the neighbours of each Simplex. Two Simplices are neighbours of they share //a facet.

public class Triangulation {
private HashMap<Simplex, HashSet<Simplex>> neighbors;
// Maps Simplex to Set of neighbours

//Constructor
public void init(Simplex simplex) {
        neighbors = new HashMap<Simplex, HashSet<Simplex>>();
        neighbors.put(simplex, new HashSet<Simplex>());
}

//String representation. Shows number of simplices currently in the //Triangulation.
public String toString() {
return "Triangulation (with " + neighbors.size() + " elements)";
}

...
//True iff the simplex is in this Triangulation.
public boolean contains(Simplex simplex) {
        return this.neighbors.containsKey(simplex);
}
...

//Report neighbor opposite the given vertex of simplex.
public Simplex neighborOpposite(Object vertex, Simplex simplex) {
        if (!simplex.contains(vertex))
        throw new IllegalArgumentException("Bad vertex; not in simplex");
        SimplexLoop: for (Iterator<Simplex> it = (neighbors.get(simplex))
                                        .iterator(); it.hasNext();) {
        Simplex s = it.next();
```

```
            for (Iterator<Boid> otherIt = simplex.iterator(); otherIt.hasNext();) {
                        Object v = otherIt.next();
                        if (v.equals(vertex))
                        continue;
                        if (!s.contains(v))
                        continue SimplexLoop;
                        }
                        return s;
}
            return null;
}
...

//Update by replacing one set of Simplices with another. Both sets of simplices must fill the same "hole" in the
Triangulation.
public void update(Set<Simplex> oldSet, Set<Simplex> newSet) {
// Collect all simplices neighboring the oldSet
            Set<Simplex> allNeighbors = new HashSet<Simplex>();
            for (Iterator<Simplex> it = oldSet.iterator(); it.hasNext();)
                        allNeighbors.addAll(neighbors.get(it.next()));
// Delete the oldSet
            for (Iterator<Simplex> it = oldSet.iterator(); it.hasNext();) {
                        Simplex simplex = it.next();
            for (Iterator<Simplex> otherIt = (neighbors.get(simplex))
                                                .iterator(); otherIt.hasNext();)
                        (neighbors.get(otherIt.next())).remove(simplex);
                        neighbors.remove(simplex);
                        allNeighbors.remove(simplex);
            }
// Include the newSet simplices as possible neighbors
            allNeighbors.addAll(newSet);
// Create entries for the simplices in the newSet
            for (Iterator<Simplex> it = newSet.iterator(); it.hasNext();)
                        neighbors.put(it.next(), new HashSet<Simplex>());
                        // Update all the neighbors info
                        for (Iterator<Simplex> it = newSet.iterator(); it.hasNext();) {
                        Simplex s1 = it.next();
for (Iterator<Simplex> otherIt = allNeighbors.iterator();
otherIt.hasNext();) {
                                    Simplex s2 = otherIt.next();
                                    if (!s1.isNeighbor(s2))
                                    continue;
                                    (neighbors.get(s1)).add(s2);
                                    (neighbors.get(s2)).add(s1);
                                    }
                        }
            }
}
```

References

- Bentley, P. J., 1999, Evolutionary design by computers, San Francisco, California, Morgan Kaufmann
- Buelow von P., 2002, Using Evolutionary Algorithms to Aid Designerd of Architectural Structures, in Creative Evolutionary systems (2002)/ Peter J. Bentley, David W. Corne, San Diego, California; London: Academic Press
- Bourke, <http://ozviz.wasp.uwa.edu.au/~pbourke/modelling_rendering/particle/>
- Caballero A., Carol I., Lopez C.M., 3D Meso-Mechanical Analysis of Concrete Specimens under Biaxial Loading, Fatigue & Fracture of Engineering Materials & Structures, Volume 30, Number 9, September 2007 , pp. 877-886(10), Blackwell Publishing
- Delanda M., 2002, 'Deleuze and the use of the Genetic Algorithm in Architecture', in Leach N., Designing for a Digital World, New York: Wiley
- Frazer J. 1996, The Dynamic Evolution of Designs, In 4D Dynamics Conference on Design and Research Methodologies for Dynamic Form, De Montfort University, Leicester, UK.
- Frazer J. 1995, An Evolutionary Architecture / John Frazer, London, Architectural Association
- Friedrich E., Derix C., Hanna H., 2007, Emergent Form from Structural Optimisation of the Voronoi Polyhedra Structure, Paper accessible at: http://eprints.ucl.ac.uk/5077/1/5077.pdf
- Holland J. H., 1975 Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence, University of Michigan Press
- Jaworski P. L., 2006, 'Using simulations and artificial life algorithms to grow elements of construction', MSc thesis, Bartlett School of Graduate Studies, UCL, accessed September 2007, http://eprints.ucl.ac.uk/archive/00002882
- Miranda Carranza P. 2001, Self-design and Ontogenetic evolution, Interactive Institute, Stockholm, Sweden, http://www.armyofclerks.net
- Mitchell M., 1996, An Introduction To Genetic Algorithms, Cambridge, Mass; London: MIT
- Paul Chew L, 1993, Guaranteed-quality mesh generation for curved surfaces. In SCG '93: Proceedings of the ninth annual symposium on Computational geometry, pages 274–280, New York, NY, USA, ACM Press.
- Resnick, Mitchel, (1997) "Reflections" from Resnick, Mitchel, Turtles, termites, and traffic jams: explorations in massively parallel microworlds pp. 119-144, Cambridge, Mass.: MIT Press
- Thompson, D. 1961, On Growth and Form, London, Cambridge University Press
- Weinstock M., Self-Organisation and the Structural Dynamics of Plants, Techniques and technologies in Morphogenetic Design, AD Architectural Design, March/April 2006
- http://en.wikipedia.org/wiki/Delaunay_triangulation, accessed July 2008.
- http://www.oasys-software.com, accessed June 2008.