Truss Design and Optimisation Using Grammatical Evolution

Doctoral Thesis

Michael Fenton, B.Sc., M.E.

Student Number: 05589584

This thesis is submitted to University College Dublin in fulfilment of the requirements for the degree of Doctor of Philosophy in Engineering

School of Civil, Structural and Environmental Engineering

Head of School:

Prof. Mark Richardson

Research Supervisors:

Dr. Ciaran McNally

Prof. Eugene O'Brien

April, 2015

Abstract

Truss optimisation in the field of Structural Engineering is an ever growing subject. The field can be divided into two main disciplines – continuum and discrete topology optimisation. Continuum topology optimisation methods represent the current state of the art in engineering design optimisation. However, in large scale civil and structural engineering projects it is currently prohibitively expensive and difficult to manufacture solid structures fully optimised using these techniques due to the current limits of both computational power and manufacturing capabilities. At present, discrete beam structure optimisation methods remain more appropriate for larger scale designs, as they allow regular elements and construction methods to be used. This leads to savings in cost and weight over traditional construction methods.

Existing discrete truss optimisation methods focus primarily on optimising global topology using a ground structure approach, with all possible node and beam locations being specified *a priori* and the algorithm selecting the most appropriate configuration from the given options. The standard method is to explore this search space, while seeking minimum cross-sectional areas for all elements in order to reduce the self-weight of the structure. In doing so, critical knowledge of section geometry and orientation is omitted. This leads to inaccurate stress calculations and structures failing to meet codes of practice. These issues can be addressed by constraining the optimisation method to only use standard construction elements. It is shown in this thesis that solutions close to the theoretical optimum can be achieved using commercially available elements.

The classical ground structure discrete optimisation method has furthermore been shown to be inherently restrictive, as it severely limits the representation space to what is explicitly defined; a larger representation space can more effectively navigate through the search space. However, a larger representation space can potentially lead to difficulties in evolving *any* fit solution. Unfit individuals must be handled carefully in order to successfully evolve any fit solution in early generations. It is therefore imperative to design the fitness function in such a way as to minimise the risk of the algorithm becoming stuck in a local optimum, before a single fit solution has been evolved.

The application of Grammatical Evolution (GE), a grammar-based form of Genetic Programming (GP), has shown that it is not only capable of generating innovative engineering designs, but that the recursive properties of formal grammars allows GE to define its own node locations for any number of nodes within a pre-specified design envelope, thereby vastly increasing its representation capabilities. Nodes are then connected via a Delaunay triangulation algorithm, leading to fully triangulated, kinematically stable structures. The net result is that discrete beam-truss structures can be optimised in a continuum manner – in a "black-box" fashion, without the need to know any information about the problem other than the design envelope. Existing discrete optimisation techniques are compared and contrasted, and notable savings in structure self-weight are demonstrated over traditional methods.

To my family - past, present, and future.

Acknowledgements

First and foremost I would like to thank my parents. Without their help and continuous support this would not have been possible. When I doubted myself they reassured me. When I wanted to quit they kept me going. I never could have achieved this without them.

I have to thank my supervisor Ciaran McNally for providing guidance over the years, and for tolerating my fantastical ideas and suggestions. He permitted me the freedom to pursue my own ideas, but always steered me towards achievable goals.

I would never have been able to finish my Master's thesis, let alone my PhD thesis if it weren't for the constant help of the NCRA team. Every single one of them has helped me in more ways than they could possibly imagine over the years, but most importantly they made me a part of a special family.

Finally, I owe a particularly special thanks to Jonathan Byrne. It never ceases to amaze me how patient and kind he was to me in the early days, when I was full of ideas but lacking the skills to implement them. He were always willing to help out, to instruct, to give feedback. It is difficult to convey how much help he provided to me over the years, but suffice it to say that I should probably name one of my kids Jonathan.

This research is based upon works supported by the Graduate Research Education Programme in Sustainable Development, jointly funded by IRCSET and IRCHSS.

Contents

List of Figures

List of Tables

Publications Arising

1. J. Byrne, M. Fenton, E. Hemberg, J. McDermott, M. O'Neill, B. Shotton and C. McNally, "Combining structural analysis and multi-objective criteria for evolutionary architectural design," in *C. Di Chio et al. (Eds.): Applications of Evolutionary Computation, Part II, LNCS 6625,* Berlin Heidelberg, Springer-Verlang, 2011, pp. 204-213.
2. J. McDermott, J. M. Swafford, M. Hemberg, C. McNally, E. Shotton, E. Hemberg, M. Fenton and M. O'Neill, "String-rewriting grammars for evolutionary architectural design," *Environment and Planning B: Planning and Design,* vol. 39, no. 4, pp. 716-731, DOI: 10.1068/b38037, 2012.
3. M. Fenton, C. McNally, J. Byrne, E. Hemberg, J. McDermott and M. O'Neill, "Automatic innovative truss design using grammatical evolution," *Automation in Construction,* vol. 39, pp. 59-69, DOI: 10.1016/j.autcon.2013.11.009, 2014.
4. J. Byrne, M. Fenton., E. Hemberg, J. McDermott, M. O'Neill, "Optimising complex pylon structures with grammatical evolution", *Information Sciences,* in press, DOI: 10.1016/j.ins.2014.03.010, 2014.

Chapter 1. Introduction

Automated problem solving is one of the most important aspects of modern computing. Human-competitive solutions to given problems with specific constraints and boundaries can be generated by computers [1]. Two prominent avenues of problem solving in computing are *gradient-based* (deterministic) and *non-gradient based* (heuristic) methods [2]. Deterministic methods are based on the principle that the correct solution can be directly calculated using specific formulae and equations. This method works for mathematically well-defined problems, whereby the globally optimal solution can be definitively computed in a finite short time. Each time the problem is calculated, the exact same solution will be generated. Heuristic methods on the other hand, do not directly compute the absolute correct solution, but balance the outright accuracy of deterministic methods with computational speed, by determining an acceptably accurate solution (though not necessarily the true optimum), in a comparatively short period of time [2]. Heuristic methods are applied either where the size or difficulty of the problem is such that deterministic methods are non-viable or where there is no known deterministic way of directly calculating the correct solution (i.e. ill-defined problems).

Population-based evolutionary search algorithms are among the most popular heuristic techniques [3]. As their name suggests, they employ populations of potential solutions to solve a particular problem. These populations are varied over time in such a way that the best solution in each population generation will tend towards an acceptable level of optimality. Due to the probabilistic nature of heuristic methods, it is generally difficult for them to find the true global optimum. As such, a different (but similarly acceptably well performing) solution may be provided each time the algorithm is run. While most mathematical optimisation problems can be solved in a deterministic manner, design is an open-ended problem, leading to the popularity of heuristic algorithms in design applications. At the very least, general engineering design can be seen as a difficult deterministic problem with many global optimums.

Heuristic algorithms may not be able to completely describe all possible solutions for a problem; if one were capable of doing so it would become a deterministic algorithm [2]. Therefore, the heuristic algorithm will only be capable of generating (and consequently searching through) *a small subset* of the total number of possible solutions to a problem. This subset of those potential solutions capable of being generated or represented by the algorithm is known as the ***representation space*** [4]. This is so called as it encapsulates all potential problem solutions that are capable of being represented by the heuristic algorithm, regardless of their actual merit as viable solutions. A larger representation space will cover a greater the percentage of the possible solutions, potentially increasing the search capabilities and effectiveness of the algorithm. Although it is desirable to have an efficient a method as possible (i.e. covering as large an area of the search space as possible), the very nature of heuristic algorithms means that a balance must be found between the size of the representation space and the complexity of the problem.

1.1. Truss Optimisation, a Brief Background

In 1904 Anthony G. M. Michell published a pioneering paper [5] in which he described the optimality criteria for the minimum weight of trusses. However, it was not until over half a century later that any further work was done in the field, with resurgence in the late 1950's rekindling interest in the area of structural optimisation [6]. Nowadays, structural optimisation is an ever expanding discipline. Topology optimisation, a subset of structural optimisation, is known as the science of optimal layout theory [7], whereby a structural material is arranged in such a way as to minimise some objective function, subject to a number of design constraints. The field of topology optimisation can be separated into two distinct strands:

1) discrete optimisation, in which the entire structure is optimised with whole beam elements, and
2) continuum optimisation, in which solid structures are assumed to be continuous and are decomposed into finite elements [8].

Continuum topology optimisation methods currently comprise of some of the most advanced structural optimisation approaches. By discretizing the entire structure into small sections, the overall structure can be assumed to be continuous, and as such, optimisation of a segment can be extrapolated to consider the whole (similar to the finite element method of structural analysis [9]). Continuum methods traditionally work by beginning with a solid block of material and then finding the most optimal arrangement of a specified percentage of that solid such that the displacement of the overall structure is minimised [8]. In continuum optimisation, the representation space is directly related to the size of the mesh used to discretize the overall solid [7].

Discrete optimisation methods on the other hand look at the entire structure as a whole, addressing individual elements (such as beams and nodal connections) rather than an optimised solid. They generally follow what is termed a "ground structure" approach, with all possible node and edge locations being specified *a priori* [8, 10]. While this allows for highly optimal beam structures to be created, the method is implicitly highly restrictive as all possible potential solutions must be known beforehand. Consequently, the representation space is extremely limited. This means that discrete optimisation methods are less effective at finding the optimal solution [7].

1.2. Research Aims & Motivation

While continuum topology optimisation is the most effective optimisation method, the technique has inherent flaws [11, 7], primarily in that computational complexity increases as the physical dimensions of the structure increase. As such, large scale structures cannot yet be easily optimised using this method. In contrast, this is an area in which ground structure optimisation excels [12], yet there are still limitations in the techniques employed. Rozvany [13] explicitly stated that globally optimal solutions *cannot* be found with the enforced layouts and low member count inherent in traditional ground structures (i.e. the representation associated with a ground structure approach is too constrained to effectively find the global optimum of the search space, regardless of the size of the representation space). Deb and Gulati [14] confirmed this by proving that even a limited increase in the variation of node placing in a ground structure approach (i.e. an increase in the representation space) led to better fitness results. As such, a maximally unconstrained discrete optimisation method, which can improve any given fitness value, would be the most effective tool.

Rozvany [7] provided analytical, theoretically correct solutions to a series of benchmark problems for continuum sections based on Michell's work. Therein, as the volume fraction of a solid approached zero, the structure of optimised shell plates tended towards that of a truss. Using this correlation, there is the possibility to derive optimal two dimensional (2D) truss topologies from stressed plate structures (i.e. a globally optimal structure can be found purely from a given design envelope [15]). The primary aim of this thesis is to investigate whether the same correlation holds true for discrete optimisation methods as for continuum methods (i.e. given only boundary conditions, can a viable discrete solution be generated?) In order for this to be successful, the unconstrained discrete optimisation method must *at the very* least match the best achieved results from a generic ground structure approach. This aim can be further decomposed into a number of research objectives.

1.2.1. Research Objectives

- *Both sizing and topology optimisation conducted in tandem.*

  Existing truss optimisation methods in GP focus either on optimisation on a global topology scale, i.e. optimisation of the structural layout over the entire structure, or optimisation of the sizing of individual members in a fixed structure. In order to emulate the continuum method, the member sizing and the overall topology of a structure must be optimised in tandem. Since sizing can be seen as a deterministic problem, the requisite sizing of members can be calculated for every individual in the evolutionary process. This method must then be compared with the simultaneous method to determine the more effective method.

- *A comparison of commercially available materials and code compliant constraints against their idealised counterparts.*

  Traditional sizing optimisation methods have some fundamental weaknesses. The standard approach employed is to explore the search space, while seeking the absolute minimum cross-sectional area for all structural elements. In doing so, critical knowledge of section geometry and orientation is omitted. It is therefore not possible to calculate fully accurate material stress limits as a constraint for structural design. Consequently structures cannot be designed to codes of practice. Another issue is that current sizing optimisation methods exceed the manufacturing precision of structural components. It is hypothesized that these issues can be addressed by constraining the optimisation method to only use readily available common construction elements. In conjunction with these construction elements, solutions must be evolved using design constraints given from codes of practice in order to observe any differences between idealised methods and code-compliant techniques.

- *An efficient penalty function for violated constraints in Grammatical Evolution*

Structural engineering optimisation will often require the designer to satisfy multiple parallel objectives, and there may be overlaps between both constraints and objectives. Understanding the interaction between these constraints and the overall individual fitness will, therefore, have a significant impact on the quality of the designs produced. As such, a key challenge for designers when using evolutionary approaches is to find an accurate metric that will allow the designer to judge individual constraints and to transform the performance of the individual (relative to those constraints) into a single coherent value for use by the fitness function. The number of these constraints may have a positive or negative effect on the evolutionary process. Constraints can either be seen as a necessary design component or an instrument to improve search. The most efficient way to relate constraint violations to the overall fitness of the individual in the context of Grammatical Evolution (GE) must be ascertained.

- *Creation of an unconstrained grammar representation.*

Traditional ground structure optimisation methods use fixed node locations and vary the connectivity between nodes to generate different structures. Even a slight variation in node locations can improve fitness results in ground structure optimisation [14]. As such, more variation (i.e. more design freedom leading to greater representation) allows a heuristic search process to get closer to the true global optimum. The degree of variation allowed before the representation space becomes too large and the search breaks down must be examined. Various methods of creating sufficient design variation using a formal grammar representation must be identified, along with the amount of genetic information required in order to generate this variation.

- *Evolution of a viable solution given only boundary conditions and constraints.*

Continuum optimisation methods only require a design envelope and boundary conditions to generate a solution. Discrete ground structure approaches, however, require all node locations and connections to be specified *a priori* to generate a solution. The recursive properties of formal grammars can be used to allow the algorithm to define its own solutions within a pre-specified design envelope. In such a way it is hypothesized that a discrete method could operate in a continuum fashion. Viable solutions must be evolved if no information is given other than the bare minimum.

1.3. Thesis Summary

Chapter 2 contains a summary of related research in the areas of evolutionary computation, evolutionary optimisation, and engineering optimisation. Based on a review of the literature, a number of key research areas have been identified. These aspects incorporate both structural engineering problems and evolutionary computation problems. Identified areas are as follows:

- The majority of reviewed discrete optimisation literature uses unrealistic assumptions about material properties and physical structural layout. There is a tendency in the literature to focus on minute improvements to optimisation methodologies, while fundamental engineering principles are overlooked;
- Traditional ground structure discrete optimisation approaches are limited in their representation capabilities and, thus, cannot cover the search space as effectively as possible;
- Simultaneous optimisation of member sizing, structural shape, and structure topology produces the best theoretical optimisation results but is difficult to represent with a constrained ground structure representation and even more difficult to achieve effectively;
- A larger the representation space theoretically has a better chance of success, with a better overall quality of solutions. However, a larger representation space results in less fit individuals; the management of unfit individuals via the handling of constraints then becomes crucial to improve evolutionary performance.

Chapter 3 details design generation and the evolutionary process as used in this study. An in-depth description of the Grammatical Evolution method is provided, and explanations of every step in the process are given.

Chapter 4 demonstrates that the majority of reviewed discrete optimisation techniques found in the literature use unrealistic assumptions about material properties and physical structural layout. Experiments show that realistic construction materials, in conjunction with code-compliant design constraints, cannot produce the same results when applied to the same problem. These experiments conclude that the heavily optimised results of the literature give a false impression of the efficiency of the algorithms in question. A number of experiments demonstrate the ability of GE to evolve and optimise two datasets simultaneously, namely the structural topology and member sizing of a structure. In order to achieve this, a new evolutionary method was developed, which utilized two separate chromosomes. The method is shown to be both powerful and reliable in evolving fit solutions, thereby addressing another of the key research objectives.

Chapter 5 aims to ascertain the most appropriate method of handling individuals with violated constraints. It begins with a description of fitness landscapes and the terms search space, representation space, and fitness space. Variation in the fitness landscape can occur even if the representation space remains constant. This variation is consistent with increasing problem difficulty. In order to ascertain the most appropriate method of handling constraint violations, different constraint handling methods are applied to problems of increasing difficulty. Variation in these fitness landscapes is observed through a number of experiments, which demonstrate the effectiveness of varying methods of penalty function applications. Appropriate methods for handling both single and multiple failed constraints are discussed, and finally, recommendations are made that as much information as possible on the performance of all constraints for any infeasible individual be included in the fitness function.

Chapter 6 aims to produce an unconstrained grammatical representation for discrete truss structures. Previous methods from the literature are based on the ground structure method whereby node locations are pre-specified. This chapter hypothesizes that the evolution of node locations, rather than of node connections, could lead to improved fitness results. The hypothesis of generating discrete structures through placement and variation of nodes rather than the variation in connections of pre-existing nodes is introduced. A Delaunay triangulation algorithm is used to connect evolved nodes, allowing for the generation of repeatable, fully triangulated, kinematically stable structures. The use of two chromosomes allows for the simultaneous optimisation of structural shape, structural topology, and individual member sizing. These structures are then evolved to find minimum weight arrangements for benchmark problems.

Chapter 7 contains an overview of the work done in this thesis and the overall contributions. Conclusions and suggestions for future work are detailed.

Appendix A contains grammars and phenotypes created for this body of work.

Chapter 2. Literature Review

2.1. Introduction

Alan Turing [16] first proposed the idea of machine learning, whereby computers could modify their behavior over time to suit their environment or environments. His theory was that a "child machine" would be produced with little knowledge of its environment, but that over time and through interaction with its environment could gain sufficient knowledge to interact in a meaningful manner. The end result of this would be a machine, which could solve a problem without explicitly being told how to solve it. The term "Turing Test" was coined to demonstrate a simple concept defined by Turing: consider two unknown entities, one human and one machine, in communication with a blind observer whose sole objective is to determine which is which. By Turing's reasoning, the machine would be deemed to be intelligent, if it passed as human more than 50% of the time. Arthur Samuel [17] extended the description of artificial (machine) intelligence:

"to exhibit behaviour, which if done by humans,
would be assumed to involve the use of intelligence."

Combining these two concepts would produce a machine which could, over time, interact with its environment to change its behavior and produce results, which would (from a black box perspective) be deemed to have been produced by an intelligent entity. This is the driving force behind the concept of the evolutionary computation method.

2.2. Evolutionary Computation

Evolutionary Computation (EC) is a field of computing techniques based on the evolutionary principle of natural selection [18], wherein populations of individuals are continually improved within their environment based on the theory of survival of the fittest, taking strong cues from biological terminology [19]. A problem is addressed using populations of individuals, each representing a potential solution to the problem. These populations are then slowly evolved over a number of generations by interbreeding and varying individual solutions to produce new child solutions that form the basis of each subsequent generation. At each generation, an objective function (usually known as the fitness function) determines the suitability of each individual for its designed environment (i.e. its ability to solve the problem at hand) and assigns it a value by which it can be judged against its peers (usually known as a fitness) to determine the best solutions in the generation. The most suitable solutions in a generation (known as the fittest solutions) then have a higher probability of passing on information to subsequent generations. After a certain number of generations, or after some stopping criterion is met, the evolutionary process is terminated and the best overall individual is presented as the evolved solution [20, 3].

Numerous EC techniques have been used in a wide variety of applications. In the field of the arts, Kaliakatsos-Papakostas et al. [21] used an interactive genetic algorithm to evolve rhythmic patterns in the form of a self-programming drum machine. McDermott and O'Reilly [22] used a graph-based representation to map Musical Instrument Digital Interface (MIDI) paths for evolutionary generative music in both interactive and non-interactive scenarios, while Nicolau and Costelloe [23] generated award-winning three-dimensional (3D) fractal images (Figure 2.1).

EC techniques have been used in computer game generation [24, 25], and numerous engineering applications from circuit design [26] to space antenna design successfully deployed by NASA [27]. Financial trading algorithms have been evolved [28], as well as architectural and structural engineering designs [29, 12].

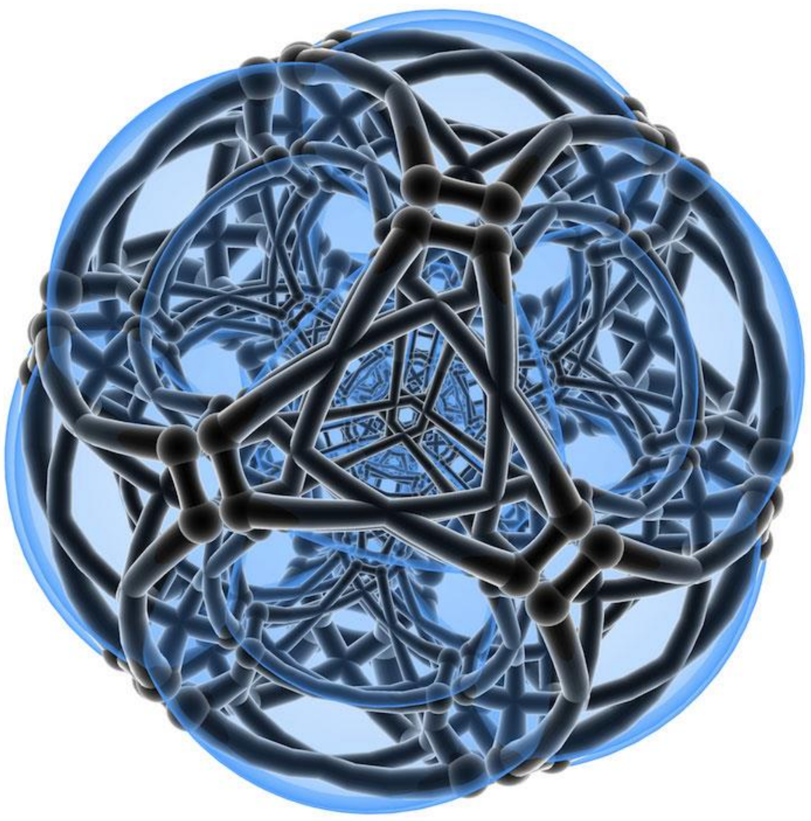


Figure 2.1: Fractal image generated using evolutionary techniques [23]

2.2.1. Evolutionary Algorithms

Evolutionary Algorithms (EAs) are a sub-field of EC techniques; whereas EC is seen as the overall field of evolutionary population-based computing, EAs represent a broad range of actual applications of EC techniques. The four most popular avenues of EA are Evolutionary Programming [30], Evolutionary Strategies [31, 32], Genetic Algorithms [19], and more recently Genetic Programming [33].

Since the evolutionary method used in this study, Grammatical Evolution, is essentially a derivative of the latter two methods (as shown in Figure 2.2), a short review of both of these methods is presented.

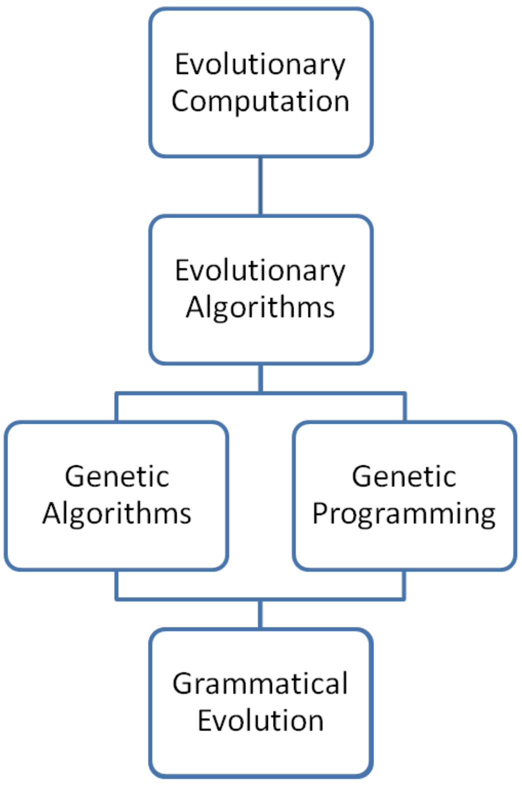


Figure 2.2: Evolutionary Computation hierarchy

2.2.2. Genetic Algorithms

Genetic Algorithms (GAs) are a form of EA that use a binary integer string as a genetic representation for an individual solution. The idea of representing an individual with the use of a genotype (or chromosome) was first proposed by Holland [19] and implemented by Goldberg [34]. A chromosome typically consists of an array or string of genetic information, which is then mapped to an observable set of characteristics of the solution, known as the phenotype. GAs traditionally represent their chromosomes as a fixed-length string of bits (consisting of either 0 or 1), and map directly from genotype to phenotype without the use of an intermediate interpreter. As such a GA representation is very effective at parameter tuning and in situations where direct variance of values is required [34]. A single codon (gene) on the chromosome can either be represented as a single bit or as a fixed number of bits from the chromosome.

GAs use evolutionary selection and replacement parameters to augment or replace populations (depending on the replacement strategy used) in a generational manner in order to improve the overall fittest solution, as shown in Figure 2.3.

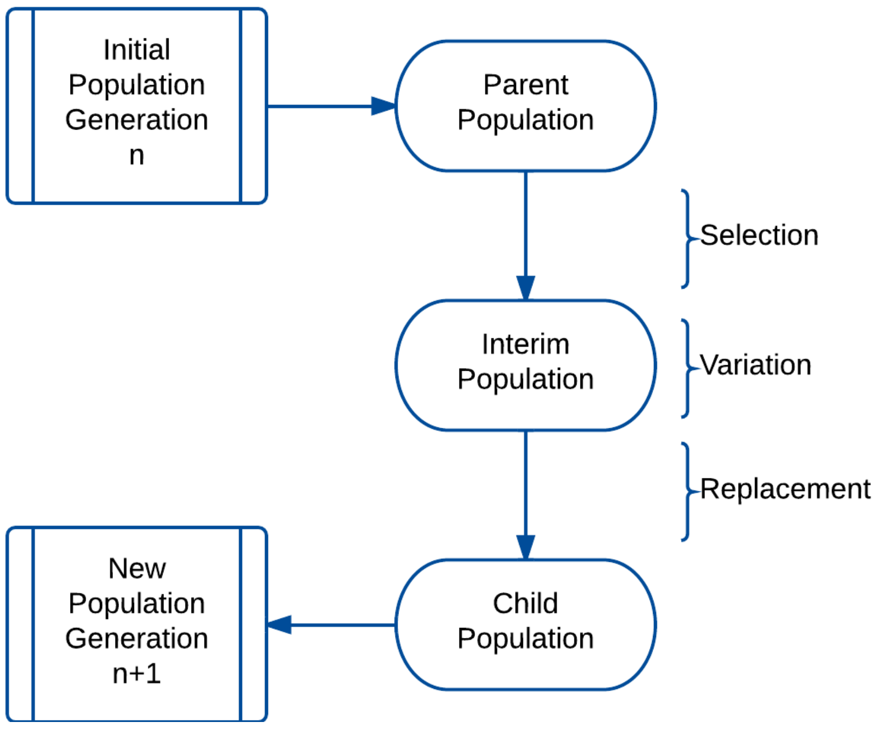


Figure 2.3: The creation of each new generation is accomplished through selection, variation, and replacement of the population of the previous generation

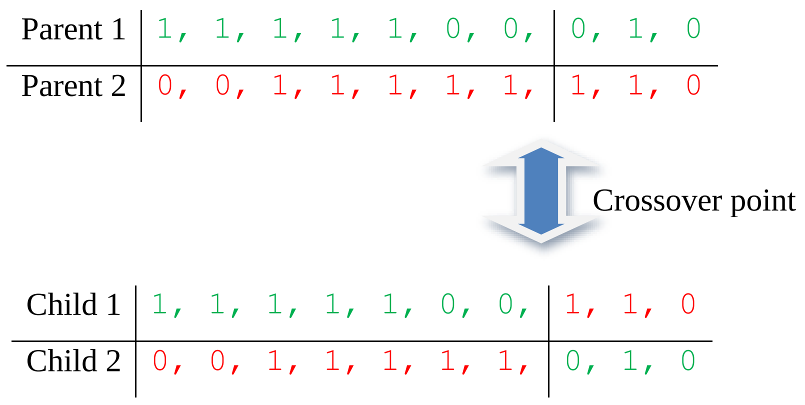
At each generational step, a pool of parents is chosen from the parent population based on the fitness values of each individual using a selection mechanism, such that the fittest individuals will have a greater probability of passing on genetic material to subsequent generations. A number of selection mechanisms are available in EAs [35], including tournament selection, linear ranked selection, and fitness proportional selection:

- **Tournament Selection**
  A small subset of the overall population is selected at random from the initial population, and the fittest individual within the subset is added to the parent pool. Once the parent pool reaches the same size as the initial population, selection terminates.
- **Linear Ranked Selection**
  The population is ranked in order of fitness, and selection of an individual is random with a weighted probability based on the individual's rank [36].
- **Fitness Proportional Selection** (also known as **Roulette Wheel Selection**)
  Each individual in a population is given a probability of selection equal to their normalised fitness value. The better the fitness, the better the probability of selection [19].

Since the probability of selection of each individual with these methods is based on its fitness value, it is possible for the same individual to be selected multiple times for inclusion in the parent population, while other individuals may never be selected regardless of their fitness. So-called “fair” selection mechanisms also exist such as Fair Tournament Selection, where each individual is guaranteed a chance to participate in at least one tournament [34]. Once the interim population is fully populated via the selection mechanism, a child population is created which will form the basis of the next generation. This child population is generated by variation operators, which are performed on individuals from the interim population. The most prominent such methods are mutation and crossover:

- Mutation
  Information from a single individual is randomly mutated with a probability P_{mut} to create a new individual.
- Crossover (also known as recombination)
  A combination of two parents produces a pair of children with qualities of both parents, with a probability P_{cross}.

Crossover of two parent chromosomes is usually done in a single-point manner, where a single crossover point on both chromosomes is randomly selected and then all subsequent genetic information is swapped between individuals, as shown in Figure 2.4:



Parent 1	1, 1, 1, 1, 1, 0, 0,	0, 1, 0
Parent 2	0, 0, 1, 1, 1, 1, 1,	1, 1, 0

Child 1	1, 1, 1, 1, 1, 0, 0,	1, 1, 0
Child 2	0, 0, 1, 1, 1, 1, 1,	0, 1, 0

Figure 2.4: Single-point crossover of two parent GA chromosomes

Mutation then occurs on those children in a “bit-flip” fashion by randomly changing codons on the chromosome between 0 and 1 (as shown in Figure 2.5).

Original	1, 1, 1, 1, 1, 0, 0, 0, 1, 0
Mutated	1, 1, 0, 1, 1, 0, 0, 1, 1, 0

Figure 2.5: Bit-flip mutation of GA chromosome

Canonical crossover is generally applied first on randomly selected pairs of parents, and then mutation is applied to the newly formed child population of resulting individuals [34]. It is advisable for P_{cross} to be set at a high value, generally between 70% and 90% (i.e. a chosen pair of individuals will have between a 70% to 90% chance that crossover will be performed on them, otherwise they remain unchanged), while P_{mut} is more usually given a very low probability of around 1% (i.e. each gene on the chromosome has a 1% chance of mutating to a new value, otherwise it remains unchanged) [37]. In each case, if a randomly selected probability value falls within the set operator probability (P_{cross} or P_{mut}) then that operation is performed, otherwise the individual is conveyed to the child population unchanged from the parent population.

The final step in the creation of a new generation is replacement. There are two main options [3]:

- Generational Replacement: The entire previous generation is replaced with the newly generated child population.
- Steady State Replacement: Only a portion of the previous generation is replaced with the child population.

GAs typically feature random initialisation of the first generation, tournament selection for generation of each parent population, and mutation and crossover as described above. EA search operators such as mutation and crossover act directly onor the chromosome itself, which has a direct impact on the phenotype of the individual [38].

Gupta and Ghafir [39] emphasised the need to maintain diversity in the population in order to avoid getting stuck in a local optimum (whereby the search process converges on a solution which is not the global optimum). This is done by carefully controlling selection and replacement mechanisms to maintain a diverse genetic pool. Variations on traditional GA methods exist. For example, Li et al. [40] used Diploid (two chromosomes) and Triploid (three chromosomes) GA representations to reach a balance between convergence and diversity in the population. In contrast, Cavill *et al.* [41] used a GA with both multiple chromosomes and variable length chromosomes to tackle the Onemax problem – a simple problem designed to test whether an algorithm can handle multiple variables in parallel.

2.2.3. Genetic Programming

Genetic Programming (GP) uses a tree-based genetic representation to modify and combine modular programs and functions to solve a combinatorial task [20]. It differs from a traditional string-based GA by using a tree-based mapping system. This tree-based system contains terminals (leaves) and sub-trees (branches) [42]. Terminals are final values that represent a fixed part of the overall solution, whereas branches represent some operator or condition to be applied to those connecting terminals and sub-trees.

An example of tree-based derivation for the function `max((x+(y*x)), (y/x))` is shown in Figure 2.6. The problem consists of an overall function `max()`, sub-trees in the form of various mathematical operators `[+ - / *]`, and terminals in the form of `x` and `y`. By combining different amounts and types of terminals and branches, this tree-based system allows for highly complex programs to be evolved [33].

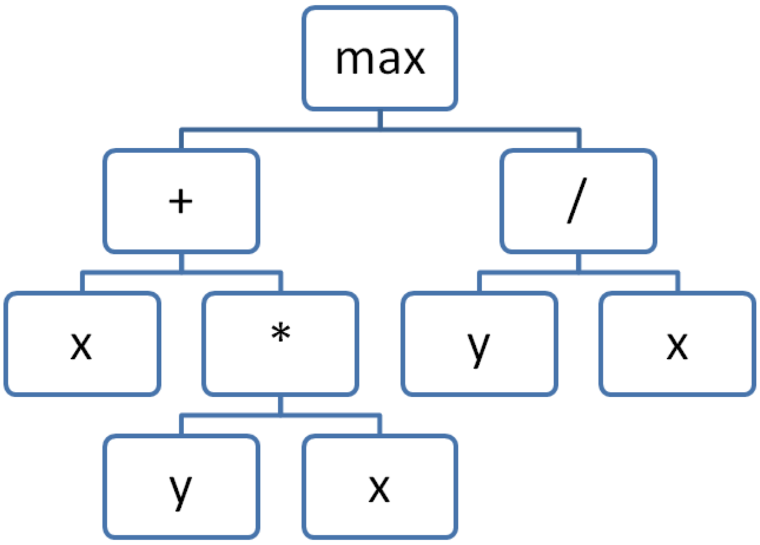


Figure 2.6: GP tree-based derivation for the function `max((x+(y*x)), (y/x))`

GP also differs from traditional EAs and GAs in its use of recombination (crossover) and mutation operators [42]. In a crossover with a GA, two parent chromosomes are split at a single crossover point, and the heads and tails are swapped between parents to produce two child chromosomes. GP on the other hand utilizes “sub-tree” crossover (i.e. a crossover branch point is selected on the trees of two parents and all information *below* that branch (including the branch itself) on one parent is combined with all information *above* that branch (including the branch itself) on the other parent). Mutation in GP is also most commonly done in a sub-tree manner, with a random point on the tree being replaced by a new randomly generated tree (which may have a greater or lesser depth than the portion of tree it is replacing) [33].

Banzhaf [20] asserted that any problem that can be solved using a Neural Network can be solved using GP. Cavill *et al.* [43] took inspiration from biological representations in nature (where multiple chromosomes are common) and used multiple chromosomes for evolving a single objective, taking cues from both GP and Grammatical Evolution. However, the traditional implementation of multi-ploid systems uses separate chromosomes to encode for the same information (with only one chromosome being expressed at a time), rather than multiple concurrent chromosomes representing different parts of the solution [43].

2.2.4. Grammatical Evolution

Grammatical Evolution (GE) was first introduced in 1998 by Ryan *et al.* [44] as a hybrid of a GA and GP, in which a variable-length integer array genotype generates an executable phenotype via a GP-style tree-based mapping. The mapping is performed with the use of a formal grammar, written in the Backus-Naur Form (BNF) [45]. This intermediate mapping step is akin to the role of amino acids in biological terminology [46], and means that the genotype does not necessarily have to map directly to the phenotype (as is the case with traditional GAs). The grammar generates an executable program from a genotype in the same sense that amino acids create proteins from DNA/RNA. Execution of the program then creates a phenotype (the individual), similar to how a protein generates an observable phenotypic effect in biological organisms. This relationship is demonstrated in Figure 2.7.

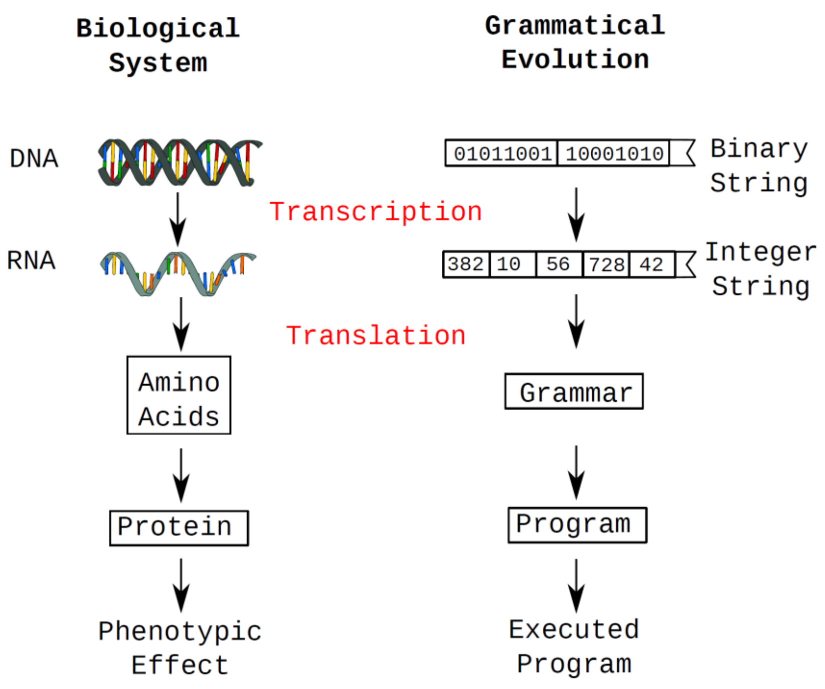


Figure 2.7: Comparison between the GE system and that of genetic biology [47, 46]

A sample grammar, capable of producing the same output as the GP example from Figure 2.6, is shown in Figure 2.8. This grammar begins with an initial production rule `<init>`, which has production choices of `max(<exp>, <exp>)` and `min(<exp>, <exp>)`. Within these production choices are non-terminals `<exp>`. These non-terminals then map to further production rules, which can become either more non-terminals (encased in angle brackets < >) or terminals (finite values or operations with no angle brackets). A more thorough explanation of GE's use of grammars is detailed in Section 3.5.

```
<init> ::= max(<exp>, <exp>) | min(<exp>,
<exp>)
<exp> ::= <exp><op><exp> | <var>
<var> ::= x | y
<op> ::= + | - | / | *
```

Figure 2.8: Basic BNF grammar example, capable of producing the same output as the GP example from Figure 2.6

This grammar-based mapping system has numerous advantages over a regular GA [48, 46]:

- The use of a formal BNF grammar in the genotype to phenotype mapping process means that the structure of the solution can be encoded in any language;
- Large amounts of phenotypic information can be generated from the grammar based on a very small amount of genotypic input;
- Bias can be added to the grammar such that it is more predisposed towards generating particular solutions;
- A variable amount of the genome is used, leading to a more diverse range of potential solutions;
- Multiple genotypes can potentially map to the same phenotype, which permits neutral mutation (a mutation that has no immediate effect on the fitness, also known as Genetic Drift). Kimura [49] argued that neutral mutation increases the effect of mutation. This is especially evident once crossover occurs on a previously neutrally mutated individual; the individual will still have the same probability of selection based on the selection process, but after crossover occurs a portion of the newly formed chromosome may now be mapped in a different manner to a non-neutrally mutated chromosome, thereby creating an extra effect one generation down the line.

GE has been used for a variety of applications, from evolutionary art [23], to video game design [24], predicting bankruptcy in financial trading environments [28] and architecture & engineering applications such as shelter design [29] and electricity pylon design [50]. A thorough explanation of the Grammatical Evolution process used in this thesis is presented in Chapter 3.

2.3. Structural Optimisation

The field of structural optimisation was established at the start of the 20th century when Anthony G. M. Michell published his first theory on the optimisation of minimum weight trusses [5]. The basic premise for structural optimisation is the minimisation of an objective function (usually either the self-weight of the structure or its compliance/strain energy/deflection) over a design space subject to some design constraints. Michell’s classical solution held that a fully optimised beam structure would have:

- Stresses in all members equal to the allowable stress limits of either tension or compression for that member

- Strains and deflections for all members and nodes that do not exceed their limits for all members and nodes
- A total structural weight less than or equal to that of any other valid structure for the same design conditions

A (purely theoretical) Michell structure has an infinite number of members, each with an infinitely small cross-sectional area and each intersecting another at right angles, as shown in Figure 2.9.

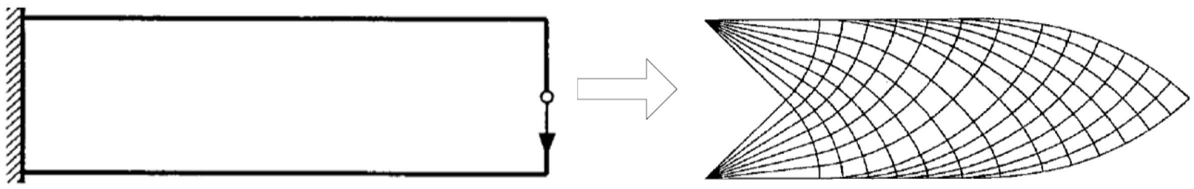


Figure 2.9: Michell truss solution for a simple cantilever [51]

It was not until over 50 years later with the advent of computer technology that Michell's work began to gain recognition. In the late 1950's, the aviation industry began research into the area of structural design, beginning with the British Aeronautics Research Council [52] and the College of Aeronautics in Cranfield [6, 53]. With the subsequent rapid advance of computational power, structural optimisation techniques have become ever more accessible in everyday engineering design. The introduction of new computational techniques has heralded engineering applications ranging from analog circuit design [54] to the design of structures such as space antennae [27], shelters [55], and bridges [56].

Kicinger *et al.* [12] conducted a survey on the applications of evolutionary computation in structural engineering design which found that (as of 2005) the vast majority of published structural optimisation research focussed purely on optimality of the structures. Apart from providing a generalised overview of the field, they noted that the most important aspects of the evolutionary engineering design process are:

i. appropriate representation of the engineering system itself;
ii. finding a suitable evaluation function.

The first of these is trivial; appropriate representation of the engineering system is possible by using the relevant design codes of practice [57, 58, 59, 60]. In the case of structural design this entails creating boundary conditions (supports and loading), material limits (usually expressed as stress/strain, buckling, etc.) and design limits (deflection). The use of the Finite Element method of structural analysis [9] as a fitness function has been proven accurate [56, 61], and it enables the EC method to assess and evaluate individuals based on the results of a finite element analysis. Furthermore, long-standing concerns about the computational efficiency of such analysis methods [12] are continually being abated by technological improvements, and the inherent parallelism of population-based optimisation techniques naturally lends itself to large computing arrays.

Evolutionary algorithms, in particular, are particularly well suited to solving structural optimisation problems as, aside from their relative ease of implementation, their heuristic nature provides an efficient method of iterating over a wide search space with many local optima in search of the global optimum [62, 63]. As such, they are prime methods for finding solutions to problems where either there are no mathematical assumptions (i.e. the structure of the solution is not known) [64], or more frequently where classical methods such as deterministic calculation have failed [65]. In the structural optimisation literature, common design challenges (known as benchmarks) are attempted by researchers proposing new optimisation methods. Perhaps a legacy of Michell's pioneering work is that these design challenges are most prevalent in the area of truss optimisation [12]. There are two distinct areas in structural design and optimisation: Continuum and Discrete design (Figure 2.10). Definitions and reviews of the literature from both areas are presented hereafter.

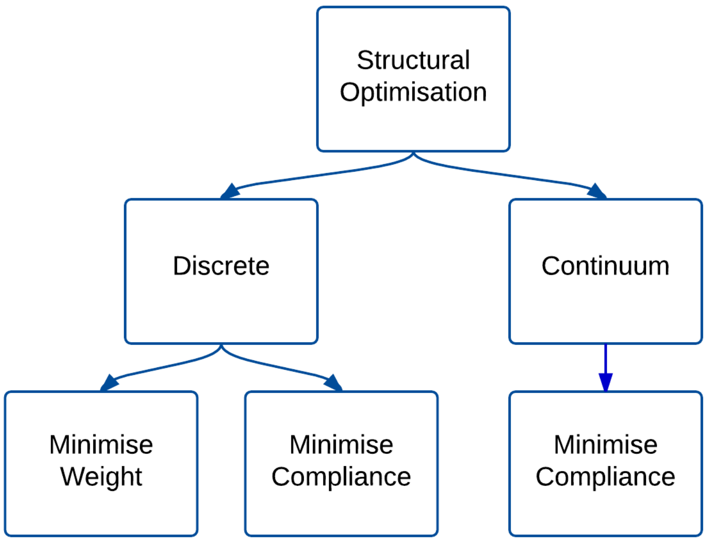


Figure 2.10: Structural optimisation flowchart

2.3.1. Discrete optimisation in Evolutionary Computation

The discrete design approach is similar to traditional beam-truss design in that it looks at the design of the complete structure, with particular focus on element connectivity. This method lends itself especially well to truss design, where appropriate connectivity of the members is paramount. Discrete optimisation looks at optimisation of the entire structure as a whole, with a general goal of minimising the self-weight of the structure. Optimal design of discrete truss structures can be generally divided into three sections [12]:

1. Topology optimisation, which deals with the connectivity of nodes (Figure 2.11).

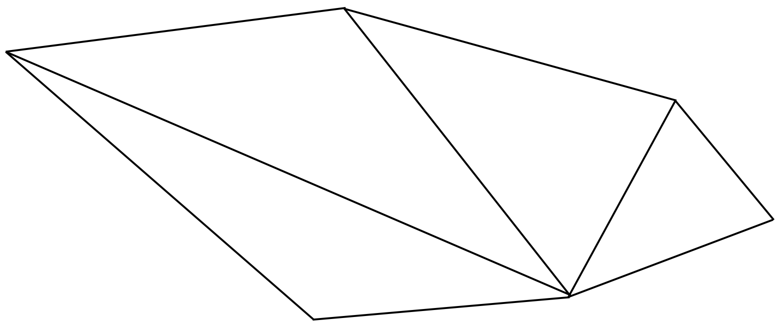


Figure 2.11: Topology connection of nodes – optimising the connections between nodes (node location independent)

2. Shape optimisation, which deals with the locations of nodes (Figure 2.12).

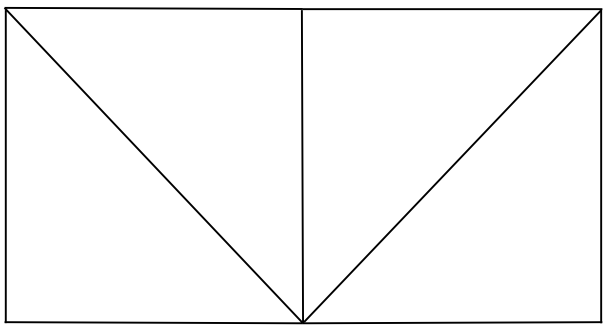


Figure 2.12: Shape optimisation of nodes – optimising the locations of individual nodes

3. Sizing optimisation, which concerns the sizing of individual members (i.e. the required area of connections between nodes, Figure 2.13).

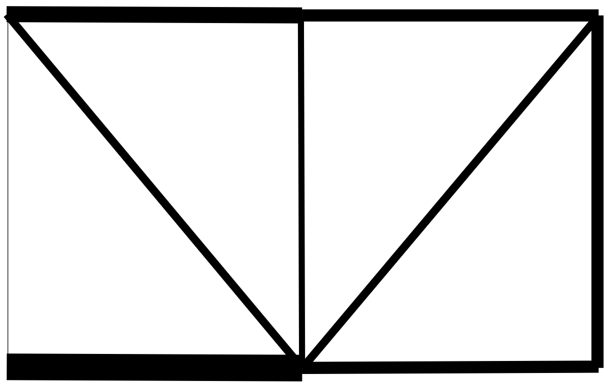


Figure 2.13: Sizing optimisation of node connections

Optimisation of two objectives (i.e. sizing and topology in tandem) represents a doubling of the complexity and, therefore, the difficulty of the problem. Optimisation of all three – sizing, shape, and topology – represents an exponential increase in the inherent difficulty of the problem.

Discrete optimisation of truss structures is most popularly achieved using the ground-structure optimisation method proposed by Dorn *et al.* [10]. In this method all selectable node and edge locations are pre-determined (thus, precluding shape optimisation), with the algorithm searching through different permutations and combinations of beam layouts (topology optimisation) and sizes (sizing optimisation) to find an optimal design. The archetypal ground structure will consist of a fixed layout of nodes (defining the overall shape of the structure), with all possible connections between those nodes being outlined. An example of a typical ground structure formulation is shown in Figure 2.14 a, along with sample potential derivable structures (Figure 2.14 b and c).

Discrete truss optimisation can generally be divided into two areas: minimisation of structural self-weight given a specified compliance limit, and minimisation of compliance given a specified volume/weight (Figure 2.10).

2.3.1.1. Weight Minimisation in Discrete Optimisation

The vast majority of discrete truss optimisation methods deal with minimisation of structural self-weight [12, 7]. The problems addressed will typically feature limits on displacement and material stresses.

Genetic Algorithms in particular have enjoyed popularity in the area of ground structure weight optimisation due to the potential easy mapping relationship between the chromosome and the design space. This is usually achieved with the use of a “topological bit” in the chromosome; a simple binary value which indicates the presence or absence of a particular member.

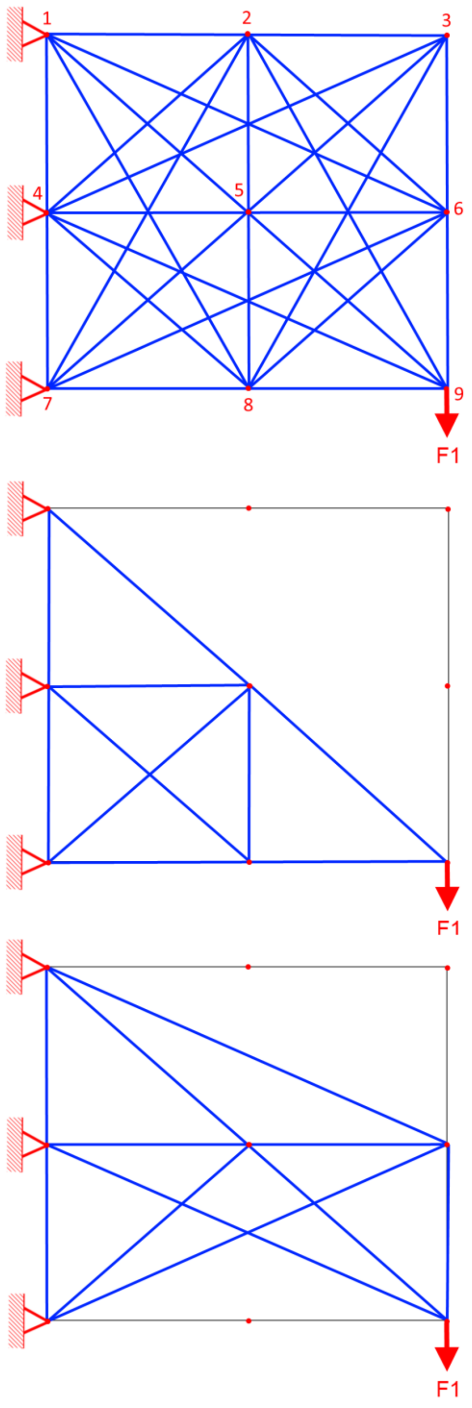


Figure 2.14: a) Typical ground structure formulation (9 nodes, 28 edges), with example derivative ground structures shown in b) and c)

Genetic Algorithms in Discrete Truss Optimisation

GAs have been used extensively in discrete topology optimisation in design (TOD) to evolve trusses, and numerous approaches have been identified.

Murawski *et al.* [66] produced a GA-based system for evolving steel structures in tall buildings (between 16 and 36 stories), named "Inventor 2000". Their system minimised the total weight of the structure subjected to varying amounts of wind loading and analysed the structures using a modified version of Structural Optimisation Design and Analysis (SODA) [67].

Kicinger et al. [68] conducted evolutionary computation parameter sweeps in the search for optimal settings in the design of tall buildings subjected to wind loading, using an update of their proprietary software Inventor 2000 named "Inventor 2001" which employed a fixed length GA with integer genes. They were able to design complex and diverse morphologies using a variety of interchangeable genetically encoded "building blocks". Their findings corroborated the original recommendations of Michell [5] and those of traditional design practice [58, 57], in that symmetrical designs prove the most efficient. They recommended larger population sizes and a high number of generations (*"up to a few thousand"*), however their proposed crossover and mutation rates (of up to 50% and 10% respectively) differ considerably from those of classical evolutionary algorithms, as discussed in Section 2.2.1 [37].

Hajela and Lee [69] implemented a two-stage approach to evolving discrete ground structures, with overall structure topologies and individual edge sizings being evolved separately with the aim of minimising weight. They first ensured that the population was seeded with kinematically stable structures, employing a bit-string Boolean indicator for the presence or absence of structural members. These stable individuals were then used as the seed for the second stage, wherein the cross-sectional areas of these members were subsequently evolved in accordance with stress, deflection, and buckling limits.

Ohsaki [63] optimised ground structures, but with the objective function of minimising the overall cost of the structure based on the number of members and nodes. This was achieved by removal of both members and nodes – the former either by a topological bit on the chromosome or once the cross-sectional area reached 0, and the latter only if the cross sectional areas of all attached members are 0. In such a manner it can be said that the author optimised both sizing and topology of truss structures, though the lack of movement of the node locations (or better yet the possibility of adding new nodes) constrained the method.

Deb and Gulati [14] used a simple binary topological bit to select between elements, but added genetic information on member sizes and node locations to the chromosome. They employed the ground structure approach with set nodes and node connections, but an important finding of their work was that an allowance for slight variations in the locations of nodes led to improved fitness results. They divided nodes within truss design into two categories:

1. Basic Nodes: Nodes with actions, such as loads, reactions, or fixings, which are essential to the structure
2. Optional Nodes: All other nodes, such as nodes connecting basic nodes

In their opinion, optimisation of truss structures involves finding the locations of non-essential nodes, the subsequent connectivity of the overall structure, and the sizes of all members such that constraints are satisfied and objectives are minimised.

Ruiyi *et al.* [70] used a GA-based technique, and attempted to improve the efficiency of the algorithm with a tracker for each individual based on their genetic chromosome. A list of all chromosomes was kept, and each newly generated chromosome is not evaluated if it already exists in the list. They reasoned that computational complexity could be reduced by stopping the process from evaluating individuals which had already been evaluated previously in that same run. This is similar in concept to Glover and Laguna’s Tabu Search method [71], though different in its implementation (a true Tabu search prevents duplicate solutions through semantic comparison rather than genetic comparison).

While the ground structure approach is the most widely used, it is not the only available discrete optimisation option. Bohnenberger *et al.* [72] used a hybrid of GAs, deterministic calculation, and evolutionary strategies to evolve minimum weight for pylon tower legs. Their approach was to maximise stress levels in each member such that the structure was operating at, or near to, its limit.

Kawamura *et al.* [73] used a modified GA to evolve stable structures by assembling combinations of triangles and demonstrated the use of their technique in both two and three dimensions. Their approach built structures in an additive fashion by extending combinations of triangles from the support conditions until all support and loading criteria were met. This triangulation method is of great interest as not only are all structures guaranteed to be both co-planar and kinematically stable, but it was also demonstrated to be extremely material efficient, with no unnecessary members being generated. This is in marked contrast to the majority of ground structure methods, where overlapping members can be used to indicate thicker member sizes [12].

Particle Swarm Optimisation

Particle Swarm Optimisation (PSO) is a form of Natural Computing algorithm in which individuals in a population are represented by a particle consisting of a location co-ordinate and a vector inside the search space. Each particle moves through the search space by updating its location at each step in accordance with its vector (as shown in Table 2.1). The particle is attracted to both the positions of the local best particle (i.e. the neighbouring particle with the best fitness value) and the historical global optimum (the position of the particle with the overall best fitness) [74].

Table 2.1: Sample PSO particles, showing both original and updated locations

Particle	Original Location	Vector	Updated Location
a	[1, 5, 4]	[0, 1, -1]	[1, 6, 3]
b	[3, 3, 7]	[2, 0, 2]	[5, 3, 9]

This swarm mentality mimics behaviour of large groups of animals, which appear to co-operate with one another towards a common goal (i.e. evasion of predators), despite their lack of apparent communication [65]. PSO algorithms are generally favoured, as they have fewer tuneable parameters over traditional GA methods and are deemed easier to implement [64].

PSO algorithms have proven particularly efficient in solving dynamic and noisy problems. For example, Li *et al.* [64] showed that a combination of a PSO and the so-called “harmony search” scheme [75, 76] improved the overall fitness and decreased the convergence rates over traditional PSOs in the optimisation of discrete truss structures. Kaveh and Talatahari [65] improved this by combining the method proposed by Li *et al.* with an ant colony strategy for an even more efficient hybridised approach. Their method used a PSO for global optimisation and then further modified the position of individual particles using an ant colony optimisation approach. Kaveh and Talatahari [65] noted that a hybrid approach of two methods produced better results with both notably fewer iterations and evaluations and a higher rate of convergence.

Luh and Lin [77] used PSOs for full topological optimisation of discrete ground structures. Their work described a two-stage approach, firstly optimising the topology of the structure (in terms of node locations and connectivity) and then optimising the sizing of the elements, along with further node location adjustments once the topology was selected. They reasoned that a single-stage optimisation process is too complex, with the search space too large to effectively optimise a solution. They employed PSO as a "quick fix"; it is freely admitted among the PSO literature that its performance may not be competitive with other evolutionary algorithms, as the number of iterations and generations increases [64, 77]. One interesting observation that Luh and Lin made is that the larger the physical size of the design, the greater the number of potentially acceptable solutions that exist (i.e. highly optimal solutions, though not necessarily the global optimum solutions).

Multiple Objective Optimisation

Optimisation of multiple objectives using evolutionary algorithms is entirely possible but is done in a different manner to optimisation of a single objective (where simple minimization or maximization of a single objective function is all that is required). When multiple objectives are present in engineering optimisation, they are often in competition with one another (depending on how said objectives are defined). As a result, an increase in the fitness of one objective may have a corresponding decrease in the fitness of another [62]. This correlation produces a series of non-dominated (Pareto) fronts, where all individuals on the same front are treated as equally fit to one another. The optimisation of these fronts is the concern of multiple objective optimisation [56].

Marler and Arora [78] conducted a survey of multi-objective methods for engineering optimisation and concluded that no single approach is necessarily superior to any other, but that the appropriate method to use must be determined from the problem definition. They echo both Jones's terminology in defining different parts of the search process as individual spaces [79] and Michalewicz's distinction between fit and unfit individuals within these spaces [80]. They note that even with an unconstrained problem only certain fit individuals within the representation space will actually be attainable in multiobjective problems as the number of objectives increases. This is due to increasing dimensionality of the Pareto fronts, which subsequently leads to difficulties in selecting optimal solutions.

Recent methods [62] have successfully used combinations of evolutionary algorithms and approximate gradients in truss topology optimisation in minimising both the self-weight and the compliance/strain energy of trusses. Pholdee & Bureerat [62] noted that while pure evolutionary algorithms have excellent search skills for multiple objective optimisation, they tend to have difficulty in very large search cases. In a similar manner to the findings of Kaveh and Talatahari, [65], Pholdee & Bureerat found that improvements over original multi-objective evolutionary algorithms [81, 82, 83] can be achieved by combining an EA method with other methods such as local search or the approximate gradient method. Byrne *et al.* successfully used multiple objective optimisation in both the optimisation of bridge and truss structures [56] and electricity pylons [50] by combining Grammatical Evolution with the Non-dominated Sorting Genetic Algorithm (NSGA) II algorithm [84].

Simultaneous Discrete Optimisation of Sizing, Shape, and Topology

Simultaneous optimisation of member sizing, structure shape, and topology represents the most efficient method of designing optimal truss structures. However, the number of researchers who have attempted discrete optimisation in this field are few. Some of the notable contributors are discussed in the following section.

Bohnenberger *et al.* [72] performed simultaneous optimisation of the sizing, shape, and topology of electricity pylon legs, but their approach cannot be considered a true simultaneous optimisation method, as they used a different method for each section. First they used a GA for optimisation for the structure topology. Each individual in the GA then had an Evolutionary Strategy applied to optimise the geometry of the structure, before sizing of the optimised individual was directly calculated. Once the sizing was optimised, the fitness of the GA individual was then set as the overall weight of the structure. A fault can be found with this method, however, in that no unfit individuals (individuals whose components violate design constraints) are capable of being generated. As any optimum solution (be it global or local) lies directly on the feasible/infeasible boundary [64], if a representation is incapable of generating an unfit solution then it cannot locate the feasible/infeasible boundary and as such will have difficulty locating a true optimum solution [62].

Shea and Smith [85] optimised the topology, envelope, connectivity, and member sizing of large-scale electricity pylon structures to great effect using a combined simulated annealing and structural grammar representation. They aimed to minimise both the self-weight and the overall material cost of the designs subject to multiple constraints including tensile and compressive stress, critical buckling, compressive and tensile slenderness ratios, angles between members, number of members and connections, and the weight of members. An important aspect of their approach was that they did not create designs from scratch. Instead they started with existing structures and improved them. This seeding method is similar in principle (though different in execution) to that used by Kaliakatsos-Papakostas *et al.* [21] in their evoDrummer work, where the authors created an initial population of multiple different rhythmic patterns by hand and then performed interactive evolution on that population. Shea and Smith compared the simultaneous optimisation of structural shape, topology, and member sizing against various combinations of sizing, node positioning, and overall envelope topology. Notably, their best solution (in terms of minimum weight) was not achieved with simultaneous optimisation but rather when the algorithm was constrained to use only optimisation of member sizing, node positioning, and overall envelope topology. The reason behind this can be explained by the fact that their fitness function considered far more than just the structure self-weight, taking into account other objectives such as the number of elements or joints. When compared with the full topology optimisation method, simultaneous optimisation was the best performing method, achieving convergence for all design constraints.

Kawamura *et al.* [73] appear to have implemented a layered GA to evolve all three, but unfortunately only devote four lines of their paper to this fact. Li and Chen [51], although optimising for minimum compliance rather than minimum weight (in accordance with the rest of the discrete optimisation literature), also derived a full topology optimisation method for discrete trusses.

Stromberg *et.al.* [86] recently implemented a highly innovative technique by combining both continuum and discrete analysis methods in the optimisation of laterally braced frames for wind loading on high-rise buildings. Their system used a continuum topology optimisation to approximate the required discrete layout, before performing a sizing optimisation on those discrete elements. Similar to Kicinger et al. [68] and Murawski *et al.* [66], they employed symmetry by only optimising half of the required solution and then mirroring it along the central axis as they found that symmetrical structures give the best performance. They also presented methodologies for connecting both continuum and discrete elements.

A Brief Note on Diversity of Optimisation Methods

As the number of different optimisation techniques is ever growing, it is impractical to conduct a comprehensive review of the literature detailing each different approach. Popular heuristic techniques such as PSOs and the ant colony search method have all achieved acclaim in their own right, but the sheer diversity of heuristic approaches is staggering [2]. Thus, this thesis will focus only on the areas of the literature that have addressed engineering design optimisation.

2.3.1.2. Compliance Minimisation in Discrete Optimisation

Gilbert and Tyas [87] optimised large-scale pin-jointed ground structures (>100,000,000 members). Their approach iteratively added members as required to an initial ground structure in order to optimise the structure for the lowest possible structural volume. They demonstrated that their solutions were demonstrably optimal for the given representation, and were very close to the true optimum solutions. They also noted that their method had a number of drawbacks in that i) it was slightly restricted as it could only add members, but not remove them and, ii) it could not optimise structures to minimise self-weight.

Li and Chen [51] provided a highly effective approach to generating beam structures based on principle stress lines, as proposed by Mitchell [5]. Building on the work of Rozvany [13], Li and Chen provided a mathematical model that allowed for simultaneous optimisation of shape, topology, and beam size in any given design domain subject to compliance minimisation. However, this method is only applicable in two dimensions, and due to its basis in stiffness optimisation, it can only be used in the optimisation of minimum compliance or maximum stiffness of a structure, rather than minimum weight, as with all other discrete approaches.

More recently, Zegard and Paulino [88] combined discrete and continuum methods to demonstrate how discrete elements could be optimised within a continuum. Their method looks at the changing stiffness properties of the overall continuum structure as the discrete elements are varied. The continuum structure itself is discretized into finite elements. The movement of nodal points within the continuum mesh then infers the shape of the discrete structure. This method is particularly useful in reinforced concrete design, where optimal placing of reinforcing steel bars can have a large effect on the overall performance of the material.

Wei *et al.* [89] demonstrated a similar technique to Zegard and Paulino to optimise discrete truss structures for minimum strain energy, as shown in Figure 2.15. Their method does not require the bar elements in their initial iterations to be fully connected. Full element connectivity occurs after a number of iterations. High density elements are embedded in a background mesh of a very low density, with the Young's Modulus of the background mesh being 1/1000th of that of the main elements. Changes in location, length, and size of the main bars then induces a change in the overall stiffness matrix of the structure, with increases in stiffness driving the optimisation towards its goal of minimal strain energy. Multiple bars can exist in the same location, indicating an increased member area.

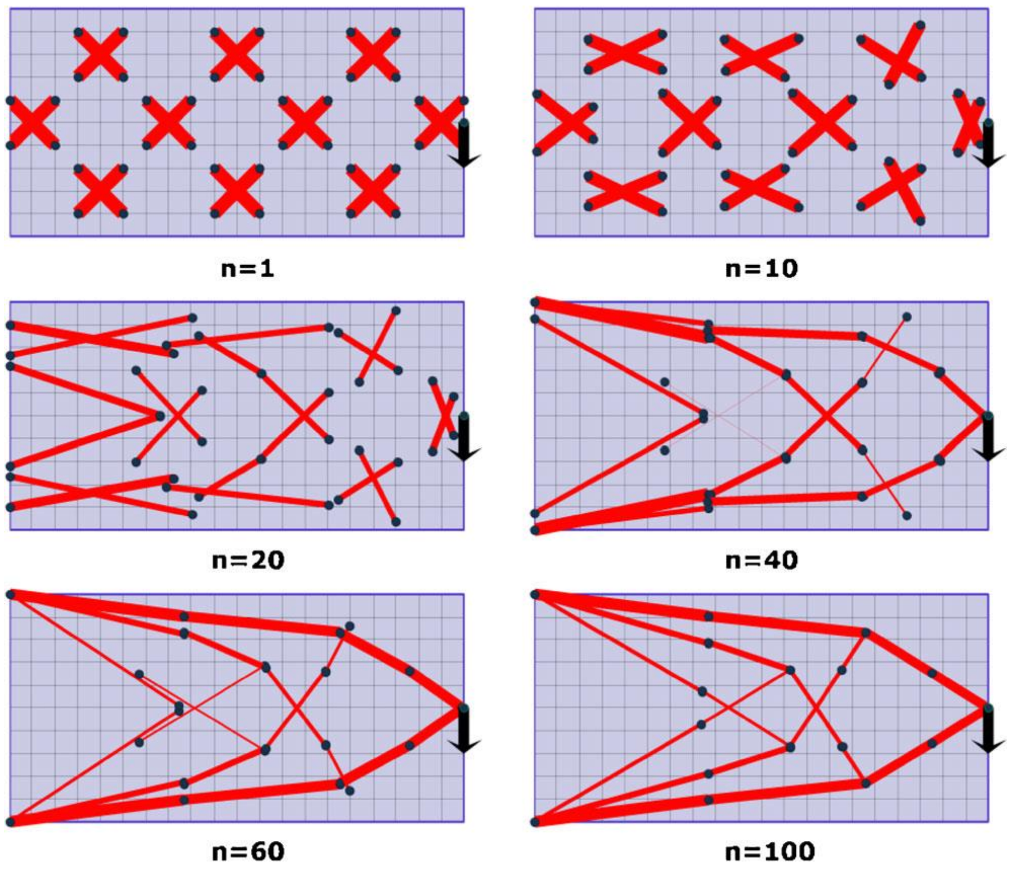


Figure 2.15: Stiffness spreading method for minimization of strain energy [89]

Furthermore, Wei *et al.* [89] do not provide units to the measurements of their design envelopes or their loading conditions, nor their exact material properties. This renders direct comparison to other methods difficult. Additionally, as with Li and Chen's approach [51], their method does not allow for optimisation in three dimensions.

Sokol [90] provided a compact and precise method for calculating exact Michell solutions [5] for discrete truss structures, written in Mathematica. Similarly to Gilbert and Tyas [87], Li and Chen [51], Zegard and Paulino [88], and Wei *et al.* [89] this method aims to minimise compliance rather than structural self-weight in the optimisation of the structure. This means comparisons can only be made with similar methods; discrete methods which minimise self-weight cannot be benchmarked against these approaches.

2.3.2. Continuum Optimisation

Continuum TOD is similar in principle to the finite element method of structural analysis [9] in that the system is assumed to be continuous and as such can be discretized into smaller elements which, when optimised, can be extrapolated to account for the overall design [91, 92]. The advantage of this approach over a ground-structure style approach is that it negates the need to know any information about the design space other than the boundary conditions (i.e. loads and reactions). These methods have proven to be highly efficient at finding minimum compliance material layouts for given load and boundary conditions [7]. A generalized description of the continuum approach can be seen as a material distribution problem over a design space upon which loading and reactionary conditions are imposed. Material is then added, removed, or rearranged from within the design space as needed (subject to a specified volume fraction of the original space). The structure is, thus, optimised such that structure compliance is minimised [93], as shown in Figure 2.16. The design space consists of a mesh of small elements with variable properties. The properties of these individual elements are changed to optimise the overall structure.

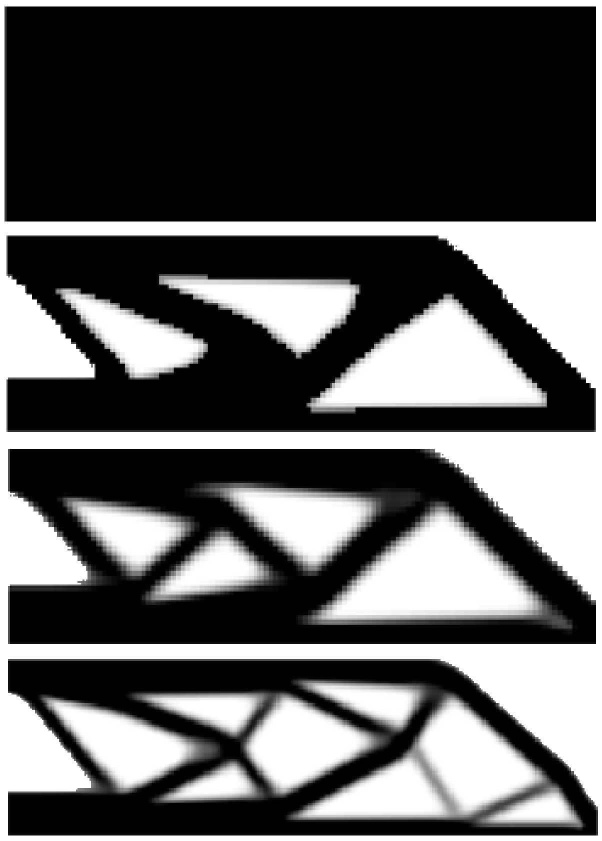


Figure 2.16: Continuum topology optimisation of a simple cantilever (created using the Solid Isotropic Material Penalisation (SIMP) method [94])

Comprehensive reviews of the field of continuum topology optimisation have been completed by Eschenauer and Olhoff [95] and Rozvany [7]. Topology optimisation of continuum structures began in 1988 when Bendsøe and Kikuchi published their homogenization method for the optimum material layout of structures [96]. Their iterative deterministic method was based on a series of equations for optimisation of shape and topology of structures for a given set of boundary conditions. The approach represented a key departure from previous optimisation methods as it only required a boundary in the form of a simple geometric shape, usually a square or rectangle, in order to generate an optimum solution.

Rozvany [7] provided a creditable overview of the field of structural optimisation. Continuum optimisation can be divided into deterministic and heuristic approaches. There are also further variations in the representational forms of the continuum material; the most prominent methods treat materials as either solids or voids (regularly called the Isotropic Solid or Empty (ISE) method [7]).

Deterministic Continuum Approaches

Bendsøe's homogenization method [93] and the subsequent SIMP (Solid Isotropic Material and Penalization) method proposed by Bendsøe and implemented by Rozvany *et al.* [97] are both considered the foundations of the entire continuum field and are the most popular methods [7]. The approach approximates the binary-style ISE representation for individual elements by varying the density/thickness of individual elements between 0 (white) and 1 (black). However, since a "grey" density of between 0 and 1 (i.e. neither 0 nor 1 but somewhere in between) cannot be truly represented nor manufactured, these intermediate elements incur a penalty. The overall compliance of the structure is computed through the use of finite element analysis. The deterministic method has been used in this manner to minimise the cost of the structure as a function of the thickness/density of elements for given compliances [97] and to minimise compliances for given volume fractions/weights [94].

The formulation for the classic compliance minimization deterministic continuum optimisation approach (as given by Sigmund [94]) is given in Eq. 2.1:

$$c(x) = U^T K U$$

$$= \sum_{e=1}^{N} (x_e)^p u_e^T k_e u_e \qquad \text{Eq. 2.1}$$

subject to:

$$\frac{V(x)}{V_0} = f \text{ (volume fraction)}$$

$$KU = F$$

$$0 < x_{min} \leq x \leq 1$$

where:

K	global stiffness matrix
U	global displacement matrix
F	force vector
u_e	element displacement vector
k_e	element stiffness matrix
x	vector of design variables
x_{min}	vector of minimum relative densities

N	total number of elements in discretized design domain ($no.x \times no.y$)
p	penalization factor (typically set to 3 [94])
$V(x)$	material volume
V_0	design domain volume

At each step the volume/density of each element is computed based on their current stress state. If the volume falls below a certain level (x_{min}), then the element is set to 0 (empty). Once each element has been evaluated, the global stiffness and displacement matrices are then computed to obtain the overall structure's compliance for that iteration step. Once the change in objective function between iterations falls below 1%, convergence is considered to have been achieved and the structure is deemed optimised [94].

More recently Bruggi and Duysinx [98] published a deterministic approach that aimed to minimise both compliance and the self-weight of the structure by implementing the Drucker-Prager failure criterion [99] for elastic materials that behave differently in tension and compression. They found that it was possible to simulate multiple objective optimisation with a continuum material by including local constraints on the material properties so that the strength of the material (and therefore the weight) was optimised in conjunction with the overall compliance of the structure. Zhou [100] optimised continuum truss structures. However, instead of minimising the compliance of the structure as with the most popular approaches, the fundamental frequency of the structure (given a certain volume fraction) was maximised in order to minimise vibrations.

Heuristic Continuum Approaches

While there have been many deterministic approaches towards continuum optimisation, evolutionary approaches towards continuum optimisation have also been used. Xie and Steven [101] first proposed an evolutionary structural optimisation method (ESO) in which redundant material is gradually removed in a binary fashion (the so-called "hard-kill" method) using populations of individuals with different arrangements of material. Low-stress elements are gradually removed from the solutions, until a final solution is achieved where all elements are highly stressed, in accordance with Michell's assertion [5] that the optimal solution will have all elements at a highly stressed state equal to their respective stress limits. A drawback of this, as noted by Zhou and Rozvany [92], is that ESO (along with other "hard kill" methods) can produce a non-optimal design (such as a kinetic mechanism) under specific conditions, due to the fact that removed material cannot be reinstated to the design, thereby limiting the effectiveness of the approach. Querin *et al.* rectified this with bi-directional evolutionary structural optimisation (BESO) [102], in which design elements could either be removed or added, thereby leading to a vast improvement in performance.

An alternative version of the ESO approach was proposed by Querin *et al.* [103], who reversed the ESO method of beginning with a solid block of material and slowly removed it by gradually adding structural material to an empty design space until the given constraints and structural properties were met.

Huang and Xie [91] provided an excellent introduction to the current field of structural optimisation via ESO and BESO. They concurred with Zhou and Rozvany's findings that the entire ESO method can break down and develop a highly non-optimal solution to a seemingly trivial problem. They suggested that this problem could be solved by either increasing the density of the mesh or by introducing "soft-kill" methods (i.e. where redundant material is replaced with low density elements). The soft-kill idea was expanded in Huang and Xie's subsequent paper [104], which introduced multiple materials to the BESO method thereby allowing for materials of different densities and characteristics to be incorporated into a single evolutionary design. Huang and Xie also used the BESO technique with a repetitive feature to design periodic structures [105], which proved both efficient and cost-effective, in terms of the computational effort required to design large structures using continuum methods. Finally, Huang and Xie [11] published an ESO method based on Sigmund's deterministic method [94]. In it, they also addressed some of the more critical comments on the ESO method [7] and admitted that ESO can fail to achieve convergence for an optimal solution. This can occur if the mesh is either too coarse or too fine but ESO is ultimately able to achieve more optimal solutions than the deterministic SIMP method [94].

However, Rozvany [7] noted that an increase in the mesh resolution will have a detrimental effect on the computational complexity of the problem. In fact, Rozvany is highly critical of any form of ESO-based method for their heuristic methodology:

"ESO is fully heuristic, that is, there exists no rigorous proof that element eliminations or admissions on the above basis do give an optimal solution... It is not particularly efficient if we have to select the best solution by comparison out of a very large number of intuitively generated solutions...

Although ESO usually requires a much greater number of iterations than gradient-type methods, it may yield an entirely non-optimal solution even with respect to ESO's objective function"

Genetic algorithms have also been extensively used in continuum optimisation. Numerous approaches have used a basic "bit-flip" approach, where individual pixels in a design space are assigned material in a binary fashion. This approach is popular due to the ease of mapping directly from the binary chromosome to the design space (in a similar fashion to discrete GA methods described above). Jakiela *et al.* [106] used a simple version of the technique to solve highly discretized two-dimensional cantilevered plates, with a particular focus on the use of the fitness function to effectively evaluate individual designs. However, this basic GA mapping approach has limited applicability, as any increase in the size of the design space requires a significant expansion of the chromosome length.

More complex tasks were completed by Zuo *et al.* [107] who combined Genetic Algorithms with BESO. In their approach the individual pixels (described as elements) within the design space were evolved from set length GA binary strings, rather than each pixel being mapped directly to a gene on the chromosome (which in itself would represent the entire individual). They also extended the BESO method to optimise structures in three dimensions, as shown in Figure 2.17:

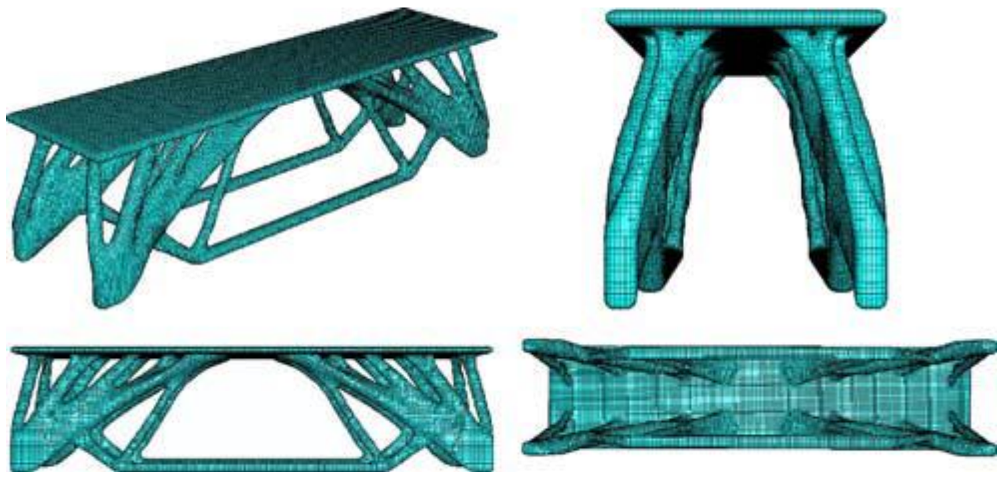


Figure 2.17: Three-dimensional continuum bridge [107]

In Figure 2.17 Michell's recommendation [5] of maximising symmetry in optimisation of structures is being used to its full effect, as only one quarter of the solution has been optimised. Huang *et al.* [108] also extended ESO methods to three dimensions and combined this with

Huang and Xie's [105] use of periodic structures and multiple material usage [104] to efficiently evolve three-dimensional bridge structures thereby requiring less computational effort than previously achieved.

2.4. Constraint Handling

Before an evolutionary algorithm can evolve an optimal solution to a problem, it must first learn to handle the constraints surrounding the problem [20]. Constraints are important for design optimisation as they represent high-level information about the performance of an individual in its design environment. Design constraints can be either "hard", i.e. non-adjustable conditions such as the physical size of the design space, or "soft", i.e. values which can vary between individuals in a population, such as the stress in each member or the overall maximum deflection. Only once these constraints have been minimised or overcome (i.e. truly fit individuals have been evolved, regardless of their optimality) can the evolutionary process be considered to be successful. Constraint handling techniques therefore become important when designing a strategy for addressing constraint violations [80, 109]. Two main such methods exist, with further sub-methods within each area [110]:

1. Penalty Functions
2. Repair Mechanisms

2.4.1. Penalty Functions

Penalty functions encompass all methods of constraint handling where an individual's fitness is penalised in some fashion for its performance with regards to the imposed constraints. If an individual breaks no constraints (i.e. they are fit), its fitness is unchanged. If, however, it fails a constraint (or a number of constraints), then its fitness is altered such that it becomes less likely to succeed in successive generations. Penalty functions generally have two distinct areas [111]:

1. Non-scaling Techniques – techniques that deal with constraint violations for each individual in the same manner regardless of the number or severity of said violations for each individual in a population
    a. The "death penalty" - outright rejection of unfit individuals from the population. Rejected individuals are replaced with new individuals who remain in the population, if they are fit [68]. This approach is recognised as being the

most common strategy of constraint handling [80]. However, Coello Coello [109] notes that it is only really applicable when the feasible search space constitutes a reasonable proportion of the entire representation space (i.e. when the ratio of fit to unfit solutions is large). If this is not the case, then the algorithm might have difficulty finding suitable individuals to replace unfit solutions, especially when there are a large number of unfit solutions in the population.

b. Static penalties - immediate stoppage of inspection of that individual and assignment of a single pre-determined "bad" fitness, uniform regardless of the severity of the constraint violation [12].

2. Scaling Techniques – techniques that impose variable constraint violations for a particular individual to modify the fitness function so that that individual receives a proportionally bad fitness, thereby hindering its chances of reproduction in subsequent generations.

   a. Count the total number of constraint violations per individual and modify the fitness of that individual based on some function of this number

   $$Penalty = f(Number\ of\ violations) \qquad \text{Eq. 2.2}$$

   Penalties which are purely functions of the number of violated constraints are unlikely to evolve an optimal solution, due to an insufficient description of the fitness space [111, 12].

   b. Ascertain the maximum value by which any single constraint across all elements in an individual fails, and modify the fitness of that individual based on some function of this violation

   $$Penalty = f(\max(violation\ for\ all\ violations)) \qquad \text{Eq. 2.3}$$

   c. Combine information on all violated constraints across every element in an individual to modify the fitness based on some function of that combination

   $$Penalty = f\left(\sum all\ violations\right) \qquad \text{Eq. 2.4}$$

Penalty Functions in Discrete Evolutionary Structural Optimisation

An overview of the methods used in a selection of the literature reveals a wide variety of approaches in constraint handling (comprehensive reviews have been completed by Michalewicz [80] and more recently Coello Coello [110]).

Kicinger *et al.* [68] used a version of the Death Penalty method in the form of a "feasibility filter", which simply eliminates unfit individuals from the population, and replaces them with newly created individuals.

Both Luh and Lin [77] and Deb and Gulati [14] used a scaling/non-scaling hybrid on the static penalty method, which has three levels of violation and produces three levels of penalty (10^9, 10^8, and 10^7 respectively), after which full scaling techniques (which are a function of the amount of constraint violation) are applied. They used five objective design constraints:

1. Truss is kinematically stable
2. Deflection is within limits
3. Stress is within limits
4. Member cross-sectional area is within limits
5. Member strength is within limits

In addition to these objective constraints they also had a *subjective* constraint of "Truss is acceptable to the user", which would imply a level of interactivity with the fitness function (this is not the convention with evolutionary structural optimisation in either continuum or discrete forms). This was accomplished by visually inspecting each individual to ensure that it complies with the design constraints (i.e. all basic nodes are present). This constraint notation is arguably flawed, however, in that both the cross-sectional area and the member strength should be seen as a non-negotiable quantity (i.e. the algorithm should not have the option of creating a member with an unacceptable area or strength in the first place).

Pholdee and Bureerat [62] used scaling limits, thus modifying the fitness by adding a penalty factor to the original fitness in the form of the summation of maximum failed constraints. This summation was then multiplied by a constant to further penalise the fitness value.

Kaveh and Talatahari [65] took an interesting approach to constraint violation with their hybrid PSO method. Since movement of particles is based in part on the fitness of their neighbours, a neighbour with a bad fitness will repel other particles. In their approach, if a particle/individual breaks any of the constraints as specified in the problem, the particle is automatically reset to its original position. This would seem to imply that particles at a local optimum (i.e. a position in the fitness space around which all other positions have a worse fitness) would essentially be immobile and would be unable to escape from the confines of their location.

Kawamura *et al.* [73] used a cumulative multiplier, which multiplies the summation of all violated constraints against each other. This in effect is a scaling penalty, but the authors failed to elaborate whether this violation multiplier was applied for each constraint per element in an individual or simply for a single member.

Ohsaki [63] used scaling penalties comprising the summation of all amounts by which constraints are violated, but only considered the analysis of stress and deflection; no buckling calculations were done.

Hajela and Lee [69] added a scaled penalty value of the sum of the failed constraint values, and compared the fitnesses of all individuals against the worst performing individual (usually limited to twice the average fitness of the population).

The most comprehensive of constraint handling techniques comes from Shea and Smith [85], who applied seven separate constraints to their evolutionary strategy. These constraints were handled as a mixture of hard and soft limits, with factors such as member slenderness ratios and angles between members as hard limits, and stress and buckling limits classified as soft limits (no displacement constraints were used, as no such information in the design documentation was found). The soft limits summed the violations of all constraints across all members and nodes in an individual, which was then multiplied further by a weighting/scaling factor.

2.4.2. Repair Operators for Unfit Individuals

With repair of unfit individuals, solutions which fail constraints are altered/repaired in some way such that they will then pass the constraints. These repair operators are especially suitable for discrete structural optimisation problems, as given a specific topological layout, there is the possibility to directly calculate the required cross-sectional areas for all members and, thus, improve the fitness of the structure [12].

Deb [112] noted that the most difficult aspect of penalty assignation is finding the most appropriate method of guiding the search towards the optimum (i.e. how to assign penalties to unfit individuals in such a way that the search process does not deteriorate). Deb proposed a method for use with GAs that used information of previously known fit individuals combined with a modified version of tournament selection to increase the proportion of fit individuals in each generation, thereby skirting the issue of constraint handling in unfit individuals by minimising their numbers. Deb argues that this forces the search towards the more feasible region of the search space.

However, Michalewicz [80] has a contrasting view on this matter noting that there can be multiple feasible regions within a search space - each surrounded by unfeasible regions. Given a random pair of individuals from the search space (one fit and one unfit) there is a possibility that the unfit individual could be "closer" in its semantic representation to the global optimum than the fit individual. Figure 2.18 shows a theoretical idealised fitness landscape, with the global optimum being represented by "x". In this instance, the unfit individual "f" is closer to the true global optimum than the fit individual "d", which is a local optimum. However, if fit individuals are always treated as superior to unfit individuals, then this makes it more difficult for the algorithm to find the true global optimum as it may become stuck in a local optimum. By this logic, using Deb's suggestion of forcing search towards feasible regions of the search space would potentially prevent isolated feasible regions from being included in the search process.

To put this concept into context, Li *et al.* [64] noted that Kaveh and Talatahari's method of resetting PSO particles to their previous best known location in the instance of violated constraints [65] can degrade the efficiency of the PSO method and ultimately prevent it from finding the global optimum. Li *et al.* allowed violations to occur, but then forced the particles to return to the feasible region in the design space. While Pholdee and Bureerat [62] stated that the optimised values of all constraints will be at or near to their respective constraint limits for a true optimum (though not necessarily the global optimum), Li *et al.* took this concept a step further by stating that any optimum solution will be found on the very boundary of the feasible/infeasible region within the search space (i.e. every constraint is at its very limit). This holds with Michell's description (as given in Section 2.3 [5]) that the global optimum exists where all constraints are at their maximum limits. Since any single increase in any constraint value would violate that constraint limit and, thus, render the solution unfit, arguably the true global optimum will be found on this boundary region [64].

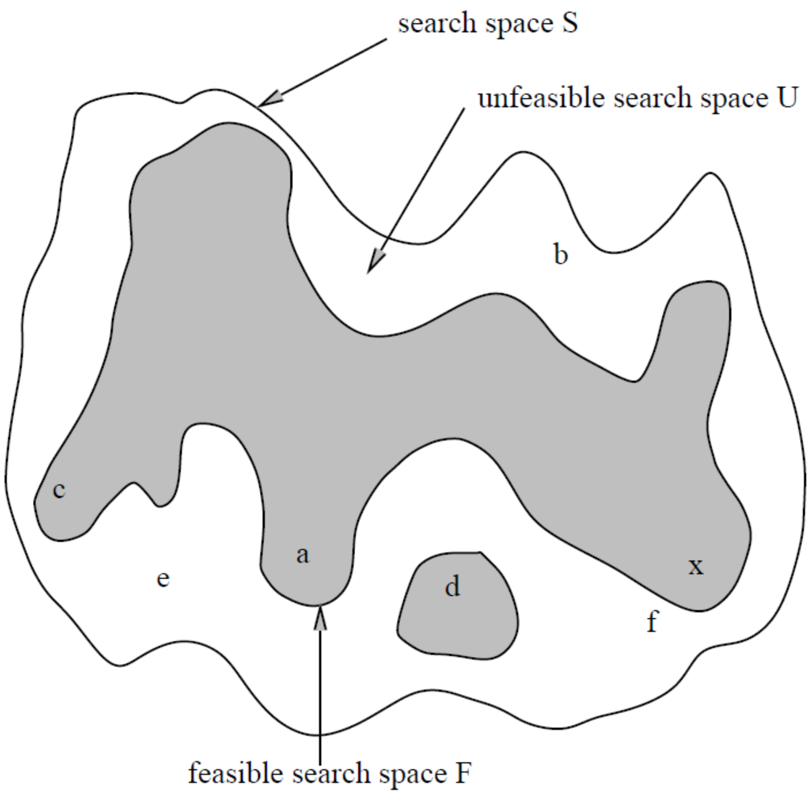


Figure 2.18: Multiple feasible regions (shaded in grey) within a search space [80]

In forcing particles to return to the feasible region of the design space, Li *et al*. [64] ensured that these particles crossed this feasible/infeasible boundary and, thus, had a much higher chance of finding a local (and the global) optimum. Since this crossing of the feasible/unfeasible boundary region leads to a better search, it is preferred to "repair" unfit individuals which lie just outside of the feasible boundary, such that they cross over (and lie close to) the barrier.

Shea and Smith [85] allow for small constraint violations on each individual; any individual with a constraint violation within some pre-defined limit is deemed to be repairable. In this instance, manual reassignment of member section sizes is done, until all constraint violations fall below a level of 3% (implying a level of interactivity, similar to that demonstrated by Luh and Lin [77] and Deb and Gulati [14]). Since in their work the authors must address 18 test load cases, optimisation of member sizings must be compatible for all load cases. However, they were unable to use deflection constraints in conjunction with this repair operator due to a lack of corresponding design documentation information.

Such a repair operator for minimally unfit individuals was also used by Kicinger *et al.* [68]. A key finding of their work was that symmetry produces better individuals; based on this, the authors modified specific genotypes of asymmetrical individuals such that they would become symmetrical, potentially increasing their fitness values.

2.5. Discussion

From the reviewed literature presented above, a number of observations can be gleaned with respect to the efficiency of various methods of evolutionary structural optimisation. While continuum topology optimisation approaches have been repeatedly proven to be computationally and structurally more efficient than heuristic forms of discrete truss design [7, 11], they are implicitly cost-ineffective to manufacture, as non-standard elements, forms, and construction methods are required [12]. Discrete optimisation methods currently have the potential to be applied to a far wider array of applications in a real world environment, as existing fabrication technologies and construction practices can still be used, thus requiring no bespoke industries.

All of the reviewed discrete optimisation literature related to discrete pin-jointed ground structure trusses optimised their member sizings by minimising the required cross-sectional area of each member. However, a number of questions can be raised about the validity of these results. A typical benchmark, discrete optimisation problem attempted by many within the reviewed literature is the 10-member, 6-node ground structure cantilever truss shown in Figure 2.19.

A selection of optimised member sizings from various approaches in the literature is presented in Table 2.2. The overall differences between three varying approaches is just 4.36 lb. To obtain these highly optimised overall structure weights, some very fine control of member sizings is needed; element cross-sectional areas with a precision of up to 0.0001 square inches (in the case of Kaveh and Talatahari) are required for construction of these solutions. Common sense would tell the reader that such fine control is impossible to achieve in current mass material production in the civil engineering domain, and even if it were this would render the manufacturing of such elements prohibitively expensive.

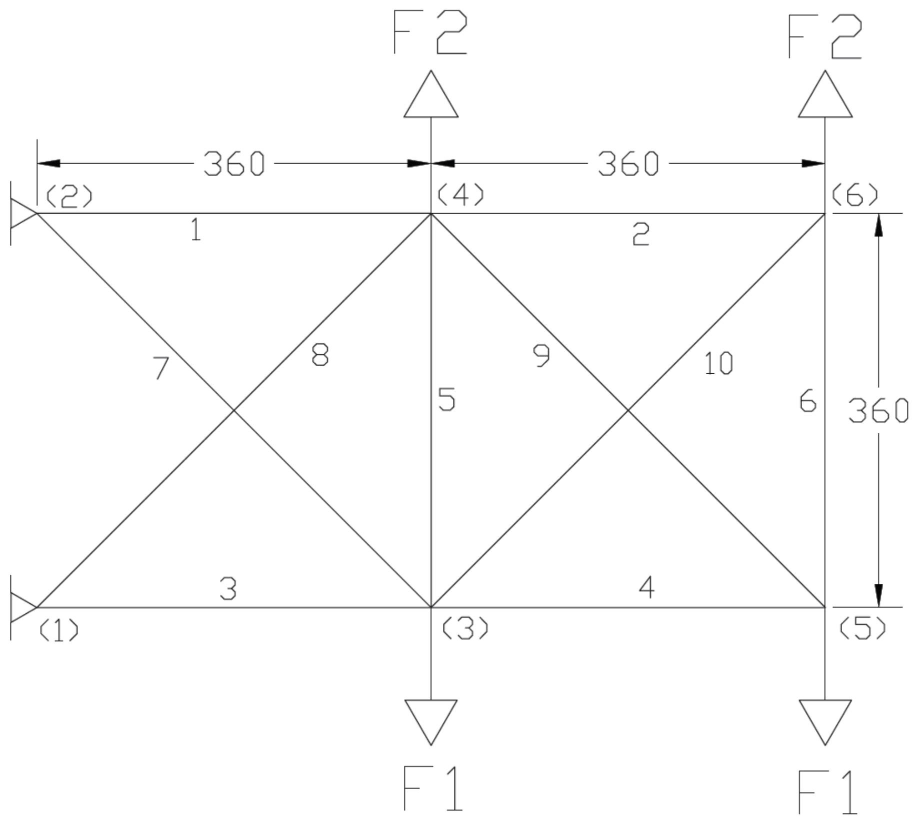


Figure 2.19: 10-member, 6-node ground structure problem [14]

A far more realistic approach would be to use readily available common construction elements. Yet, this review could find no benchmark optimisation problems, which were tackled in this manner. The only literature using real construction elements is that literature which deals with real-world optimisation problems, such as Bohnenberger *et al.* and Shea & Smith's electricity pylon optimisation approaches [72, 85]. The difference between idealised materials and real world construction elements on these same benchmark problems is therefore of interest to gauge the efficiency of the proposed evolutionary optimisation method and to observe any disparities between the use of different materials.

Table 2.2: Optimised element cross-sectional areas (in^2) for 10-bar, 6-node truss. F1 = 444,800 N, F2 = 0 N

Element	**Cross Sectional Area (in^2)**		
	Lee & Geem (2004)	**Li *et al.* (2007)**	**Kaveh & Talatahari (2009)**
1	30.15	30.70	30.31
2	0.10	0.10	0.10
3	22.71	23.17	23.43
4	15.27	15.18	15.51
5	0.10	0.1	0.10
6	0.54	0.55	0.52
7	7.54	7.46	7.44
8	21.56	20.98	21.08
9	21.45	21.51	21.23
10	0.10	0.10	0.10
Structure Weight (lb)	5057.88	5060.92	5056.56

Deb & Gulati [14], Ruiyi et al. [70], Hajela & Lee [69], and Luh & Lin [77] also proposed similar topology optimised solutions to a 10-member, 6-node ground structure (as shown in Figure 2.19) for a cantilevered truss. Their solutions allowed for the subtraction of members, but not the addition or location variation of nodes. In all cases the weight of the truss was minimised by varying the cross-sectional area of members. Their proposed optimised topology is shown in Figure 2.20.

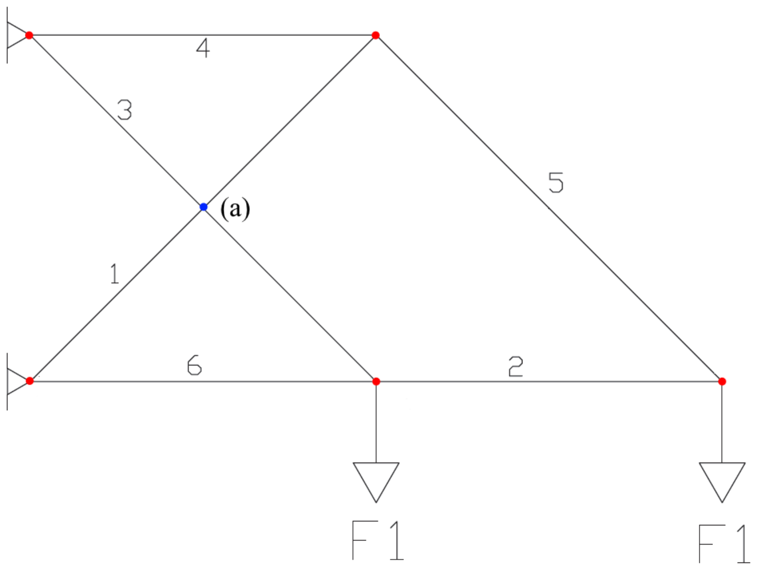


Figure 2.20: Proposed solution to 10-member, 6-node ground structure [14]

In this solution, it can be seen that there are five of the six original nodes remaining (shown in red), while there are six of the original ten elements. What is interesting to note here, however, is that while Deb and Gulati acknowledge that co-axially overlapping members (i.e. if member 2 extended through member 6 to connect the outermost loaded node to the support) are physically impractical, their solution, and indeed every proposed solution to this particular ground structure problem (including those solutions lacking a topological bit operator, as shown in Table 2.2) involves overlapping members.

In Figure 2.20 members 1 and 3 overlap at their midpoint, thereby creating a new node (noted at point (a)). Ohsaki [63] defines a node as a point where at least one extra member with a positive cross-sectional area meets another. Hajela and Lee [69] impose a specific design constraint that no solution can add new nodes, yet by any definition of the term, new nodes are necessarily being created by their designs where members intersect. All ground structure optimisation methods reviewed in the literature ignore such overlaps with the exception of the work presented by Gilbert and Tyas [87]. In reality, however, these planar member overlaps have significant implications for the structure. In the given example, member 1 (one of the three longer members) is in compression. As the longest compression member in the structure, it would be at risk of buckling under high loading conditions. However, the intersection of member 3 at its midpoint creates a new nodal point, effectively laterally bracing the member. This means that the analysis and optimisation results of this structure are unrealistic as they are based on a fundamentally unsound assumption, and, thus, they do not represent the true performance of the structure.

Arguably, the most interesting discrete optimisation methods found in the literature are those presented by Li and Chen [51] and Wei *et al.* [89]. Li and Chen developed an approach for calculating the principle stress field for any given design domain and were subsequently able to optimise a beam structure to satisfy all of the design criteria within that domain, with respect to minimization of the strain energy of the structure, as shown in Figure 2.21.

Both this approach and that of Wei *et al.* [89] (shown in Figure 2.15) are notable for one main reason: just as with continuum optimisation methods, they do not require any information about the design domain other than the boundary conditions (i.e. loading, fixing locations, etc.). However, their approach is quite limited, in that it only allows for minimization of strain energy (yet again the same as continuum optimisation methods). As such, no direct comparisons with discrete optimisation techniques addressing weight minimization could be made.

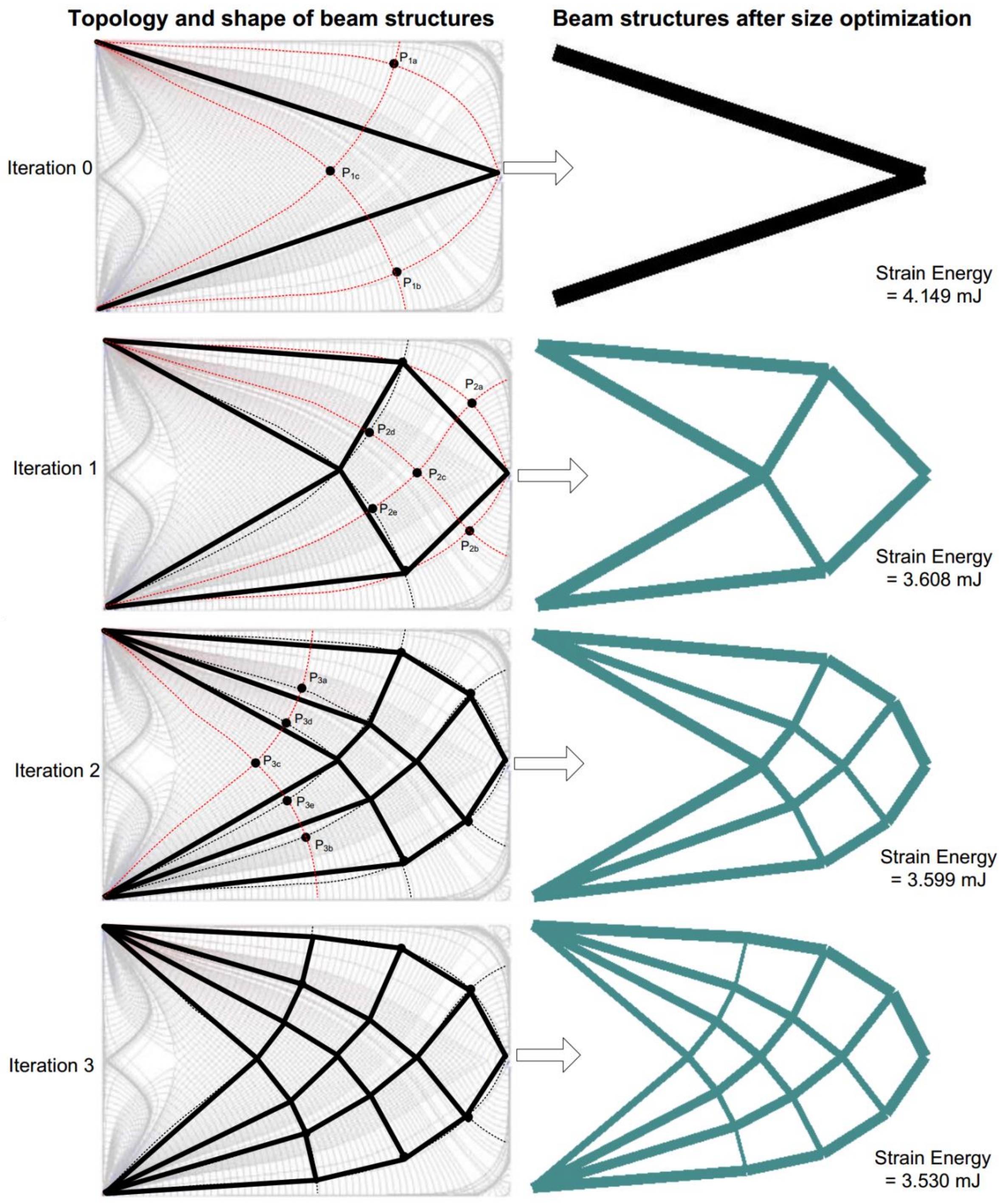


Figure 2.21: Discrete optimisation of the strain energy of a cantilever structure [51]

2.6. Summary

This chapter presented an overview of the literature surrounding the areas discussed in this thesis. Evolutionary Computation and its derivatives were reviewed, including Evolutionary

Algorithms, Genetic Algorithms, and Grammatical Evolution. An overview of the state of evolutionary structural optimisation was given, and the various methods therein – including both discrete and continuous optimisation methods - were examined. Differing optimisation methods from the literature were compared and contrasted, and a number of recommendations and interesting potential research areas were identified:

- The majority of reviewed discrete optimisation literature use unrealistic assumptions about material properties and physical structure layout, while fundamental engineering principles are overlooked. This is not typically an inherent defect of the methods used, but rather a poor definition of the problem;
- There is a tendency in the literature to focus on minute improvements to optimisation methodologies, leading to results that require manufacturing precisions that are currently unrealistic in the field of civil and structural engineering. This issue can be readily addressed by constraining the methods to use common construction elements.
- Traditional ground structure discrete optimisation approaches are severely limited in their representation capabilities by the necessity to specify all potential solutions *a priori*, and thus cannot cover the search space as effectively as possible;
- Simultaneous optimisation of member sizing, structure shape, and structure topology produces the best theoretical optimisation results but is difficult to represent with a constrained ground structure representation and even more difficult to achieve effectively due to the size of the required representation space;
- The larger the representation space, the better the chance of success is and the better is the overall quality of the individual solutions. However, a larger representation space results in less fit individuals; the management of unfit individuals via the handling of constraints then becomes crucial to improve evolutionary performance.

The following chapter provides an in-depth discussion of the Grammatical Evolution method, and the approach to design generation and the evolutionary process as used within this thesis.

Chapter 3. Design Generation & the Evolutionary Process

3.1. Introduction

This chapter will outline the approach to design generation and the evolutionary process as used in this study. The chapter describes the methodology used in this thesis and provides a full description of the Grammatical Evolution (GE) method employed [48, 46].

GE is a grammar-based form of GP where the grammar provides a representation in which one can easily encode the structure of the possible solutions [46, 113, 114]. In its most basic form, GE begins with a population of individuals, with each represented by a genotype - a single chromosome (an array of integers). Each chromosome is passed through a grammar, which acts as a form of translator to convert the integer array into an executable program. Execution of the program produces a solution, known as the phenotype, which is evaluated by passing it through the fitness function in order to determine its suitability for purpose. The fully evaluated population is then ranked in order of fitness. The next generation is created by varying the population to produce new individuals with genetic qualities of those from the previous population, in such a way that the fittest individuals have the greatest chance of passing on genetic material to subsequent generations. After a number of iterations, the single best individual is returned as the best evolved solution. A basic graphical overview of the GE process is presented in Figure 3.1:

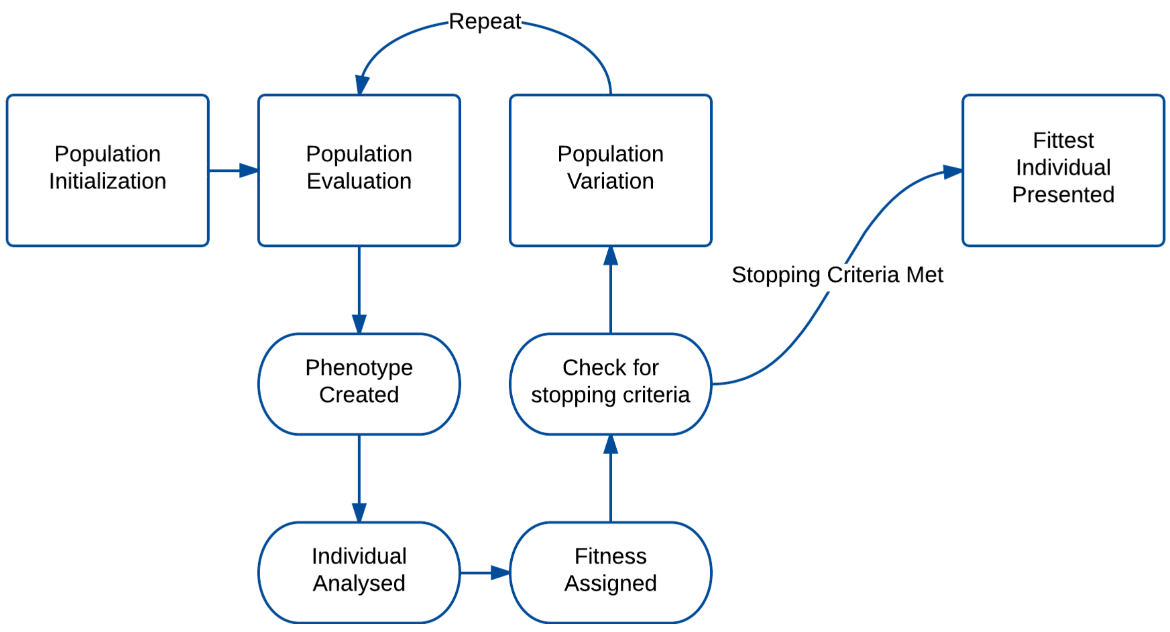


Figure 3.1: Basic overview of the GE process

GE's advantages over a regular GA lie primarily in the power of the genotype to phenotype mapping process. A formal grammar is used to define a set of production rules, which generate production choices using a GP-style tree-based mapping method as described in Section 2.2.3. Each gene on the chromosome of an individual dictates which specific production choice is to be chosen from a particular production rule (this process is referred to as the "mapping"). Production rules and choices are expanded to create an executable program, which generates a solution when executed. This intermediate step means that direct mapping from the genotype to the phenotype (the standard method for GAs [34]) is not necessary. While similar results may be technically possible using other methods, GE's use of the formal grammar makes the process far easier and faster, while also allowing for greater flexibility in the representation of the solution [46]. The combination of multiple production rules and choices allows for complex programs to be derived with ease, and one can efficiently embed all manner of useful domain knowledge in the grammar constraining the form of the generated solution [46, 115].

When comparing the use of grammars in structural optimisation to pre-existing methods such as Artificial Neural Networks or Particle Swarm Optimisation, grammars have the advantage that the output from the search process is human-readable. The grammar can be readily altered to suit any application, which allows for easy and quick analysis of results across a variety of platforms. The method also enables the addition of numerous constraints and bias about the structure of the solution into the grammar itself [48, 113, 56, 55], all of which can reduce the representation space to a more manageable size.

From the literature discussed in Chapter 2, it has been identified that the most efficient method of evolving optimal discrete truss structures is to optimise the structural shape, structural topology, and the member sizings in parallel. In order to achieve this, a new technique must be utilized that will allow for the generation and evaluation of a highly diverse range of structural forms. GE has been previously proven with regards to generating architectural [29] and sculptural forms [55], and can be readily employed for structural engineering applications by adding analytical capabilities that allow for the evaluation of the physical stress states and resultant properties of the designs [61].

Evolution of multiple data sets using genetic-based evolutionary algorithms requires information regarding those data sets to be encoded in the chromosome. In order to evolve different data sets (i.e. multiple aspects of a single design concurrently), a large chromosome would be required, along with the ability to not only identify separate sections on the chromosome relating to the different data sets but also to keep those sections distinct during variation and replacement between generations. Whereas traditional GE uses a single chromosome to evolve solutions, this thesis theorises here that simultaneous optimisation of structural shape, structural topology, and member sizing could be done in parallel with the addition of a second chromosome. By restricting the first chromosome to defining the topology and shape of the overall structure and leaving the governance of member sizing to the second chromosome, there would theoretically be complete isolation between data sets. With careful management, this would allow for synchronous evolution of the structural shape, structural topology, and member sizing. This concept is discussed in more detail in Chapter 4.

3.2. Methodology Overview

What is presented in this chapter is a tool for design and optimisation of physical truss structures using Grammatical Evolution. The Python programming language is used due to its relative usability, its capabilities in interacting with other programs, and its excellent documentation. The truss structures described here are composed of simple beam elements, each with specific material properties. Each structure must obey design constraints such as the maximum size of the structure and the physical limits of the materials, along with structural limits imposed by design codes of practice [60]. Using a method that has previously been found to be fast, reliable, and accurate [61], structures are analyzed using a separate open source analysis program [116], and are evaluated based on comparisons between the results of the analysis and the provided design constraints.

In a departure from traditional GE [44], the implementation described here uses two separate chromosomes to simultaneously evolve two aspects of the solution [117]. Chromosome A (hereafter referred to as Ch.A) governs the topology and member layout of the structure by defining node locations and connections (edges/beam members), while Chromosome B (hereafter referred to as Ch.B) governs the sizing of the individual structural members within the structure. In this manner the simultaneous optimisation of structural shape, topology, and member sizing was attempted.

A graphical overview of the entire evolutionary process used here is shown in Figure 3.2. Each of the main steps in the leftmost column of Figure 3.2 represents a major process acting on populations of individuals. Each main process is composed of many smaller steps, which are performed on each of the individuals (or groups of individuals) in the population at that step. Details of each of these main processes, steps, and individual functions are elaborated in subsequent sections in this chapter:

- Section 3.3 details material specification;
- Section 3.4 details population initialization;
- Sections 3.5 to 3.8 detail population evaluation;
- Section 3.9 details selection, variation, and replacement.

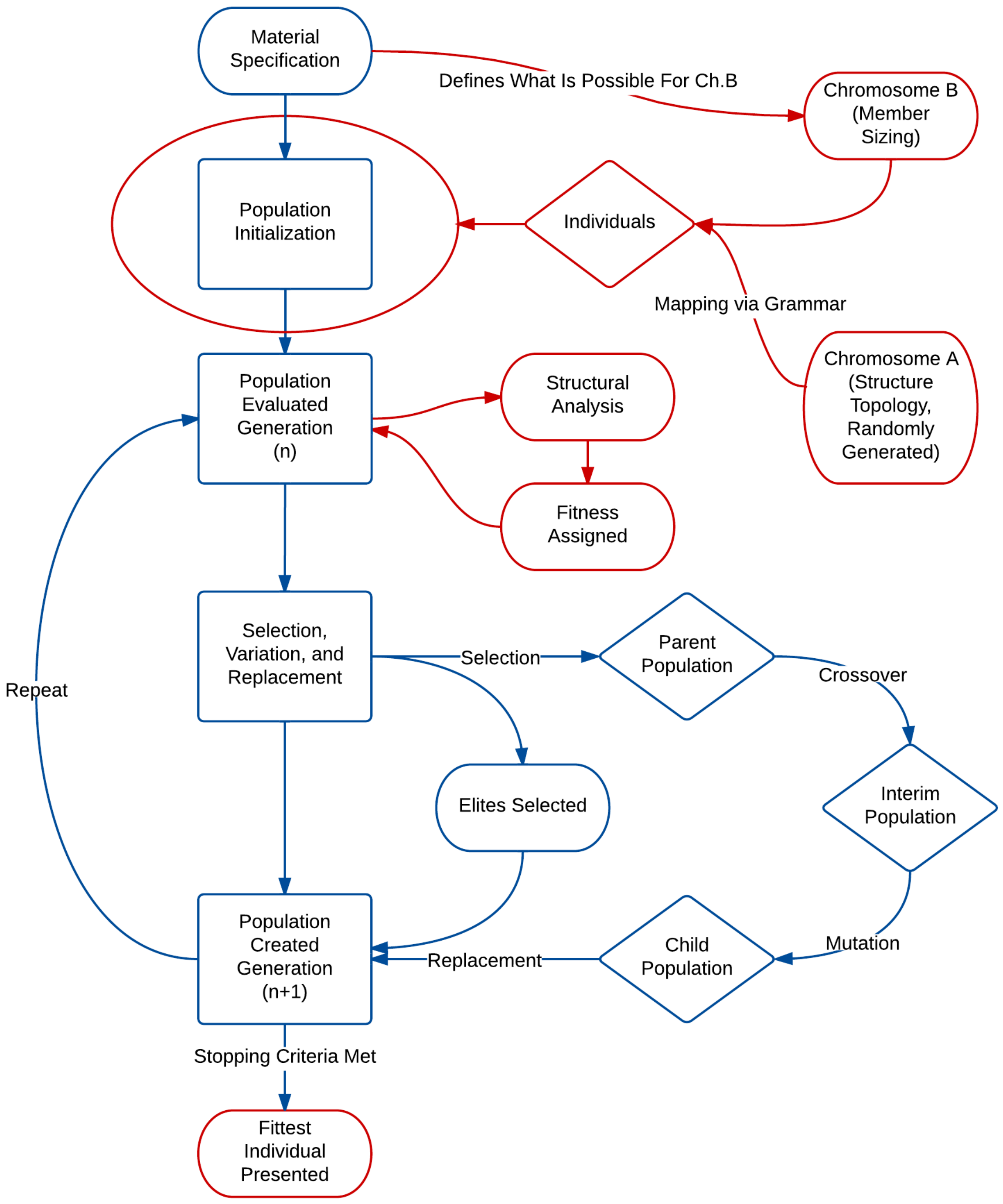

Material Specification
Defines What Is Possible For Ch.B
Chromosome B (Member Sizing)
Population Initialization
Individuals
Mapping via Grammar
Chromosome A (Structure Topology, Randomly Generated)
Population Evaluated Generation (n)
Structural Analysis
Fitness Assigned
Selection, Variation, and Replacement
Selection
Parent Population
Crossover
Interim Population
Elites Selected
Repeat
Population Created Generation (n+1)
Replacement
Child Population
Mutation
Stopping Criteria Met
Fittest Individual Presented

Figure 3.2: Flowchart of the Grammatical Evolution process as used in this thesis. Blue represents actions performed on populations of individuals, whereas red indicates actions performed on individuals themselves.

3.3. Material Specification

Much of the GE implementation presented in this thesis depends on the available construction materials and section sizes from which the structures are composed. The process begins by generating a list of the available materials and material section sizes selected for that specific run based on a pre-determined material input file. While any type of material can be used, for this study available construction materials are taken from the Tata Steel Blue Book [118]. Unless otherwise specified, the materials used are S355 steel "Celsius®" hot finished Circular Hollow Sections (CHS), which are in accordance with BS 5950-1: 2000 [60] and BS EN 10210-2: 2006 [119]. Circular hollow sections were chosen due to their uniformity and consequently the fact that orientation of the members is not a consideration. The material has a density of 7,850 kg/m^3 and a Young's Modulus of 210,000N/mm^2.

Once the program is initialized, a list of available section sizes is compiled based on the specified materials. Each section has the following properties:

i. Section id. This is the reference tag by which each particular section size is known. Ch.B is essentially comprised of section id tags, each of which will reference a particular steel section.
ii. External diameter
iii. Thickness
iv. Unit weight
v. Cross-sectional area
vi. Second Moment of area
vii. Young's Modulus
viii. Density

In addition to these basic material section properties, each section carries a knowledge of its compressive resistance (P_c) when in axial compression, as detailed in the Tata Steel Blue Book [118]. These compressive force limits are given for specific effective lengths of members. For section sizes with an outer diameter of less than 273mm, compressive resistances are given for effective lengths of 1m, 1.5m, 2m, 2.5m, 3m, 3.5m, 4m, 5m, 6m, 7m, 8m, 9m, and 10m. For sections with an outer diameter of 273mm or greater, compressive resistances are given for effective lengths of 2m, 3m, 4m, 5m, 6m, 7m, 8m, 9m, 10m, 11m, 12m, 13m, and 14m. Since truss members are considered to have pinned connections and the effective length factor of a pinned-pinned member is 1.0, the given compressive resistances can be applied to members in compression without modification.

These sections not only define the strength of each element in the structure but also are used to constrain the values for Ch.B of each individual – the maximum permissible value of any part of Ch.B is set equal to the number of available section sizes. Using this correlation, it is then possible to directly map from Ch.B to the phenotype of the individual (in a similar manner to a traditional GA mapping) to define the section size of each member in a structure.

3.4. Population Initialization

Initialization of the population in the first generation is achieved using a GA-specific initialization method known as random initialization. As its name would suggest, the population is initialized by randomly generating chromosomes of a pre-specified maximum length (rather than specifying different tree layouts, as would be the case in canonical GP [33]). GAs usage of a binary genotypic string lends itself well to random initialization; a GA population is most usually initialized by generating a population of random binary strings [34]. However, using this method, it is possible for the initializer (or the evolutionary process) to create a chromosome that does not map fully.

As detailed in Section 3.5, a production choice may either designate a terminal or may lead to a new production rule (itself with further production choices). Since each gene on the chromosome dictates which production choice is to be selected from a production rule, the potential then exists so that by the time the end of the chromosome is reached, all production rules have not finished being expanded (i.e. the mapping has not terminated and there are still open branches). In this instance, GE has the inherent ability to "wrap" chromosomes. If an individual is not fully mapped by the time the last codon on the chromosome is reached, the mapping process will continue to expand the remaining production rules by restarting from the beginning of the chromosome [48]. However, in recent work [120] wrapping has been shown to be damaging, if the grammar is poorly designed. If an individual is not fully mapped after the mapping has wrapped twice around the chromosome, it can be considered that the mapping process will never terminate. The chromosome length, thus, must be sufficiently large that this does not occur. For problems where no recursion exists (i.e. all individuals will have the same chromosome length) a randomized initialization process is the most efficient method [20].

In this thesis (unless otherwise specified), each individual instance will have either one (Chromosome A, or Ch.A) or two (Ch.A and Ch.B) chromosomes. When using one chromosome in this study, GE is operating in a similar fashion to a traditional GA. This study exclusively uses random population initialization of individuals in the first generation. For each chromosome, an integer string is generated up to a specified length. Longer chromosomes have the effect of lengthening the run time of the evolutionary algorithm, while shorter chromosomes may not be able to allow the grammar to adequately expand to its maximum tree depth. In most cases in this work, only a very short portion of the chromosome is actually used by the most successful individuals. The maximum chromosome length must still be carefully chosen such that it is long enough to ensure every individual will be completely expanded but not so long as to adversely affect the run time of the evolutionary process. A number of initial experiments were performed with varying chromosome lengths, and maximum chromosome length of 500 codons was selected for all experiments performed in this thesis.

3.4.1. Population Size

The population size has a significant impact on the effectiveness of the evolutionary process. Too small a population will see the algorithm getting stuck in a local optimum (whereby any single change on the chromosome of an individual would result in a worse fitness), while too large a population size will increase the run time of the program to infeasible levels. The majority of computational efforts required in each evolutionary run revolves around the evaluation of individuals; this phase can therefore be viewed as a “bottleneck” – a larger population will mean more individuals to evaluate. Multi-core threading is used to great effect in this study in order to minimize the computational time required to evaluate a population of individuals, as the evaluation and analysis of one individual has no bearing on that of another individual. Thus, multiple individuals can be evaluated simultaneously across multiple cores, thereby leading to greatly reduced overall computational time. This multi-core processing allows for larger populations to be evolved. All other steps outside of these evaluations must be run on a single process thread and as such cannot be parallelized.

However, a question still remains as to the most appropriate ratio of population size to generation size for an evolutionary run. The only way to truly tell the most suitable settings for an evolutionary run is to perform a parameter sweep set of experiments, whereby a number of variables are kept constant and a select few are varied. A simple such experiment was run in order to ascertain the most effective ratio between the population size and generation size. To this end, a basic grammar was created to evolve a simple cantilevered truss, as shown in Figure 3.3.

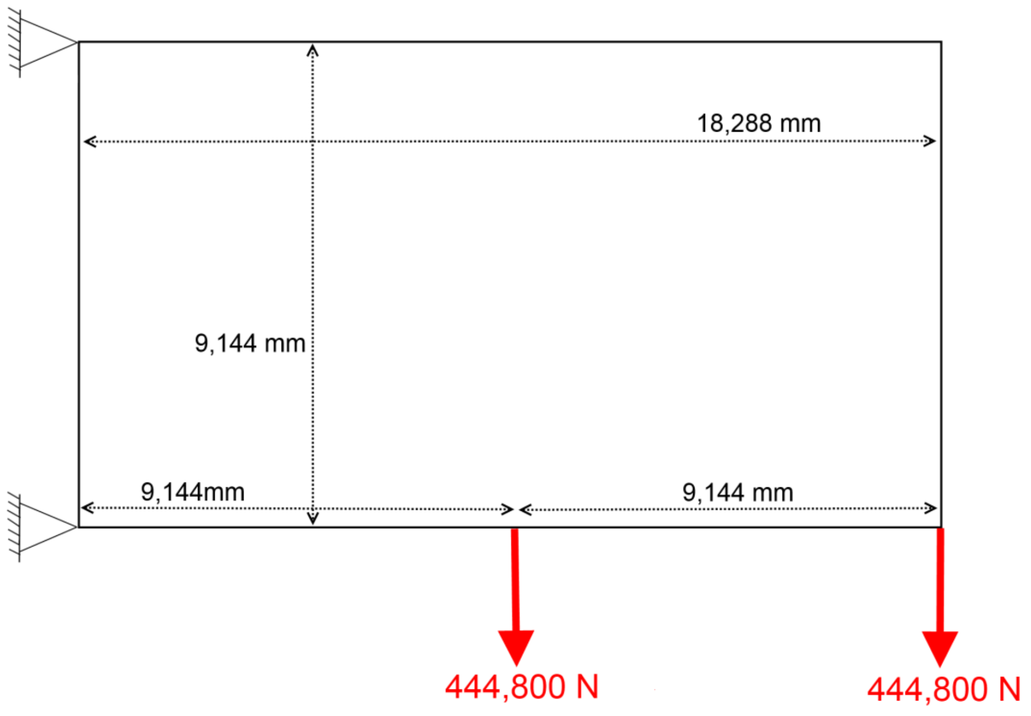


Figure 3.3: Design envelope for parameter sweep experiment

The total span of the structure was set at 18,288 mm (60 feet) and a height of 9,144 mm (30 feet). Two vertical loads of 444,800 N were placed on the structure. Mutation rates and crossover probability were held constant at 1% and 75%, respectively. The population size was then varied from 100 to 1000, while generation size correspondingly decreased from 1000 to 100. The total number of evaluations for every run remained constant at 100,000.

The objective of this experiment was to see which combinations of population size and generation size produced not only the lowest average fitness values but also the fastest convergence and lowest standard deviation. For each evolutionary setting, 30 runs were completed. At each generation in each run, the fitness value of the fittest individual in the population was recorded. Once a full set of runs were completed for particular parameter settings, the average and standard deviations of the best fitness values across all runs were calculated for that run. These average fitness values were then graphed to easily visualize differences between evolutionary parameter settings.

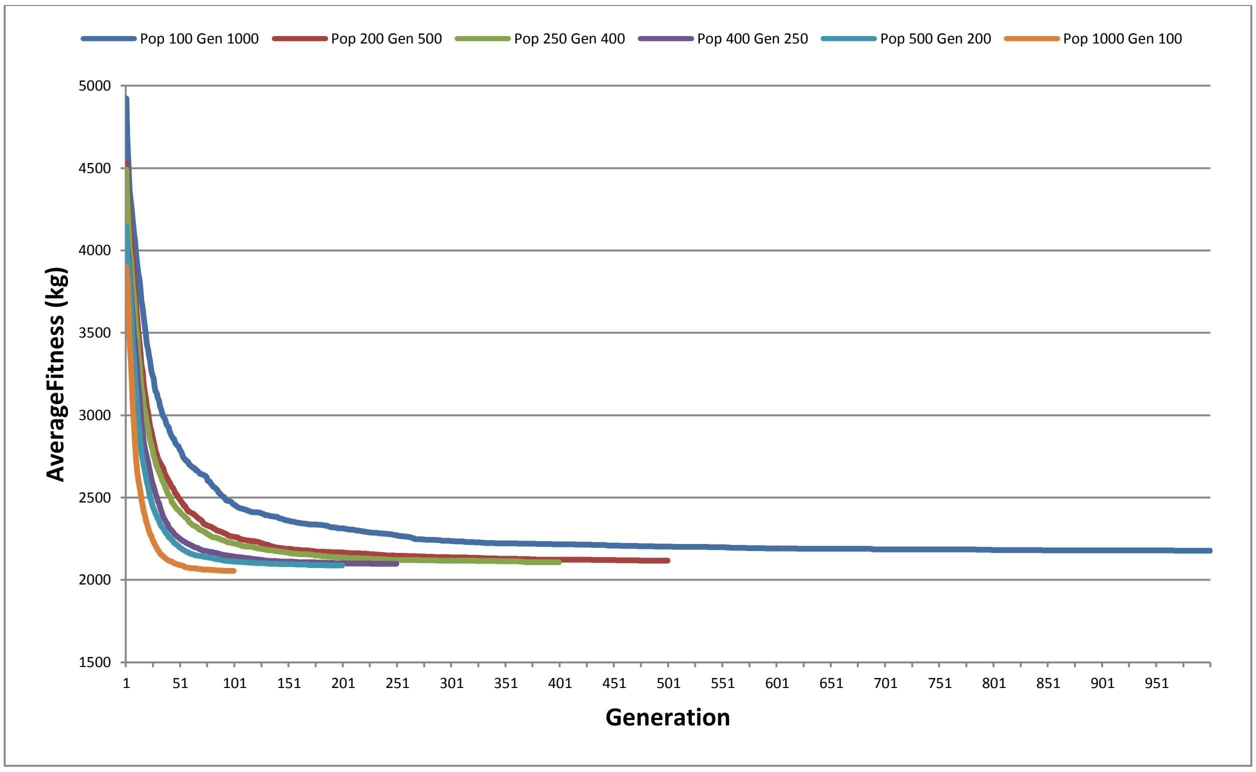


Figure 3.4: Parameter sweep experiment to demonstrate the difference between population and generation sizes

Figure 3.4 shows the combination of a high population size and a small number of generations leads to both a higher convergence rate and a far lower average fitness than a low population size with a higher number of generations. A lower overall standard deviation can also be seen in the results presented in Table 3.1:

Table 3.1: Final results from evolutionary parameter sweep

Population	Generations	Standard Deviation (final, in kg)	Average Fitness (final, in kg)	Average Run Time (hh:mm:ss)
100	1000	167.17	2177.97	00:10:48
200	500	108.33	2116.38	00:10:57
250	400	111.33	2106.92	00:10:58
400	250	148.92	2096.91	00:11:19
500	200	83.49	2086.96	00:11:37
1000	100	62.96	2054.65	00:13:15

Figure 3.4 and Table 3.1 both show that the combination of a larger population size and a comparatively small number of generations leads to better evolutionary performance in the problems addressed in this thesis. A marginal increase in computational run time can be observed with increasing population sizes. These findings corroborate those of both Koza [33] and Goldberg [121], who recommended as much as possible to keep the population size high in relation to the number of generations.

3.5. Grammar

As with spoken language, a formal grammar defines a language via a series of rules. These rules (called production rules) each have a number of choices (called production choices) that they can create. Production choices can be terminals (i.e. no further choices can be made, similar to a leaf at the end of a branch), non-terminals (leading to more production choices, similar to a fork in a branch which leads to more branches), or combinations of both. Figure 3.5 shows a basic non-recursive Bacus Naur Form (BNF, [45]) grammar, starting with the production rule `<Rule_1>`. The non-terminal production choices for this rule are `<a>` or `<b>`, each of which have their own terminal production choices.

```
<Rule_1> ::= <a> | <b>
<a> ::= x | y
<b> ::= x + y | x - y
```

Figure 3.5: Example of a non-recursive grammar rule

The grammar example shown above uses a fixed-length chromosome, with a depth of 2 (i.e. each time the grammar is used, it will only require the use of two codons). However, there is the possibility for grammars to be recursive by setting a production choice as a part of its own non-terminal, as shown in Figure 3.6.

```
<Rule_2> ::= <a> | <b> + <Rule_2>
<a> ::= x | y | z
<b> ::= x + y | x - y | (x + y)/z
```

Figure 3.6: Example of a recursive grammar rule

In this example, there is a 50% chance that the production rule `<Rule_2>` will be mapped again, meaning a variable length chromosome is needed. The lowest number of required codons is 2 (as in the fixed-length example shown in Figure 3.5), but theoretically there is no upper bound to the length of the chromosome required for this grammar. While this may seem like a potential issue, probability theory ensures that recursion does not become unmanageable. From the outset, there is a 50% chance that the production rule `<Rule_2>` will be called again. If it is called again, it again has a 50% chance of being re-called for the next iteration. The probability of `<Rule_2>` being called over n iterations is $1/2^n$; as the recursion depth increases, so too does the probability of termination. In this thesis, only Ch.A is passed through the grammar itself to generate the derived program in the above manner. Ch.B comes into effect once the mapping process is completed and the derived program is executed.

3.5.1. Truss Generation

Traditional ground structure approaches to design generation use individual genes in the chromosome to indicate the presence or absence of elements in the design representation in a binary fashion [69]. More elaborate approaches expand on this to allow multiple genes to encode information for each design element, thereby increasing the design representation capabilities and, thus, the representation space [63]. The grammatical approach described here differs in that physical designs are created by generating node locations and edge properties. The chromosome can indicate the presence, absence, or location of any number of nodes or edges via the grammar. The derived program will then create a list of those nodes and edges for later manipulation by the evaluation function. This intermediate step allows the grammar more freedom in its expression of physical designs, thereby negating the need for restrictive direct mapping from chromosome to phenotype.

3.6. Genotype to Phenotype Mapping

The mapping process in GE is a major departure from traditional GP [46]. Whereas GP generates individuals using a tree-based system [33], GE is based on the biological notion of genotype to phenotype mapping. Production rules/non-terminals are expanded from left to right [46]. The expansion of a non-terminal is determined by the value of the current gene modulo the number of production choices in the current rule.

This GE mapping process can be readily explained using the corresponding biological terminology. The phenotype of a solution generated by GE is defined by that solution's chromosome, in the same way that the phenotype of an organism is defined by that organism's DNA. Any changes to the DNA of that organism, such as mutation of individual genes, or crossover of entire portions of the DNA with that of another organism, will have corresponding observable effects on the phenotype of the organism. The phenotype of the organism is translated from DNA via RNA, whose sequence establishes the combination of amino acids, which in turn creates a protein which has a phenotypic effect. Similarly, the binary genotype (traditionally favoured by GA methods [34]) is converted to an integer string, whose sequence selects a combination of productions from the grammar, which in turn generates a phenotype (in the form of an executable program), which finally creates a solution when executed.

The production choice for a codon is the current chromosome codon modulo the number of available production choices:

$$Choice = Current\ codon\ \%\ No.\ Productions$$

Eq. 3.1

GE can also be modified to map directly from the chromosome to the phenotype. This is achieved with a special production rule known as `<GECodonValue>`. Whenever the grammar parser reads this string, it substitutes the next available codon value from the chromosome as an integer. The mapping terminates once no non-terminals remain in the derivation string.

To give an example, let the chromosome shown in Figure 3.7 be applied to the variable-length grammar presented in Figure 3.6.

```
[13,4,5,8,16,1,14,19,3,16,15,15,0,7,3]
```

Figure 3.7: Sample chromosome of length 15 and maximum codon depth 20

There are two production choices for `<Rule_2>`, three for `<a>`, and three for `<b>`. Using this information, each codon on the chromosome creates a part of the derived solution. This mapping process is shown in Table 3.2.

For this sample chromosome, once the mapping reaches the sixth codon, all non-terminals have been mapped, and the mapping itself terminates. The remaining portion of the chromosome (known as the tail) remains unused, although subsequent crossover or mutation on the individual could bring it in to play. The final phenotype for the sample chromosome is shown in Figure 3.8.

```
(x - y) + ((x - y)/z) + y
```

Figure 3.8: Derived phenotype from sample chromosome

Table 3.2: Mapping of a sample chromosome through a sample grammar

Codon % No. Choices	Choice	Grammar Production
13 % 2	1	`<b> + <Rule_2>`
4 % 3	1	`x - y`
5 % 2	1	`<b> + <Rule_2>`
8 % 3	2	`(x + y)/z`
16 % 2	0	`<a>`
1 % 3	1	`<y>`

GE codons are intrinsically polymorphic, with the exact meaning of any single codon being affected by the preceding codons. This means that the mapping for the second codon on the chromosome is wholly dependent on the value of the first codon. Similarly, the mapping for the third codon will be dependent on both the first and second, and so on. If the value of the first codon in the example chromosome from Figure 3.7 were to change from an odd to an even number, it would have a drastic effect on the derived phenotype. This is known as the GE Ripple Effect [122], and is discussed in greater detail in Section 6.3.7.

3.6.1. Maximum Codon Size

Ch.A is seeded with random integers throughout (random number generation in Python is accomplished using the Mersene Twister random number generator (RNG) [123]). Each codon in the chromosome is selected from within a pre-specified range of numbers – between 0 and the maximum permissible value, known as "`CODON_A_SIZE`". `CODON_A_SIZE` must be carefully selected such that it allows for selection of the full range of grammatical possibilities (i.e. the maximum number of non-terminals for any production rule must be less than or equal to `CODON_A_SIZE)`, in order for the representation space to be properly explored. In general, `CODON_A_SIZE` does not influence the time taken to complete an evolutionary run. The recommendation is, thus, to set the `CODON_A_SIZE` as high as possible to introduce as little bias as possible from the codon value itself [124].

For example, consider a grammar written that would select a number from 1 to 100 in equal increments of 1. If the maximum codon size were to be set to 9, then the only available solutions from this grammar would be 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10[1]. In this instance, `CODON_A_SIZE` would need to be set at a minimum of 99 in order to cover the entire set of numbers. However, if it were set at 100 this would essentially mean that the probability of the grammar selecting 0 would be twice as great as any other number as 100 % 100 = 0. This introduces a bias into the grammar towards selecting the outcome 0. Due to this, `CODON_A_SIZE` should be set either at the maximum number of non-terminals across all production rules in the grammar, a multiple thereof, or else some extremely high number so as to minimize bias.

For the first generation, Ch.B is seeded with a uniform random number throughout (i.e. each codon in the chromosome will be identical for each individual in the first generation). The pre-specified maximum permissible range for Ch.B (`CODON_B_SIZE`) is equal to the number of potential section sizes available. If there are 127 different section sizes available, then `CODON_B_SIZE` will vary between 0 and 126. Values for Ch.B are taken directly from the chromosome (i.e. there is direct mapping between the genotype and phenotype for Ch.B).

3.6.2. Parsing of Phenotype Information

For each individual, Ch.A and Ch.B are passed through the grammar to create a phenotype in the form of an executable Python program (a sample phenotype is shown in Appendix A.2). Once this program is executed, a list of information is then constructed for each individual in the form of a Python dictionary. An instance (individual object) of the "Individual" class is created for each individual in the population. Each individual will have the following unique characteristics:

i. A phenotype, in the form of an executable python program
ii. Ch.A, in the form of an integer array
iii. Ch.B, in the form of an integer array
iv. Number of used codons from Ch.A, i.e. the amount of the chromosome that is required to generate the individual
v. Number of used codons from Ch.B, i.e. the amount of the chromosome that is required to generate the individual

[1] Since python indexes from 0, a codon value of 0 would return the number 1; a codon value of 1 would return the number 2, etc.

Once both chromosomes have been passed through the grammar to generate a phenotype, this phenotype is executed in order to generate a graph object. A graph object is a dictionary of information about both the locations and characteristics of nodes and edges (connections between nodes). Any amount of information about nodes or edges dictated in the grammar can be stored in this dictionary and can be then returned for later use in the analysis section of the program. The information contained in the graph object is extracted to form separate dictionary lists of nodes and edges for each individual, which are then added to the previously listed knowledge of the individual's phenotype and chromosomal data. The list of information on each individual will now additionally contain:

vi. A list of all graph nodes known as the *node list*, with each node comprising of:

- Node id (an integer used to identify that particular node)
- X- coordinate
- Y- coordinate
- Z- coordinate
- Node label (i.e. "loaded", "fixed", etc.)
- Node loading vector

  Loads are applied to nodes in the grammar as a three-dimensional vector of forces in the x, y, and z directions. For example, a load vector of

  ```
  [0, -100, 0]
  ```

  would indicate a load of 100N acting vertically downwards in the y direction. If a node has no load applied to it, the loading vector is set to `None`. Loading conditions are indicated in the grammar itself. Using this implementation, it is possible to specify different loading conditions for any number of nodes in a structure.
- Node fixing conditions

  In a similar manner to the loading vector, node fixing conditions are applied as a three-dimensional array of Boolean operators, each indicating the presence or absence of a fixing in a particular dimension. For example, the fixing array of

  ```
  [False, True, True]
  ```

would indicate support in the y and z directions, but no support in the x-direction. This would be consistent with a rolling bearing for a simply supported beam. Similarly to the loading conditions described above, fixing conditions are also indicated in the grammar itself. Using this implementation, it is possible to specify different fixing conditions for any number of nodes in a structure. Since the focus of this thesis is on truss structures, no fully fixed supports are used; only pinned supports are available.

vii. A list of all graph edges known as the *edge list*, with each edge comprising of:

- Edge id (an integer used to identify that particular edge)
- Node id for point A of edge
- Node id for point B of edge
- Section id (an integer corresponding to a specific codon from Ch.B, used to identify the section size of that particular edge)
- Edge length
- Section diameter
- Section thickness
- Cross-sectional area
- Material density
- Material unit weight (mass per meter)
- Element mass
- I (second moment of area)
- Young’s modulus
- Maximum compressive stress
- Genome id (the index of the corresponding gene in Ch.B responsible for that particular edge)

Once the full lists of nodes and edges have been generated, the structure is analyzed.

3.7. Structural Analysis

Analysis of generated designs is completed using the free open-source finite element analysis program SLFFEA [116]. Previous studies by the author [61] have found this analysis method to be fast, reliable, and accurate in its calculations. Each individual in the population is analysed in the same manner using a simple static truss analysis method. The analysis program does not pass judgement on the structure. There is no knowledge of the limitations of the materials at this stage, only the observed state of the structure after loading has been applied is described.

SLFFEA has a number of drawbacks, however. It cannot correctly model cables, and the orientation of members must be specified (i.e. if I beams were to be used each member would need to be given a specific orientation with regards to the global origin). Consequently, any loading on beam members themselves (rather than on nodes) is conducted according to the local orientation of the member rather than the global orientation, meaning that if the beam members were not oriented correctly, vertical loads would need to be amended to account for the change in global coordinates. These limitations are not an issue in this study, as:

1. Cables are not used in this thesis;
2. Loading is only applied to node locations by default;
3. Local orientation of members is bypassed by the use of symmetrical, circular, hollow sections.

3.7.1. SLFFEA Input File Format

The input file required for this program allows for a number of options with regard to the analysis of the structure. Information (as described in Section 3.6.2) on specific physical attributes of each individual is written out to a specialised input file for use by SLFFEA. An abridged sample input file for the structure shown in Figure 3.9 can be seen in Appendix B.1 (in practice all 157 available sections are listed but for the purposes of brevity only the six that are used are shown here).

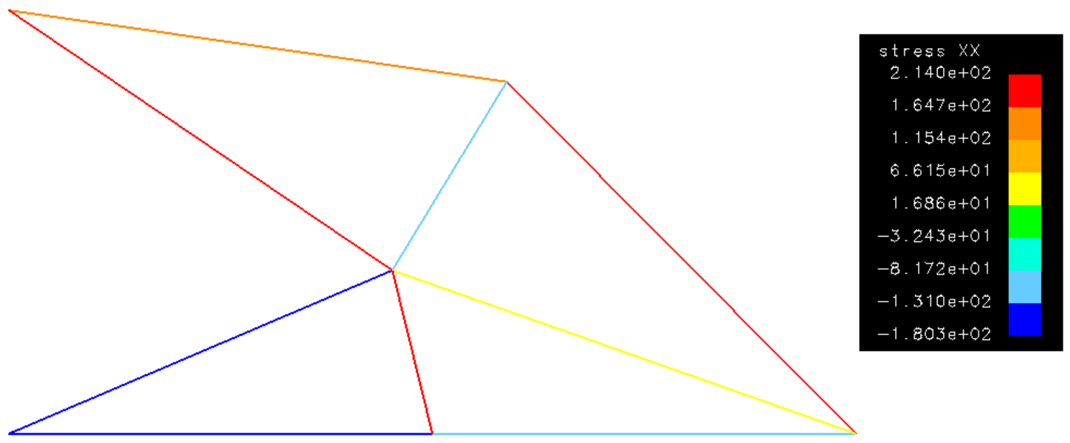


Figure 3.9: Example truss result from SLFFEA analysis. Compression members shown in blue, tensile members in red

The input file is executed using SLFFEA's specialised truss analysis method "ts". A full analysis of the SLFFEA input file includes the following:

- The number of elements and nodes in the structure
- The total number of available materials and sections
- A full list of all available materials and sections, with specific details on each:
    - Section id number (which will correspond to the section id of each particular member as dictated by Ch.B)
    - Young's Modulus
    - Density
    - Area
- A full list of all structure elements, with each edge containing details of:
    - Element id number
    - Element connectivity in the form of which node id's are connected, e.g. [1, 4] would connect nodes 1 and 4.
    - Section id number
- A full list of all nodes, with specific details for each node:
    - Node id number
    - Node z, y, and x coordinates
- Prescribed node displacement in the x, y, and z directions. Setting this option to 0 for any node will fix that node in that direction by providing a reaction and restraining

movement. Using this option alone will provide a pinned support in any of the three available dimensions.

- Point loading for specific nodes including:
    - Node id
    - Point load options in the x, y, and z directions

3.7.2. Parsing of SLFFEA Analysis Results

Execution of the analysis program will instantly generate a separate results file (as shown in Appendix B.2), which is then parsed to extract the relevant information. The resulting files begin with a full list of identical information to the input file but with the following differences:

- Node coordinates are now the displaced coordinates after loading analysis.
- Prescribed node displacements in the x-, y-, and z-directions now detail the amount of displacement in each direction for each node after loading.
- Point loading for specific nodes now includes both applied loading and reactions to said loading. Reactions act on whichever nodes were fixed and are applied similarly to the loading description from Section 3.6.2.
- Stress results for all elements in the structure are shown. These are the primary values, along with the displacements of each node, which are used by the fitness function to determine the overall fitness of each design. Since truss joints are effectively pinned, there are only axial forces present in the members and as such there are no moments present in the analysis of the results.
- Strain results for all elements in the structure are enclosed, but these results are not currently used.

These analysis results are now added to the dictionary lists of information for the nodes and edges of the individual. The aforementioned *node list* is now updated to add information about the displaced co-ordinates of each node, while each edge in the *edge list* is updated to include the stress conditions of that edge. Once all information about the structure has been gathered, the structure is passed through the fitness function.

3.8. Fitness Function

The fitness function is one of the most important aspects of the evolutionary process. It translates a physical design into a single (or multiple in the case of multi-objective optimisation) metric by which each design can be judged and ranked relative to its generational peers. Objective design metrics such as structure self-weight or cost are included in the fitness function, as are analytical values of material stress and strain, nodal deflection, and design constraints. The results are then analysed, and judgement is passed on every aspect of a design.

The fitness function combines a number of specific design constraints, which are taken directly from the relevant design codes of practice [59, 60, 119, 57, 58]. These constraints are viewed as hard limits; if any aspect of a structure fails a single constraint, that individual is deemed to have failed and is such unfit for purpose.

3.8.1. Nodal Deflection

The maximum deflection of the structure is obtained by calculating the vertical displacement of each node in the structure after loading analysis. The maximum displacement of any single node is set as the maximum deflection of the structure. If any single node deflects in excess of the pre-specified deflection limit, then the design is deemed to have failed. Deflection limits are taken from design codes of practice [59, 60, 57, 58] and are as Eq. 3.2 for simply supported trusses and Eq. 3.3 for cantilevered trusses unless specified otherwise.

$$\delta_{max}\text{-} = \frac{Span}{250} \qquad \text{Eq. 3.2}$$

$$\delta_{max}\text{-} = \frac{Span}{180} \qquad \text{Eq. 3.3}$$

3.8.2. Member Stress – Tension and Compression

Member stress is either tensile or compressive. All members are checked against their respective stress limits. The maximum tensile stress limit is set at 215 N/mm^2 for S355 steel [57, 58, 59, 60]. Stress limits for members in compression are governed by the axial compression limits as detailed in Section 3.3. The length of the member in question is compared against the available effective lengths and is rounded up to the nearest available figure. For example, a 3.21 m long member with an outside diameter of 114.3mm and a thickness of 5mm would be rounded up to 3.5m and be given a maximum compressive resistance P_c of 343 kN. When this figure is combined with a cross-sectional area of 1720 mm^2, a maximum permissible compressive stress of 199.4186 N/mm^2 is obtained.

3.8.3. Local Buckling of Compression Members

Members in compression have the additional constraint of local Euler buckling imposed on them. The compressive force in the member is compared against the Euler buckling limit, as given by the following equation:

$$\frac{\pi^2 EI}{(KL)^2} \qquad \text{Eq. 3.4}$$

where:

E = Young's Modulus

I = Second Moment of Area

L = Length

K = Effective length factor (for pinned-pinned structures this is 1).

To take the same example as given in Section 3.8.2, the Euler buckling limit would be given as:

$$\frac{\pi^2 EI}{(KL)^2} = \frac{\pi^2 * 210{,}000 * 2570000}{(1 * 3210)^2} = 516{,}942.33N \qquad \text{Eq. 3.5}$$

Given a maximum permissible compressive stress of 199.4186 N/mm^2, the maximum permissible compressive force in this member would be 343,000 N, well below the limit for buckling. In this particular instance, buckling would not be a consideration over pure compressive resistance, and the element would fail in compression, before buckling became an issue.

3.8.1. Load Path/Michell Number

Along with the imposed constraints, the load path (also known as Michell's number [5]) of each individual structure is calculated. This is given by Zegard *et al.* [125] as:

$$Z = \sum_i |F_i|\, L_i \qquad \text{Eq. 3.6}$$

where F_i is the axial force in member *i*. Since the load path does not require any information on the member section sizes, it is independent of section geometry and, thus, gives an accurate assessment of the efficiency of the topological layout of the structure [5, 125]. Minimization of the load path indicates an improvement in structural topological efficiency.

Implementation of Fitness

The fitness of a solution is a numerical representation of the suitability of the solution for the defined task. Differing fitness values for separate individuals will indicate differing levels of performance. There are a number of options by which the fitnesses of all individuals in a population can be compared. The most popular fitness value to use is that of the self-weight of the structure [7]. This is calculated by simply summing the individual masses of each element in the design. Other options include the maximum nodal deflection of the structure, or the overall strain energy/compliance of the structure (as given in Eq. 2.1). Unless otherwise specified, the evolutionary objective is the minimisation of the chosen fitness value.

If a structure passes all constraints imposed upon it, then it is deemed to be fit for purpose, and a normal fitness value (or values) is returned. If, however, any single constraint (or multiple constraints) fails, then the fitness of the individual is appropriately altered such that it becomes less likely to pass on genetic material, and the entire design is deemed unfit. Design constraints and treatment of unfit individuals is covered in greater detail in Chapter 5. Once the fitness has been calculated, the final verdict is returned, and the individual is updated with the following fitness information:

- A numeric value for the total fitness of the individual
- A Boolean value indicating whether or not the individual is fit (this allows for easy classification of fit/unfit individuals)

3.9. Selection, Variation, and Replacement

Once a full population has been evaluated, the individuals are ranked by fitness. Single objective optimisation sorts the population in order of decreasing fitness, with the individual with the lowest fitness value being designated as the best individual of that generation. If multiple objective optimisation is being used, a fast non-dominated sorting of all individuals in the population is done using the NSGA-II algorithm [84, 56]. Generational replacement with elites is used throughout this study (as described in Section 2.2.2), whereby the entire generation is replaced with a new version at each step, with only a select few elites being carried over unchanged [34]. Elites comprise a small subset of the best of the population and are copied (so that the originals still remain in the overall population) and stored separately to the entire population, out of reach of variation operators, thereby ensuring that the fittest individuals survive unaltered between generations. In this study, the elite size (i.e. the number of elites) is set at 1% of the population size. The entire population (with the original copies of the elites still included) is then crossed over and mutated to form the next generation.

3.9.1. Selection

Traditional randomised tournament selection, the most popular selection mechanism used with Genetic Algorithms [126], is used in this study. Groups of individuals are randomly selected from the original population, with a tournament size of either 3 individuals or 1% of the population (whichever is greater), and the best individual is then returned. A new population of parents is then generated, of equal size to the original population, which is populated with winners from the tournaments. Since random selection is used to select individuals for the tournaments, the same individual could be selected multiple times, while other individuals may never be included in a tournament. One way to counteract this would be to use a Fair Tournament [34]. In a fair tournament, each individual in the population is guaranteed a chance to enter at least one tournament. Once a number of individuals are selected for a tournament, they are not permitted to re-enter another tournament (regardless of who won) until *all* remaining individuals have been entered into tournaments. Once every individual has partaken in one tournament, the remainder of the parent population is generated by randomly selecting individuals from the original population for the remaining tournaments.

The advantage of using a fair tournament over traditional randomised tournament selection is that it gives every individual in the population a chance to enter a tournament. This gives a better spread of parents for crossover, but since EAs are heuristic search processes a randomised tournament selection is sufficient.

3.9.2. Crossover

Once the parent population is fully generated via the selection process, single-point crossover is performed on random pairs of parents. Crossover probability is fixed at a rate of 75% for all experiments performed in this study (i.e. if two parents are selected, there is a 75% chance that they will be crossed over (otherwise they are added to the child population un-altered)). If one chromosome is used, crossover occurs as in traditional GE [122]. If two chromosomes are used, then either the Ch.As or Ch.Bs of both parents are chosen at random and crossover continues as usual. Crossover of information between Ch.A of one individual and Ch.B of another individual is not permitted. A crossover point is then randomly selected from within the range of used codons of whichever parent has the shorter used chromosome (as shown in Eq. 3.).

$$0 < crossover\ point < \min(P1.used, P2.used)$$
Eq. 3.7

If crossover were to occur based on the range of used codons from the parent with the longer chromosome then there is a possibility that the crossover point would be beyond the range of the used codons of the parent with the smaller chromosome. Therefore, the chromosome of the smaller parent would be carried over unchanged to the mutation phase.

Once a crossover point has been uniformly generated, the selected chromosomes of both parents are split in two: the head of the chromosome (from the first codon up to and including the crossover point), and the tail (from the codon immediately following the crossover point to the end of the genome). The *head* of parent 1 is then spliced with the *tail* of parent 2, and vice versa, creating two new individuals (as shown in Table 3.3). The unmodified chromosome is linked to the crossover chromosome whose *head* was spliced with the *tail* of the other chromosome, i.e. if the head of P1.A is spliced with the tail of P2.A, then P1.B will remain linked to the original P1.A head.

Table 3.3: Single-point crossover of two chromosomes is performed on one chromosome only across both individuals

If crossover is on Ch.A	If crossover is on Ch.B
P3.A = P1.A.head + P2.A.tail	P3A = P1A
P3B = P1B	P3.B = P1.B.head + P2.B.tail
P4.A = P2.A.head + P1.A.tail	P4.A = P2.A
P4.B = P2.B	P4.B = P2.B.head + P1.B.tail

An example of this process is presented below. Consider a problem with two chromosomes per individual. Maximum chromosome length is 20 codons, and the range of all codons is between 0 and 100. The amount of used codons per chromosome is variable. Two randomly sampled parents, P1 and P2, each have two chromosomes, giving us four chromosomes in total: P1.A, P1.B, P2.A, and P2.B (Figure 3.10).

Parent	Chromosome Type	Chromosome
P1	P1.A	89, 93, 97, 88, 89, 9, 33, 36, 84, 24, 58, 15, 88, 61, 33, 40, 90, 9, 89, 11
	P1.B	45, 23, 69, 16, 81, 84, 67, 15, 1, 49, 74, 96, 46, 9, 50, 60, 18, 37, 54, 96
P2	P2.A	71, 49, 85, 43, 67, 91, 78, 44, 7, 48, 31, 41, 25, 30, 5, 78, 78, 29, 29, 66
	P2.B	55, 100, 91, 4, 69, 89, 33, 21, 8, 80, 44, 7, 52, 39, 68, 31, 15, 69, 20, 56

Figure 3.10: Two sample parents, each with two chromosomes. Used codons are green, unused codons are red.

A random selection decides that crossover shall be done on the Ch.A of both parents, and the crossover point is then selected from the shorter of P1A and P2A. In Figure 3.10, P1.A has 17 used codons, whereas P2.A only has 13 used codons. If a crossover point greater than 13 were to be selected, that would leave P2.A unchanged. The crossover point in this instance is randomly selected as 7 (i.e. crossover will be performed after the 7th codon on the chromosome). Once the crossover point is established, P1A splits into P1A.head and P1A.tail, and likewise for P2A.

Parent	Chromosome Head	Chromosome Tail
P1.A	89, 93, 97, 88, 89, 9, 33	36, 84, 24, 58, 15, 88, 61, 33, 40, 90, 9, 89, 11
P2.A	71, 49, 85, 43, 67, 91, 78	44, 7, 48, 31, 41, 25, 30, 5, 78, 78, 29, 29, 66

Figure 3.11: Crossover point splits chromosomes into heads and tails

To conclude the example, the Ch.A of both children now become:

Child	Chromosome Head	Chromosome Tail
P3.A	89, 93, 97, 88, 89, 9, 33	44, 7, 48, 31, 41, 25, 30, 5, 78, 78, 29, 29, 66
P4.A	71, 49, 85, 43, 67, 91, 78	36, 84, 24, 58, 15, 88, 61, 33, 40, 90, 9, 89, 11

Figure 3.12: Child A chromosomes after crossover is completed

Once crossover has been completed on the parent individuals, the remaining chromosome of each parent is linked with the original head of that parent. The final children that are produced are:

Child	Chromosome Type	Chromosome
P3	P3.A	89, 93, 97, 88, 89, 9, 33, 44, 7, 48, 31, 41, 25, 30, 5, 78, 78, 29, 29, 66
	P3.B	45, 23, 69, 16, 81, 84, 67, 15, 1, 49, 74, 96, 46, 9, 50, 60, 18, 37, 54, 96
P4	P4.A	71, 49, 85, 43, 67, 91, 78, 36, 84, 24, 58, 15, 88, 61, 33, 40, 90, 9, 89, 11
	P4.B	55, 100, 91, 4, 69, 89, 33, 21, 8, 80, 44, 7, 52, 39, 68, 31, 15, 69, 20, 56

Figure 3.13: Fully complete child individuals after crossover

Once the interim population is fully comprised of new children produced from crossover, mutation is performed on all children.

3.9.3. Mutation

Mutation is performed on all individuals in the interim population before the child population is complete. Again, if a single chromosome is used then mutation continues as usual [34]. However, if dual chromosomes are being used, then either chromosome is selected at random, upon which mutation is then performed. Mutation in GE is performed on a per-codon basis [122], so there is no need to look at the length of used chromosome. All experiments in this study use a mutation probability of 1%. Iteration is performed over all codons in the chromosome, with each codon being given a probability p = 1% of mutating. Mutation of a codon simply replaces the codon in question with a randomly generated integer from within the range of `CODON_A_SIZE`, as detailed in section 3.6.1.

3.9.4. Replacement

Once mutation of all individuals in the interim population is completed to form the child population, the unchanged elites (which were copied from the original population, as described in Section 3.9) are re-introduced to the child population. This generates a new population, which is 1% larger than required (due to the addition of the elites from the previous generation). This new population is then re-evaluated, and the top 100% of the total population is transferred to the next generation, while the remaining 1% is discarded.

3.10. Iteration and Termination

Populations of individuals are iteratively replaced and augmented in the manner described between sections 3.5 and 3.9, either for a pre-specified number of generations or until a stopping criterion (such as convergence of the population towards a single solution, preventing further evolution) is reached. At the very end, the best overall individual (the single best remaining individual in the final generation) is presented as the best evolved solution for that particular run. Due to the use of elites, this individual is guaranteed to be the best (or equal best) solution generated during the entire run.

3.11. Summary

This chapter described the process by which all experiments in this thesis were performed. A detailed breakdown of the evolutionary method was given, with information on all aspects of the Grammatical Evolution algorithm used. A description of the mapping process was presented, along with examples of grammar derivation using this mapping process. The fitness function was explained, including the method of analysis of structures. A definition of the initialisation was given, and the chapter concluded with details on the selection and replacement operators used, including mutation and crossover. The remaining chapters run a series of experiments based on this methodology, in order to answer the research questions presented in Section 1.2.1.

Chapter 4. DO-GE – Dual Optimisation in Grammatical Evolution

As noted in Chapter 2, theoretically optimal structures can only be generated when optimisation of the structural shape, topology, and member sizing are done simultaneously. A fundamental question, which forms the basis of one of the main research questions of this thesis must, therefore, be asked as to whether or not this is truly possible. Previous research in this area has attempted simultaneous optimisation [72, 85], but different methods were used for optimisation of structural topology and member sizing. Both Darwin [18] and Turing [16] noted (albeit through different applications) that the development of an individual (or indeed of a species) is highly dependent on the environment it inhabits. The interaction between the individual and its environment is what forms the individual itself. Thus, truly simultaneous optimisation should undertake all optimisation tasks using the same method at the same time, as each has an effect on the development of the other as they evolve in the same environment. A single-stage optimisation approach also has a number of theoretical advantages over a two-stage approach of optimising topology and element sizes separately. With a single stage approach, a large number of designs can be assessed in a relatively short space of time, whereas a two-stage approach would be slower and would fail to allow for interactions between structural topology and element sizes parameters.

GE operates by evolving chromosomes, as described in Chapter 3. Traditional GAs have been used for both element sizing optimisers and structural topology optimisation [127, 69, 63, 14]. In each case, a distinct chromosome is used to evolve either the element sizings or the overall structural shape/topology. It was theorized in Chapter 2 that simultaneous evolution of structural topology and member sizing is possible by assigning a unique chromosome to each task and evolving both simultaneously. This hypothesis is compared against benchmark problems from the literature. Since sizing optimisation can be seen as a deterministic problem when given an initial configuration, this heuristic simultaneous optimisation method is also compared against a deterministic sizing optimiser which directly calculates the requisite section sizes for each individual in a population.

In addition to this, a review of the literature has found that unrealistic element sizes and material properties are commonly used for discrete optimisation. As noted in the research questions in Section 1.2.1, optimisation results ignore critical knowledge of section geometry and orientation, along with optimised results which exceed the current manufacturing precision of structural components. To this extent, the use of commercially available construction materials as design elements and design codes of practice as evolutionary constraints are also examined. A number of basic structural grammars are created to test these questions, and the evolved solutions are compared against benchmark problems from the literature. Since the benchmark problems all use Imperial units, all units shown in this chapter are Imperial, unless stated otherwise. This allows for better comparison between the literature and the work presented herein.

This chapter introduces a new method of discrete topology optimisation: Dual Optimisation in Grammatical Evolution (DO-GE). While existing structural optimisation methods in GE [56, 73, 61] primarily focus on a structural topology scale (optimisation of the structural layout), optimisation of individual element sizes is also possible [7, 13, 107]. The combination of both topology and sizing optimisation is established [7, 73, 12, 13, 62], but the use of both standard construction elements and compliance to design codes of practice in the process is novel. Standard practice is to optimise element sizings by specifying the required cross-sectional area. While this gives theoretically optimised results, the output is of little use to structural engineers, since in practice trusses are constructed using structural elements with pre-set cross-sections and geometries. This highlights a fundamental weakness in traditional sizing optimisation methods: by omitting knowledge of section geometry and orientation, it is not possible to include accurate buckling calculations as a constraint for structural design and, thus, structures cannot be designed according to standard codes of practice. Furthermore, current sizing optimisation methods exceed the manufacturing precision of structural components. The approach presented in this chapter addresses this deficiency by allowing for any number of standard commercially available construction elements [118] to be specified for any elements within a design, leading to code-compliant construction-ready designs, which truly represent the evolved form.

The remainder of this chapter is structured as follows. Section 4.1provides an introduction to the topics discussed in the chapter. Section 4.1 describes the DO-GE method, including the implementation of a second chromosome for member sizing in Section 4.1.1. The use of design constraints and codes of practice is covered in Section 4.1.2. A number of benchmark numerical examples from the literature are detailed in Section 4.2, and comparisons between deterministic methods are made in Section 4.4. The implications of these experiments are discussed in Section 4.5, and conclusions are drawn in Section 4.6.

4.1. The DO-GE Method

DO-GE differs from regular GE by utilising two separate chromosomes simultaneously. As described in Chapter 3, the DO-GE method creates two separate integer array chromosomes: Chromosome A (Ch.A), which governs the topological form of the structure, and Chromosome B (Ch.B), which assigns material section sizes to each individual edge in the individual. Ch.A operates in the normal GE fashion, controlling the derivation of the grammar which details the layout of the trusses. Ch.B is passed in as an argument to the derived program (as shown in Appendix A.2), which itself creates a graph object through its execution. Each graph edge is then assigned a section id from a corresponding gene in Ch.B. The list of edges is arranged in the order in which the edges are generated, meaning that if the edge order changes, the solution will correspondingly change, as the mapping from Ch.B to the phenotype will have changed. This is not a concern, however, as the creation of edges is explicitly controlled. While Ch.A is fixed in length in the experiments described in this chapter (variable-length Ch.As are discussed in Chapter 6), a variable-length chromosome is required for Ch.B due to the variable nature of the number of edges in each individual as the truss type changes. Structural analysis of individuals is then carried out using the free, open-source finite element modelling program SLFFEA [116]. Previous work [61, 56] has shown this method to be both reliable and fast in analysing truss structures produced by a grammar. The processing steps of each individual in a population are shown in Figure 4.1.

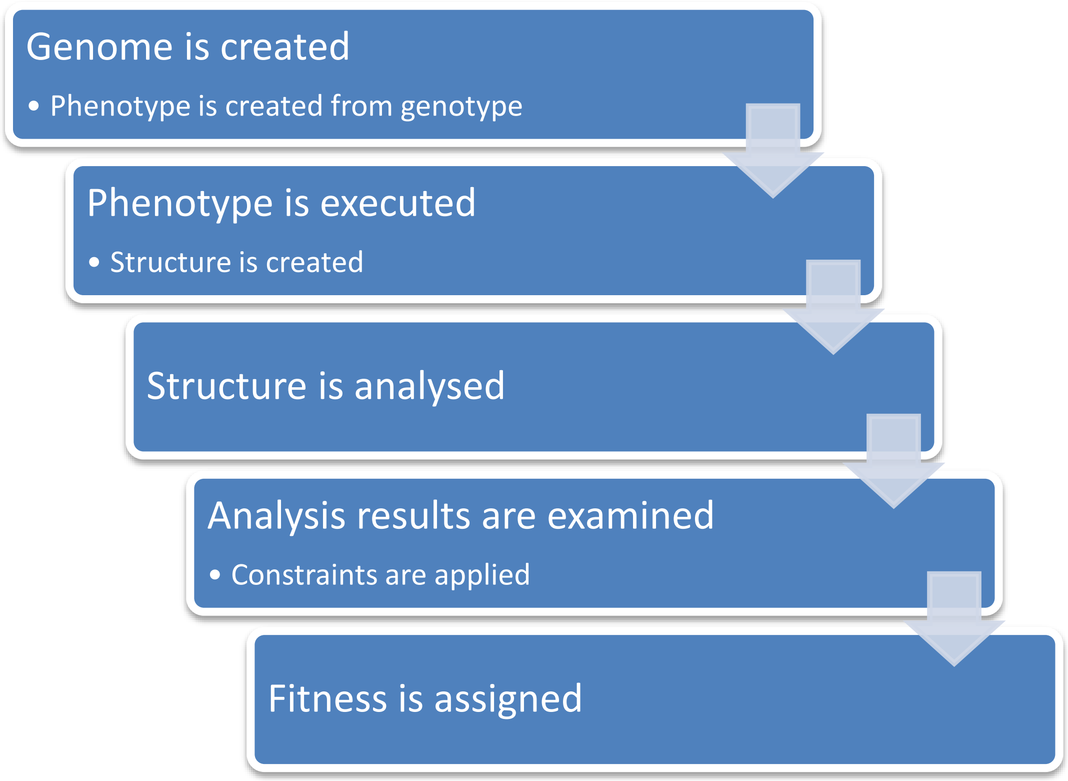


Figure 4.1: Flowchart of the evolutionary process for an individual

The regular genetic search operators used by GE (mutation and crossover) are modified to accommodate the use of two separate chromosomes. First, randomised pairs of parents are selected from the parent population using tournament selection. For each pair, either Ch.A or Ch.B is chosen, and then crossover is performed on the chosen chromosome of both parents, creating a pair of children (the other chromosome is carried over unchanged from each parent). Once the child population is fully created, mutation is applied to either Ch.A or Ch.B for each individual child (choice-independent of the crossover stage), thereby creating the next parent population (as detailed in Section 3.9).

An extra design check for detection of redundant members in truss designs is an important feature of the DO-GE method. If a particular solution is found to contain a member (or any number of members) with zero axial stresses, that solution is then re-analysed without the presence of that particular edge (or edges). If the fitness is improved (i.e. if all constraints such as stress and deflection are still within their limits and the overall structure self-weight has been lowered), then that member is omitted from the solution. Notably, for the experiments described in this chapter, this is not an evolutionary feature. It does not create a change in the chromosome, but instead is a measure that is performed within the fitness function. This is an example of a post-processing measure, any number of which can be run as part of the fitness function after the DO-GE mapping has completed.

DO-GE encourages population diversity by removing individuals from the child population which match other children. A number of checks are performed to ensure that only exact duplicates are removed from the child population:

i. The fitnesses of individuals are compared
   If any two individuals have the same fitness values, this could indicate the presence of duplicate individuals, and further checks are done.
ii. Both Chromosomes are checked
   If two individuals have the same fitness values, then their chromosomes (both Ch.A and Ch.B) are compared to see if they match. If so, then duplicate individuals exist in the population.
iii. The phenotype is checked (locations of all nodes and edges).
   Since GE permits many-to-one mapping, it is possible for two separate chromosomes to produce the same phenotypical individual (i.e. an individual which not only is visually identical to another, but performs identically too, despite being genetically diverse). In order to prevent this, individuals with identical fitness values also have their node and edge lists compared (regardless of whether or not their chromosomes match). If both nodes and edges match, then duplicates exist in the population.

This multi-level checking process catches all possible duplicates including any repeated individuals that may occur due to many-to-one mapping.

4.1.1. Material Selection and Evolution

For all individuals in the randomly initialised first generation, Ch.B is seeded with uniform genes throughout. This means that each individual in the first generation has a single section size applied across all edges, resulting in a greater probability of initial "fit" individuals upon which further evolution will be based. This is a method employed to artificially reduce the initial size of the representation space, thereby forcing the search process to focus on more promising areas. Areas of the representation space that have a lower probability of containing fit solutions (e.g. solutions with very thin members throughout) are quickly removed from the population, thus improving the efficiency of the search process. Once crossover and mutation occur in the creation of the second generation (and all subsequent generations), the Ch.Bs of the population are blended.

Depending on the number of sections supplied for possible selection (n), DO-GE assigns each gene in the chromosome an integer value ranging between 0 and (n-1). Unless stated otherwise, element sizes are taken from Tata Steel charts for S355 Hot-Finished Circular Hollow Sections (CHS) [118]. There are 157 variations of steel members on the list with diameters ranging from 21.3 mm (0.838 in) to 508 mm (20 in). In addition the wall thickness can also vary and any of the 157 possible steel sections can be applied to any element in the truss. The chromosome assigns an integer value from 0 to 156 (representing section ids 1 to 157) to each individual element in the structure, which corresponds to the index of the section on the list. Requisite properties include the cross-sectional area, mass per meter, second moments of area I_x and I_y, and section thickness. Any number of material sections could be included in the materials list, so long as these five necessary variables are provided. The program automatically increases or decreases the range of the chromosome variables based on the length of materials list provided.

4.1.2. Fitness Function Constraints

A number of constraints within the fitness function are placed on the individuals to ensure only appropriate designs are included in the population. These constraints include limits on maximum vertical nodal displacement, member axial stress (tensile & compressive) and Euler buckling loads for compression members. While these constraints are applied in the fitness function, they can be used to modify the selection pressure for each individual. Only a single objective (the self-weight of the structure in the case of single objective optimisation) is passed through as the final fitness value (this method is used by the vast majority of texts cited in this paper and is generally accepted as the standard optimisation goal in structural design). The use of a multi-objective optimiser in this particular instance is unnecessary, as any parameters other than weight that may require optimisation are merely constraints that must be imposed on the design (unlike [68] where minimising horizontal structural displacement is a design priority). For example, there is no quantifiable engineering benefit from imposing a vertical deflection limit of 10mm when building design codes might allow a deflection of 50mm. In the search for minimum structure weight, it is desirable for all constraints to be at their limits in order to find the lightest possible structure. If a constraint is not at its limit, an improvement can be made.

Multiple objective optimisation capability is possible with DO-GE, and indeed previous research [56] has successfully implemented the Non-dominated Sorting Genetic Algorithm (NSGA) II algorithm [84] in handling up to three conflicting objectives with GE. However, in this instance only a single optimisation objective is deemed necessary, as once all constraints have been satisfied, there is little or no performance improvement in trying to further minimise them.

In the case of vertical nodal displacement, a limit of (span/250) is imposed for simply supported truss structures. The positions of each node before and after loading are recorded, and whether the total vertical deflection of any node after loading is greater than the limit of (span/250) then that structure is considered to have failed in deflection [57, 58, 9].

Tensile stress limits are based on the relevant design codes for the use of structural steelwork [57, 59, 60]. A maximum tensile limit of 355 MPa (51.488 ksi) is applied to all members. If any element fails in tension, the structure is considered to have failed overall.

Compressive limits on the materials are based on the material manufacturer's specifications [60, 118]. Compressive resistance limits are given for effective lengths of individual members, and are a function of element size, geometry and end fixing conditions (i.e. fixed-fixed, fixed-pinned, pinned-pinned). These limits are applied to the appropriate elements in the structure and as with the two previous constraints, if any element fails in compression, the structure itself fails. Likewise, the use of Euler buckling limits for elements in compression ensure that designs featuring elements prone to buckling will be penalised. The application of these constraints results in high-displacement, over-stressed designs having a lower chance of passing on genetic material to subsequent generations.

4.2. Design by Code of Standards

All solutions in this thesis are compliant with BS 5950-1:2000 [60]. The design of an individual solution can be described by the following steps.

1. The topology of the structure and the loading/fixing conditions are defined
2. The reaction forces are calculated
3. The internal forces are calculated for all members

   Since all member connections are assumed to be pinned, no moments are present in the structure in accordance with Section 4.10 of BS 5950-1:2000 [60]. As such, the effective lengths of all members are equal to their full length.

4. The material strength p_y is chosen

   All experiments performed in this thesis use S355 cold rolled steel [118]. p_y varies with section wall thickness, in accordance with Table 9 of BS 5950-1:2000 [60]. Alternatively, manufacturer specified-limits can be used [118].

5. Requisite cross-sectional area A^i_{req} for each member *i* calculated such that:

$$A^i_{req} = \frac{p_y}{F_i} \qquad \text{Eq. 4.1}$$

   where F_i is the axial force in member *i*.

6. Member sections are selected such that $A^i_{req} \leq A_{actual}$
7. The internal member forces F_i and stresses S_i are re-calculated with the new section properties
8. Members in tension are subject to stress limits such that:

$$P_t = A_i p_y \qquad \text{Eq. 4.2}$$

   where P_t is the tensile resistance of the member.

9. Members in compression are classified for risk of local buckling subject to Section 3.5 of BS 5950-1:2000 [60]
   a. Sections classified 1-3 are subject to stress limits such that:

$$P_c = A_g p_c \qquad \text{Eq. 4.3}$$

      where P_c is the compressive resistance in the member, A_g is the gross cross-sectional area, and p_c is the compressive strength of the material.

      i. The compressive strength p_c is either obtained from the manufacturer specifications or through Figure 14 and Tables 23 and 24 of BS 5950-1:2000 [60]

   b. Sections classified as Class 4

$$P_c = A_{eff} p_{cs} \qquad \text{Eq. 4.4}$$

      where A_{eff} is the effective cross-sectional area and p_{cs} is the value of p_c taking into account the slenderness ratio and radius of gyration

10. Members in compression are subject to Euler buckling limits, as described in Eq. 3.4.
11. Limits on overall structural deflection are as described in Eq. 3.2 for simply supported trusses and Eq. 3.3 for cantilevered trusses.

If a structure passes all of the above tests, it can be considered to be designed appropriately to code and, thus, is fit for purpose.

4.3. Numerical Examples

The aim of this chapter is not to find fully optimised solutions for absolute minimum required cross-sectional areas in standard structures; the literature contains numerous examples of efficient processes for achieving this. Instead, the focus of this chapter is to show that viable solutions very close to the theoretical optimum can be achieved using standard construction elements. For each benchmark example attempted in this chapter, three sets of experiments are performed:

1. Topology/shape optimisation
2. Sizing optimisation
3. Simultaneous (dual) topology and shape optimisation

For sizing optimisation, a simple GA is used to evolve optimal member sizes for a fixed truss topology. Similarly, a simple GA is used to evolve optimal truss topology/shape for constant member sizes. For dual optimisation, these problems are then once again attempted in order to ascertain the effectiveness of optimising structural topology and member sizings simultaneously. The possibility of removing redundant members is also explored.

Two commonly used problems are analysed from selected papers: a 10-bar cantilever truss (shown in Figure 4.2) and a 17-bar cantilever truss (shown in Figure 4.3) [77, 64, 65, 76].

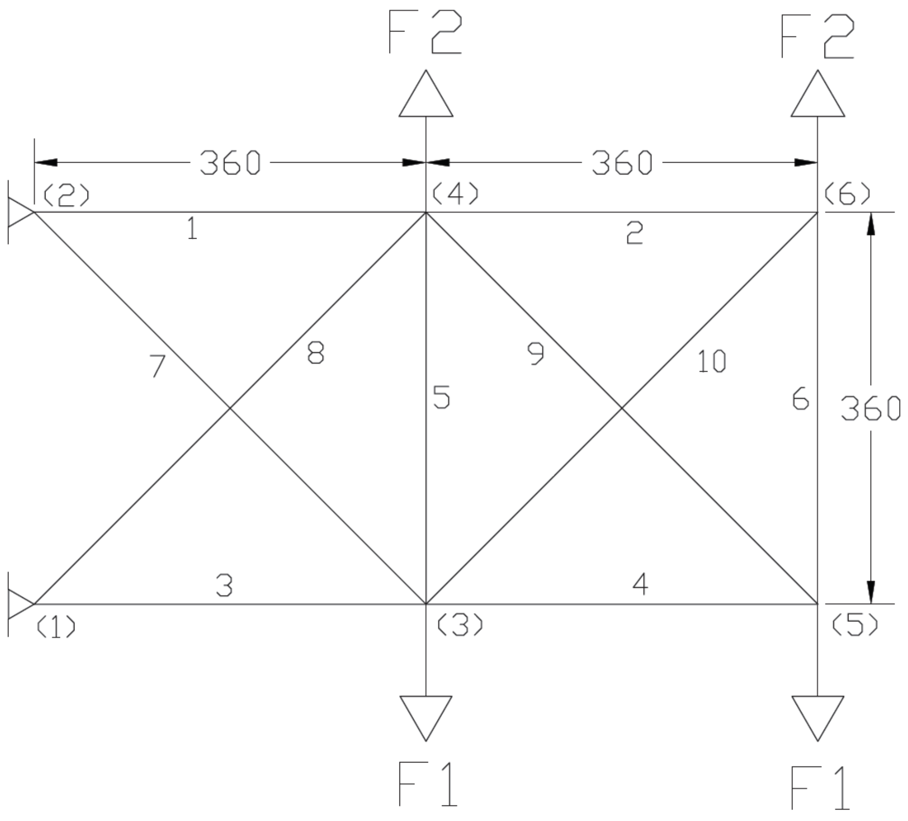


Figure 4.2: Benchmark 10-bar truss problem

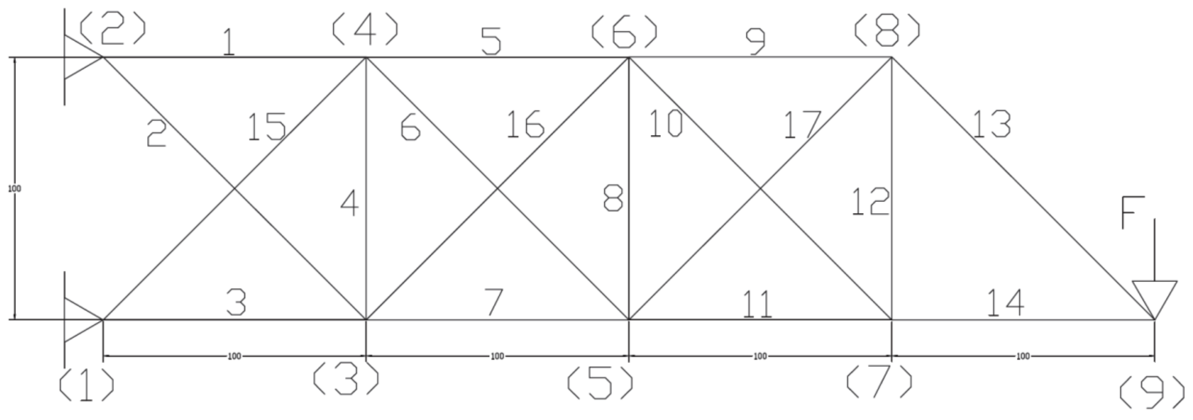


Figure 4.3: Benchmark 17-bar truss problem

Maximum vertical nodal deflection is universally limited to 50.8mm (2 in) in all cases. The examples from these papers differ significantly from the method described in this chapter in that non-standard materials are used in all benchmark examples. All 10-bar truss experiments used aluminium solid sections, while those of the 17 bar truss used steel solid sections. For this reason, all experiments conducted in this chapter were run with two different sets of materials and sections, the properties of which are described in Table 4.1. This provides a level metric across which results can be compared.

Table 4.1: Material Properties

	10 Bar Truss		**17 Bar Truss**	
	Tata steel sections	Aluminium solid sections	Tata steel sections	Steel solid sections
Section type	Circular Hollow Section	Circular Solid Section	Circular Hollow Section	Circular Solid Section
Section sizes	157 standard sizes; diam. from 0.838 to 20in.	350 sections; CSA from 0.1 to 35in^2, increments 0.1in^2	157 standard sizes; diam. from 0.838 to 20in.	350 sections; CSA from 0.1 to 35in^2, increments 0.1in^2
Young's modulus (ksi)	30458	10000	30458	30000
Density (lb/in^3)	0.285	0.100	0.285	0.268
Max tensile stress (ksi)	51.488	25	51.488	50
Max comp stress (ksi)	Manufacturers limits	25	Manufacturers limits	50

Topology Optimisation

Small truss topology optimisation problems such as the 10 and 17-bar trusses can be easily expressed using traditional ground structure topology optimisation methods, and are most notably suited to a canonical GA approach. Using this approach, the presence or absence of each member in the structure is indicated by a gene from the chromosome in binary fashion (as described in Section 2.2.2). Since every gene in the chromosome only has two possible choices (true or false), the maximum codon is set to 1 (i.e. each codon on the chromosome can be either 0, indicating the absence of that member, or 1, indicating the presence of that member). For a ground structure comprising n bars, there are 2^n possible unique topologies in the representation space. However, this figure does not take structural integrity into account, and a large proportion of these solutions will be non-viable (e.g. only one member, no direct load path from loads to supports, all members not connected, etc.). In all truss topology optimisation problems presented here the member sizing is kept constant across all members in the structure. Member sizes can take any value from the indicated section sizes (as described in Table 4.1). The addition of material information means that the total number of potential solutions for a fixed-material topology optimisation problem is:

$$Total\ unique\ solutions = m \times 2^n$$

Eq. 4.5

Material information is added in this case purely so that the viability of solutions can be calculated (structures cannot be analysed by the analysis program described in Section 3.7

without a structural material). For topology optimisation problems such as these, the individual member stresses and overall structural self-weights are of no concern. Only the optimal topological configuration is of interest.

For each load case of the 10-bar truss, there are 2^{10} unique topologies. With the use of CHS materials, 157 different materials gives rise to 160,768 unique solutions. For the 17-bar truss there are 2^{17} unique topologies. With 157 different materials this gives 20,578,304 unique solutions. For traditional ground structure topology optimisation problems such as these, the relatively small number of total unique solutions means that a deterministic full enumeration of the representation space is possible. This is accomplished by evaluating every solution generated by every possible chromosome. For a 10-bar truss the first chromosome would be:

```
[0,0,0,0,0,0,0,0,0,0]
```

indicating complete absence of all members in the structure, while the last chromosome would be:

```
[1,1,1,1,1,1,1,1,1,1]
```

indicating the presence of all members in the structure. The best obtained solution is, therefore, the best possible topology for that problem as all possible solutions will have been evaluated.

All GA-style experiments performed in this chapter use a version of GE which has been modified to run as a basic GA. This method is hereafter called GA-GE.

Sizing Optimisation

Member sizing optimisation for fixed-topology ground structures can be completed similarly to the topology optimisation problem described above. In this instance, however, the topology of the structure is held constant (as shown in Figure 4.2 and Figure 4.3) and the member sizes themselves are described directly by the chromosome rather than the presence or absence of each member. Each codon on the chromosome can then take any value from 0 to the total number of available materials, as described in Section 3.3. In essence, this GA-GE sizing optimisation method is identical to the GA-GE topology optimisation method described above, except in the topology optimisation example there are only 2 materials (0 or 1), whereas for sizing optimisation there are far more materials.

With a fixed topology of n bars and m possible materials, the total number of unique solutions is described as:

$$Total\ unique\ solutions = m^n \tag{Eq. 4.6}$$

In fact, Eq. 4.5 is a special case of Eq. 4.6 where m = 2. For each load case of the 10-bar truss, there are 9.0990599×10^{21} solutions using the CHS materials set, while the 17-bar truss has 2.1394103×10^{37} solutions using the CHS materials set. Since these figures are orders of magnitude greater than those of the topology optimisation problems, full enumeration of these representation spaces is not possible. As discussed in Chapter 1, heuristic techniques can be applied in cases where the problem is too large for deterministic processes to solve. A GA is employed, therefore, to explore the large representation spaces and to evolve suitable solutions.

Dual Topology and Sizing Optimisation

The combination of the two techniques described above is made possible through the use of two separate chromosomes, as described in Chapter 3. Ch.A describes the presence or absence of structural members, while Ch.B specifies the section sizes of those members. Combining these two separate GA-GE methods in such a way gives rise to a vastly increased representation space:

$$Total\ unique\ solutions = 2^n \times m^n \tag{Eq. 4.7}$$

For the 10-bar cantilevered truss described in Figure 4.2, the use of the CHS materials set would, therefore, give rise to 9.3174373×10^{24} unique solutions. Similarly, the 17-bar truss problem has 2.8041679×10^{42} unique solutions. Full enumeration of these spaces is clearly impossible and, as such, a heuristic technique such as an evolutionary algorithm needs to be employed.

Experimental Settings

Experimental evolutionary variables for both the sizing optimisation experiments and the dual optimisation experiments were set at:

- Population Size: 1000
- Generations: 100
- Mutation: 1%
- Crossover: 75%
- Tournament selection, with a tournament size of 1% of the overall population size
- Generational Replacement with elites, elite size of 1% of overall population size

All evolutionary runs were completed 100 times for each different experiment, and the best solution after 100 runs was presented as the best overall solution for that particular experiment. Since the topology optimisation is achieved by full enumeration over the entire representation space, evolution does not occur and no variables act on the system.

4.3.1. 10-Bar Cantilevered Truss: Load Case 1

Load Case 1 of the 10-bar truss problem is as follows:

F1 = 444,800 N (100 kips)
F2 = 0

Topology Optimisation

The least-weight topology for this problem is shown in Figure 4.4:

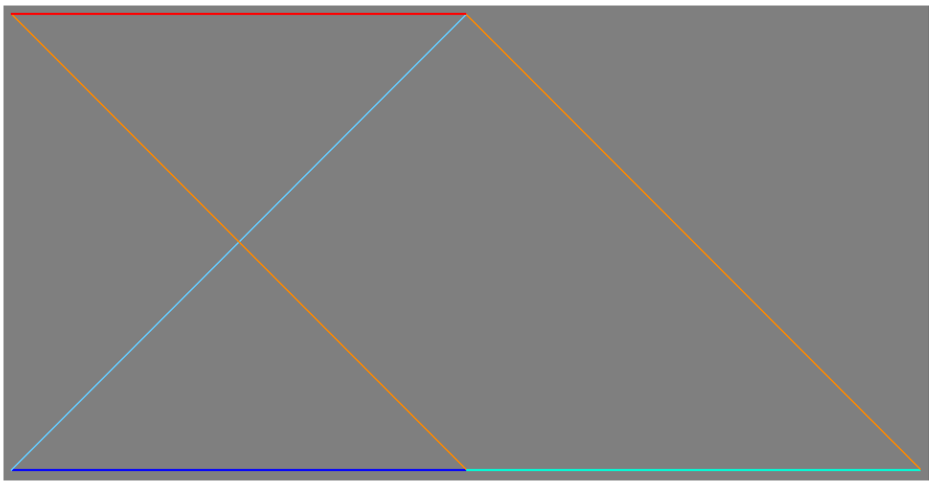


Figure 4.4: Least-weight topology of 10-bar truss, load case 1

This solution matches the optimum topology from the literature [65], and can be considered as the global optimum for this particular topology optimisation problem when uniform materials are applied.

Sizing Optimisation

Lee and Geem [76], Luh and Lin [77], Li *et al.* [64], and Kaveh and Talatahari [65] proposed solutions for a 10-bar planar truss sizing optimisation problem shown in Figure 4.2. The objective is to minimise the cross-sectional areas of all members (subject to given material constraints), such that the self-weight of the structure is minimised.

The results of the material sizing experiments are presented in Table 4.2, where they are compared with the results of recent research solutions, including overall structure weight and individual member cross-sectional areas.

Notably, the research results presented by Li *et al.* [64] and Kaveh and Talatahari [65] use aluminium as the design material, while the research presented in this chapter utilises standard steel sections as the primary material. For this reason, these experiments were run using both the TATA steel CHSs and Aluminium Solid Sections (as shown in Table 4.1). When the GE algorithm makes use of aluminium solid sections, the minimum achieved self-weight was 5163.4 lb for load case 1, exceeding the results of Kaveh and Talatahari by 106.84 lb.

Table 4.2: 10-bar truss, load case 1: Evolved minimum cross-sectional areas (in^2) using GE as a regular GA for sizing optimisation.

Element	**Li *et al.*** [64]	**Kaveh & Talatahari** [65]	**GA-GE**	
			Aluminium solid sections	Using Tata CHS steel
1	30.704	30.3070	33.0	8.184
2	0.100	0.1000	0.2	0.237
3	23.167	23.4340	20.6	7.766
4	15.183	15.5050	18.9	5.828
5	0.100	0.1000	0.1	0.237
6	0.551	0.5241	0.2	0.237
7	7.460	7.4365	9.0	3.642
8	20.978	21.0790	20.6	7.766
9	21.508	21.2290	20.0	7.238
10	0.100	0.1000	0.2	1.533
Weight (lb)	5060.92	5056.56	5163.4	5206.9

With the steel Tata material set, the best evolved self-weight using GA-GE was 5206.9 lb. This is quite a bit higher than the best achieved weights from previous works, at 150.34 lb heavier than that of Kaveh and Talatahari. The implications of these results are discussed in Section 4.4.

Dual Optimisation

For the final part of this experiment, load case 1 of the 10-bar cantilevered truss problem was solved using simultaneous optimisation of structural topology and member sizing via concurrent evolution of two chromosomes.

Progressive stages of the evolution of the best solution are shown in Figure 4.5. The self-weight and load path for each solution is shown for best solutions sampled from the population at regular intervals.

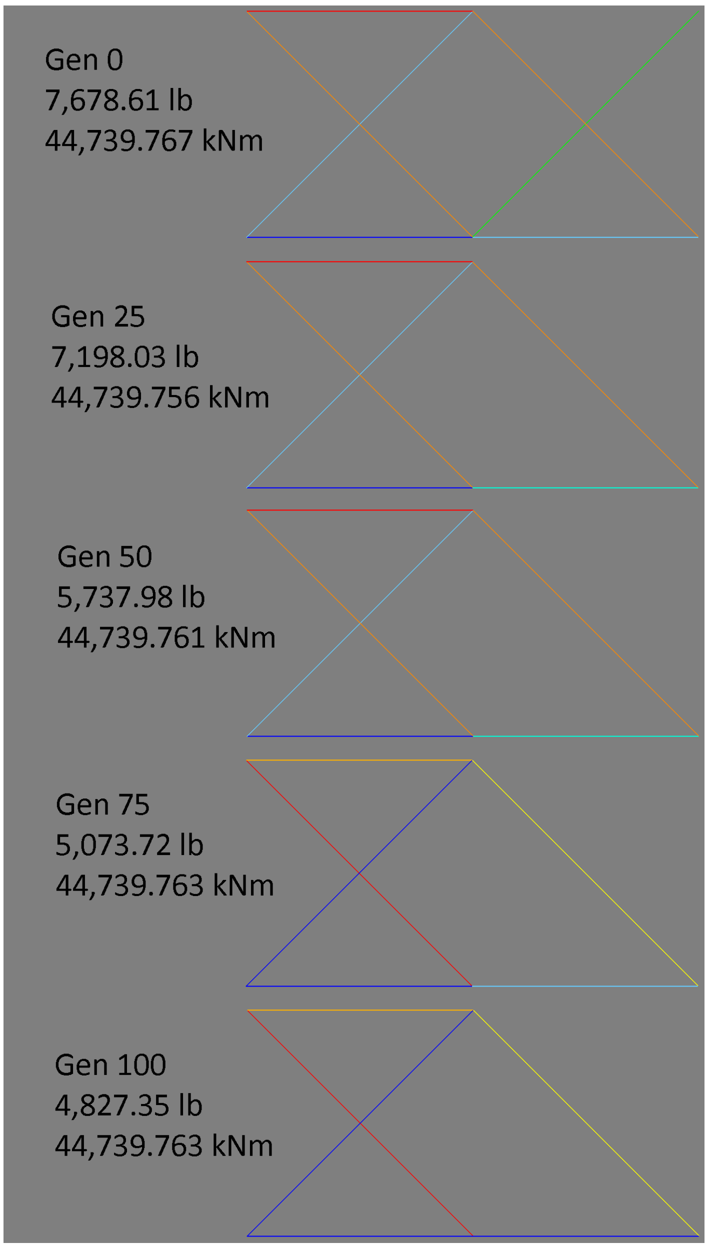


Figure 4.5: Progressive evolution of DO-GE solution for load case 1 of the 10-bar truss problem

What is notable from Figure 4.5 is that the global topological optimum is evolved very quickly, while the optimal member sizings take longer to manifest. Also, since topological changes are minimal, the load path for each generation correspondingly remains constant (or changes minimally).

The evolved optimal topology shown in Figure 4.6 matches the global optimum topology for this problem (as shown in Figure 4.4).

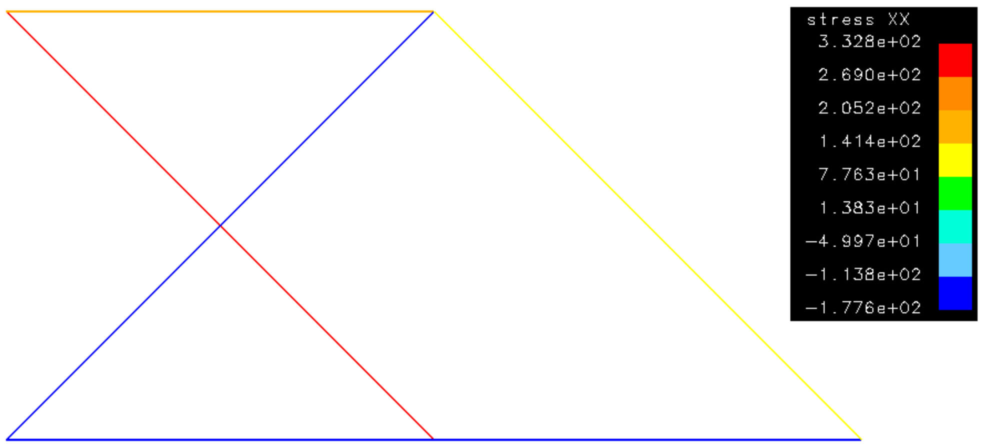


Figure 4.6: 10-bar truss problem, load case 1 - evolved minimum topology

The results of the evolutionary runs, shown in Table 4.3, were found to be much closer to the solutions from the literature.

Table 4.3: 10-bar truss, load case 1: Evolved minimum cross-sectional areas for DO-GE (in^2).

Element	**Li *et al.***	**Kaveh & Talatahari**	**DO-GE**	
			Aluminium solid sections	Using Tata CHS steel
1	30.704	30.3070	31.0	9.200
2	0.100	0.1000	None	None
3	23.167	23.4340	22.7	7.766
4	15.183	15.5050	12.7	5.828
5	0.100	0.1000	None	None
6	0.551	0.5241	None	None
7	7.460	7.4365	6.1	6.600
8	20.978	21.0790	22.8	7.766
9	21.508	21.2290	20.9	9.200
10	0.100	0.1000	None	None
Weight (lb)	5060.92	5056.56	4925.79	4827.35

The application of DO-GE resulted in the evolution of the global optimum truss topology, with the absence of elements 2, 5, 6 and 10 from the structure. This results in a lightweight, rigid truss which improves on the best minimum weight achieved by the literature. Interestingly, the use of the steel Tata material set produces even better results. Again, the implications of this are discussed in Section 4.5.

A graph of the evolution of the best solution for each run for load case 1 is shown in Figure 4.7. For both GE cases, initial convergence is gradual, and further improvements take longer to manifest. DO-GE, however, demonstrates rapid convergence within a very small number of generations.

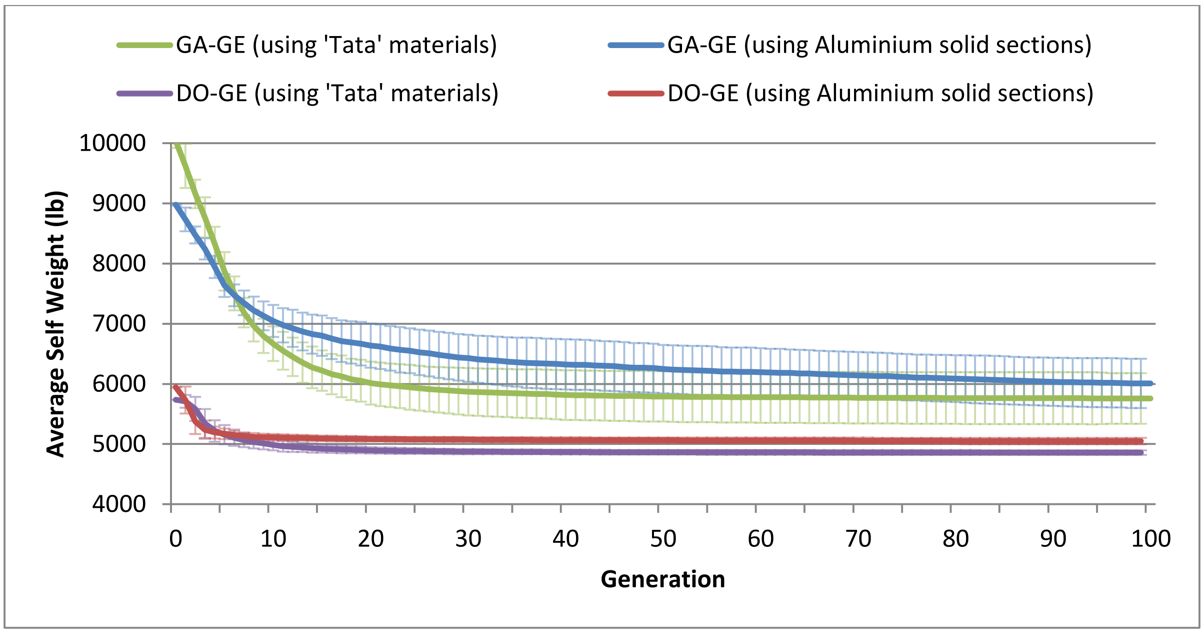


Figure 4.7: 10-bar truss, load case 1. Comparison between regular sizing optimisation (GA-GE) and dual topology and sizing optimisation (DO-GE) shows marked improvement when evolving both simultaneously.

4.3.2. 10-Bar Cantilevered Truss: Load Case 2

Load Case 2 of the 10-bar truss problem is as follows:

F1 = 667,200 N (150 kips)
F2 = 222,400 N (50 kips)

Topology Optimisation

The least-weight topology for this problem is shown in Figure 4.8. As with load case 1, this solution can be considered as the global optimum for this particular topology optimisation problem when uniform materials are applied.

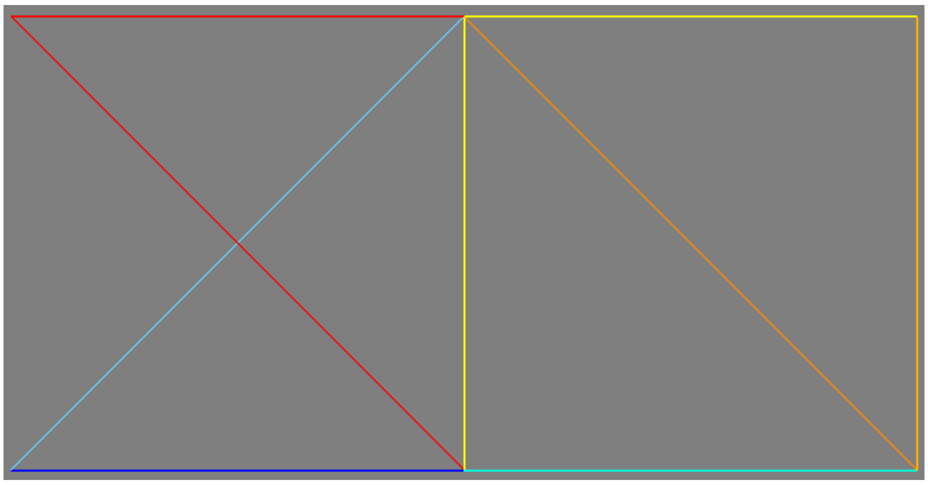


Figure 4.8: Least-weight topology of 10-bar truss, load case 2

Sizing Optimisation

The results of the material sizing experiments are presented in Table 4.4, where they are compared with the results of recent research solutions, including overall structure weight and individual member cross-sectional areas.

Using aluminium solid sections, the minimum evolved structural weight for load case 2 of the 10-bar truss was 4705.8 lb, just 30 lb off the best achieved result from the literature. As with load case 1, the use of a different material set (steel CHSs) generated results far heavier than those achieved using aluminium sections.

Table 4.4: 10-bar truss, load case 2: Evolved minimum cross-sectional areas (in^2), using GE as a regular GA for sizing optimisation.

Element	**Li *et al.*** [64]	**Kaveh & Talatahari** [65]	**GA-GE**	
			Aluminium solid sections	Using Tata CHS steel
1	23.353	23.194	21.3	6.014
2	0.100	0.100	0.2	1.530
3	25.502	24.585	24.2	7.766
4	14.250	14.221	15.6	5.828
5	0.100	0.100	0.1	0.340
6	1.972	1.969	2.0	0.910
7	12.363	12.489	12.7	4.975
8	12.894	12.925	14.1	7.285
9	20.356	20.952	20.7	6.246
10	0.101	0.101	0.1	0.340
Weight (lb)	4677.3	4675.8	4705.8	5137.42

Dual Optimisation

The evolution of the best solution for load case 2 of the 10-bar truss is shown in Figure 4.9. The self-weight and load path for each solution is shown for best solutions sampled from the population at regular intervals.

As with Figure 4.5, Figure 4.9 shows minimal changes to the load path. Since the Michell number is distinct from the geometry of the individual members, changes in section size have no effect on the load path [125]. As such, the load path remains constant once the optimal topology is evolved.

The optimum evolved topology for this problem when both sizing and topology optimisation are performed in tandem is shown in Figure 4.10.

For load case 2, again based on a fully braced 2-bay truss arrangement, the most optimal topology would involve the removal of elements 2, 5 and 10. However, the removal of elements 2 and 10 together would create a mechanism (a kinematically unstable structure), whereby element 6 would be able to rotate freely around node 2. In order to avoid this, an allowance has been made within the program to ensure that only dynamically stable configurations (i.e. no mechanisms) can be evolved; each node requires at least two connected elements. The resultant stable structure is shown in Figure 4.10.

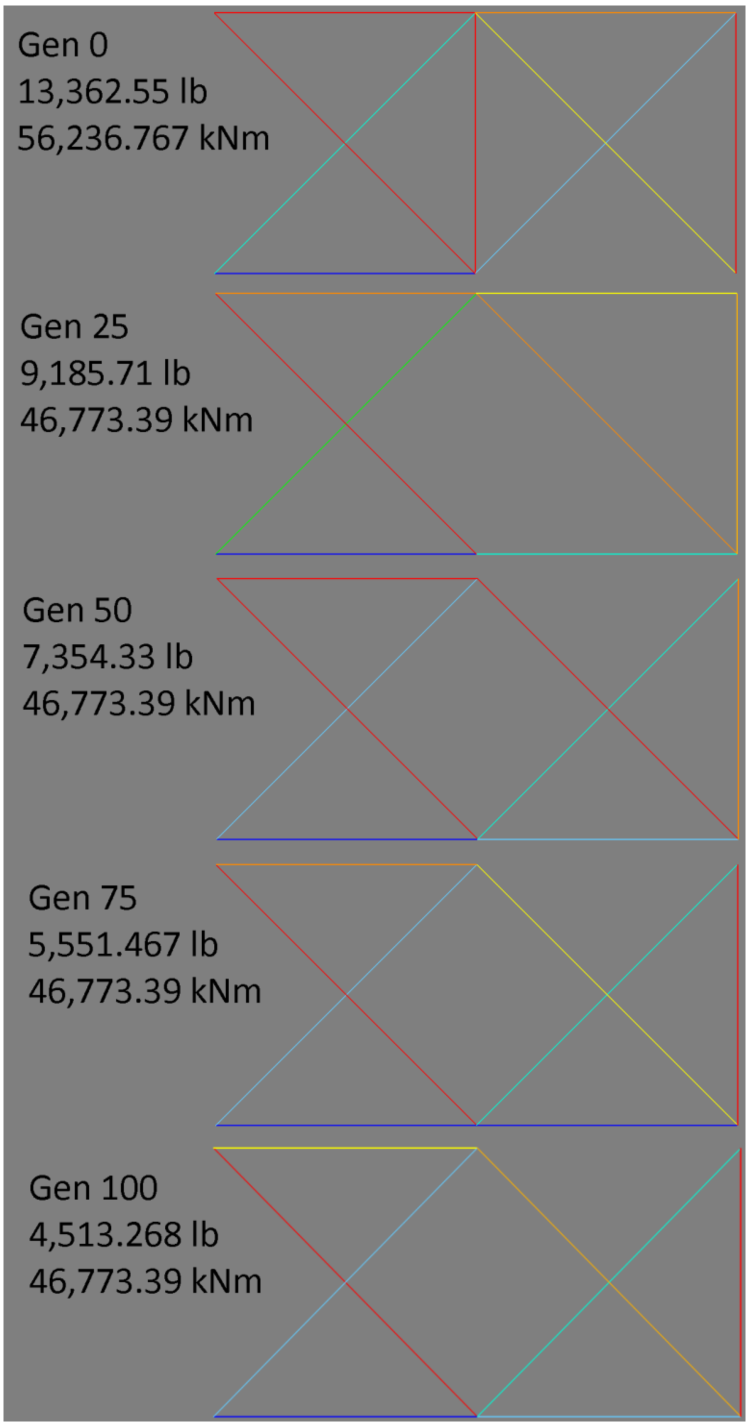


Figure 4.9: Progressive evolution of DO-GE solution for load case 2 of the 10-bar truss problem

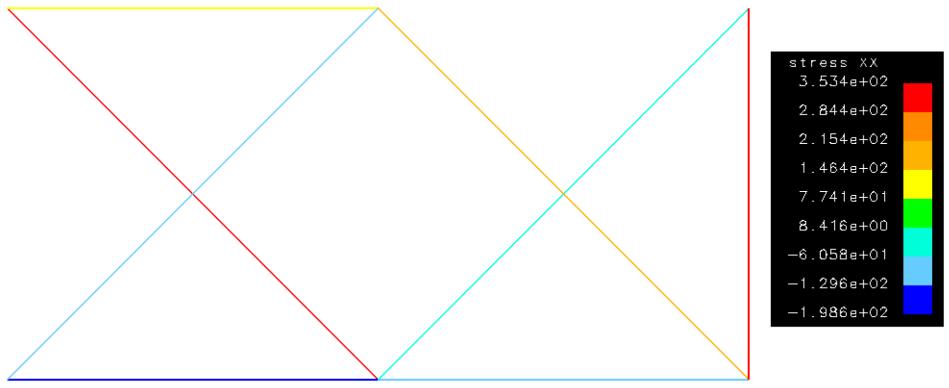


Figure 4.10: 10-bar truss problem, load case 2 - evolved minimum topology - stable structure

Interestingly, unlike load case 1 this topology does not match the global optimum for uniform materials. This suggests that there is an inherent link between the topological optimisation of a structure and the sizing optimisation of its structural members.

The evolved optimal member cross-sectional areas are shown in Table 4.5. The use of aluminium solid sections saw the best achieved solution from DO-GE surpassing that achieved by Kaveh and Talatahari [65], at 4517.8 lb. What is intriguing is that, contrary to previous examples, the use of steel sections in this instance saw an improvement in performance over aluminium sections at 4513.3 lb, 162.5 lb lighter than the best single optimisation solution from the literature.

Table 4.5: 10-bar truss, load case 2: Evolved minimum cross-sectional areas for DO-GE (in^2).

Element	**Li *et al.***	**Kaveh & Talatahari**	**DO-GE**	
			Aluminium solid sections	Using Tata CHS steel
1	23.353	23.194	27.0	7.238
2	0.100	0.100	None	None
3	25.502	24.585	19.4	8.680
4	14.250	14.221	11.8	5.828
5	0.100	0.100	None	None
6	1.972	1.969	2.1	0.993
7	12.363	12.489	10.0	4.138
8	12.894	12.925	16.4	5.828
9	20.356	20.952	19.6	4.975
10	0.101	0.101	0.1	0.237
Weight (lb)	4677.3	4675.8	4517.8	4513.3

A graph of the evolutionary runs for load case 2 comparing evolution of materials 1 and 2 using both GE and DO-GE is presented in Figure 4.11. As with the results from load case 1 (shown in Figure 4.7), dual optimisation in this specific application demonstrates both faster convergence and a lower standard deviation.

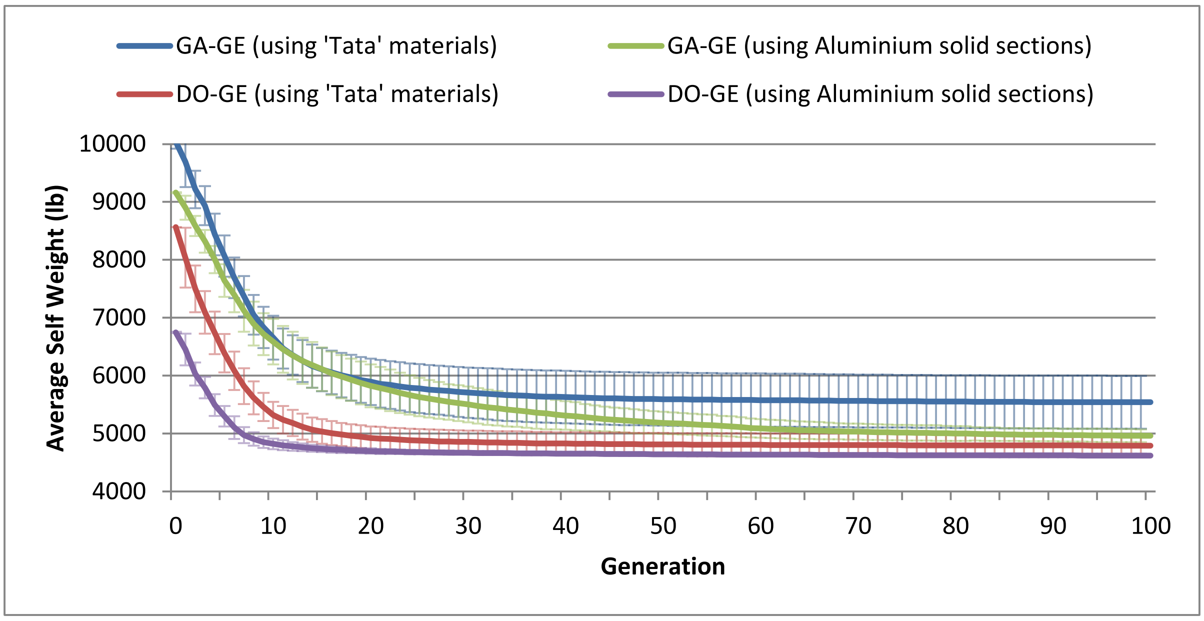


Figure 4.11: 10-bar truss optimisation - load case 2. Comparison between regular sizing optimisation (GA-GE) and dual topology and sizing optimisation (DO-GE) shows marked improvement when evolving both simultaneously.

4.3.3. 17-Bar Cantilevered Truss

Li *et al.* [64], Lee and Geem [76], Khot and Berke [128] and Adeli and Kumar [129] proposed solutions to the 17-bar planar truss problem shown in Figure 4.3. Nodes 1 and 2 were pinned, while a single vertical point load of 100 kips (444,800 N) was set at node 9. It should be noted that in this case, all of the prior research utilised steel as the design material. However, identical tensile and compressive stress limits were used. As with the 10-bar truss examples, two different sets of materials (one idealised set and one manufacturer/code compliant set) were used in order to benchmark the results against one another.

Topology Optimisation

The optimum topology for the 17 bar problem is shown in Figure 4.12.

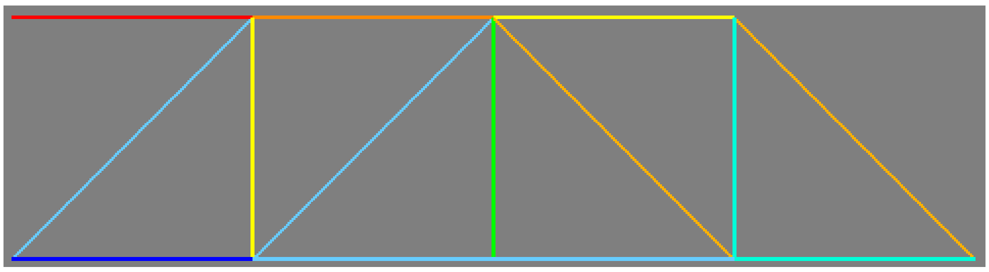


Figure 4.12: Least-weight topology of 17-bar truss

As with the 10-bar truss examples above, this topology represents the global optimum for this particular problem when uniform materials are used across the entire structure.

Sizing Optimisation

The best evolved solutions using GA-GE as a member sizing optimiser are shown in Table 4.6. These are compared with those found by previous works [64, 76, 128, 129], including individual member cross-sectional areas and overall structure weight. Using the Tata materials, the best achieved solution was 2774.5 lb. With the use of steel solid sections, the best achieved solution was 2605.7 lb, only 23.74 lb off the best solution achieved by Li *et al.* [64].

Table 4.6: 17-bar truss problem: Evolved minimum cross-sectional areas (in^2) using GA-GE

Element	**Khot & Berke** [128]	**Adeli & Kumar** [129]	**Li *et al.*** [64]	**GA-GE**	
				Steel solid sections	Tata CHS steel
1	15.930	16.029	15.896	16.0	12.276
2	0.100	0.107	0.103	0.1	2.387
3	12.070	12.183	12.092	12.2	15.392
4	0.100	0.110	0.100	0.1	0.394
5	8.067	8.417	8.063	8.1	8.944
6	5.562	5.715	5.591	5.7	2.403
7	11.933	11.331	11.915	12.1	9.486
8	0.100	0.105	0.100	0.1	0.578
9	7.945	7.301	7.965	8.0	5.828
10	0.100	0.115	0.100	0.1	2.527
11	4.055	4.046	4.076	4.0	7.301
12	0.100	0.101	0.100	0.1	1.736
13	5.657	5.611	5.670	5.6	5.208
14	4.000	4.046	3.998	4.4	4.588
15	5.558	5.152	5.548	5.7	2.651
16	0.100	0.107	0.103	0.1	3.317
17	5.579	5.286	5.537	5.7	3.658
Weight (lb)	2581.9	2594.4	2581.9	2605.7	2774.5

Dual Optimisation

The evolutionary progression for the 17-bar truss is shown in Figure 4.5. In a marked change from the 10-bar truss problem, the first generations of all runs of the 17-bar truss problem contained no fit solutions. As shown in Section 4.2, there are 2.8041679×10^{42} unique solutions to the DO-GE 17-bar truss problem. Thus, the probability of randomly finding a single fit solution in the first generation is too small in this instance.

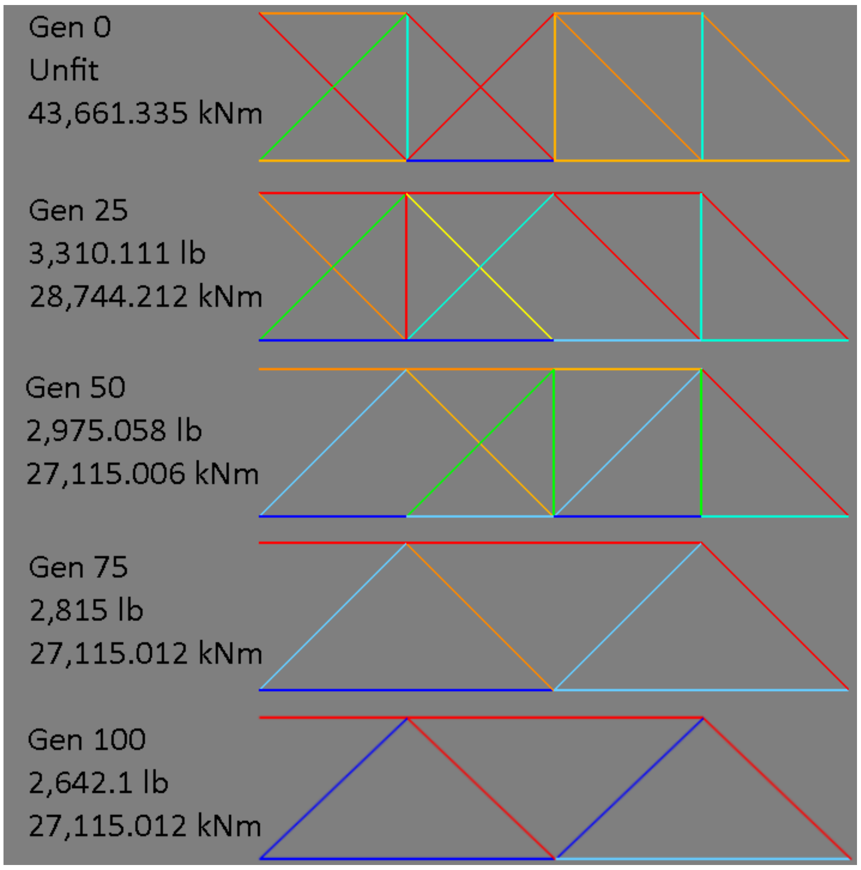


Figure 4.13: Progressive evolution of DO-GE solution for the 17-bar truss problem

The optimum evolved topology using DO-GE is shown in Figure 4.14 below.

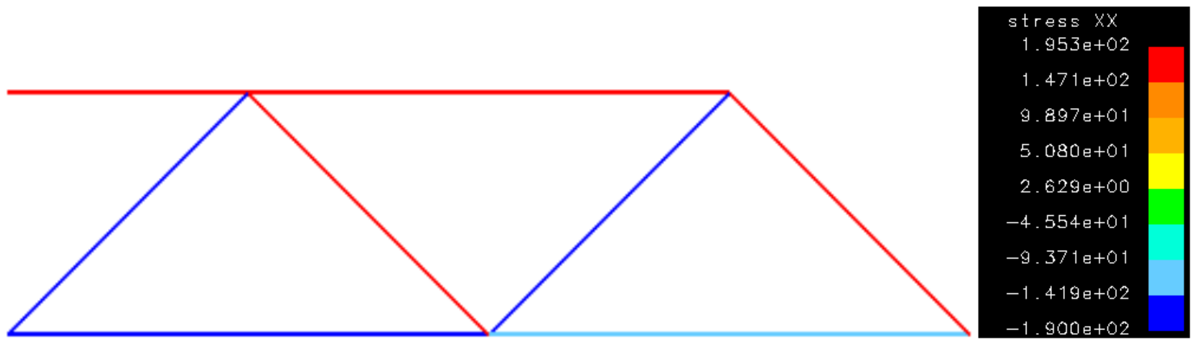


Figure 4.14: 17-bar truss problem - evolved minimum topology

As with load case 2 of the 10-bar truss problem, it can be seen that the topology evolved simultaneously with member sizing differs from the topology evolved with uniform materials (as shown in Figure 4.12). Once again this suggest that optimisation of structural topology and member sizing is inextricably linked and, thus, should be conducted in tandem.

Table 4.7 compares the performance of DO-GE against results from the literature. With the use of the steel solid sections material set, the best achieved solution was only 10.3 lb off that achieved by Li *et al.* [64], at 2595.4 lb. Using the Tata materials, the best achieved solution was 2642.1 lb.

Table 4.7: 17-bar truss problem: Evolved minimum cross-sectional areas (in^2) using DO-GE

Element	**Khot & Berke** [128]	**Adeli & Kumar** [129]	**Li *et al.*** [64]	**DO-GE**	
				Steel solid sections	Tata CHS steel
1	15.930	16.029	15.896	16.0	13.826
2	0.100	0.107	0.103	None	None
3	12.070	12.183	12.092	12.2	12.276
4	0.100	0.110	0.100	None	None
5	8.067	8.417	8.063	8.1	7.766
6	5.562	5.715	5.591	5.7	5.828
7	11.933	11.331	11.915	12.1	12.276
8	0.100	0.105	0.100	None	None
9	7.945	7.301	7.965	8.0	7.766
10	0.100	0.115	0.100	None	None
11	4.055	4.046	4.076	4.1	4.433
12	0.100	0.101	0.100	None	None
13	5.657	5.611	5.670	5.7	5.208
14	4.000	4.046	3.998	4.1	4.464
15	5.558	5.152	5.548	5.7	5.208
16	0.100	0.107	0.103	None	None
17	5.579	5.286	5.537	5.7	5.208
Weight (lb)	2581.9	2594.4	2581.9	2595.4	2642.1

A graph of the evolutionary runs comparing evolution of Tata materials and steel solid sections using both GE and DO-GE is presented in Figure 4.15.

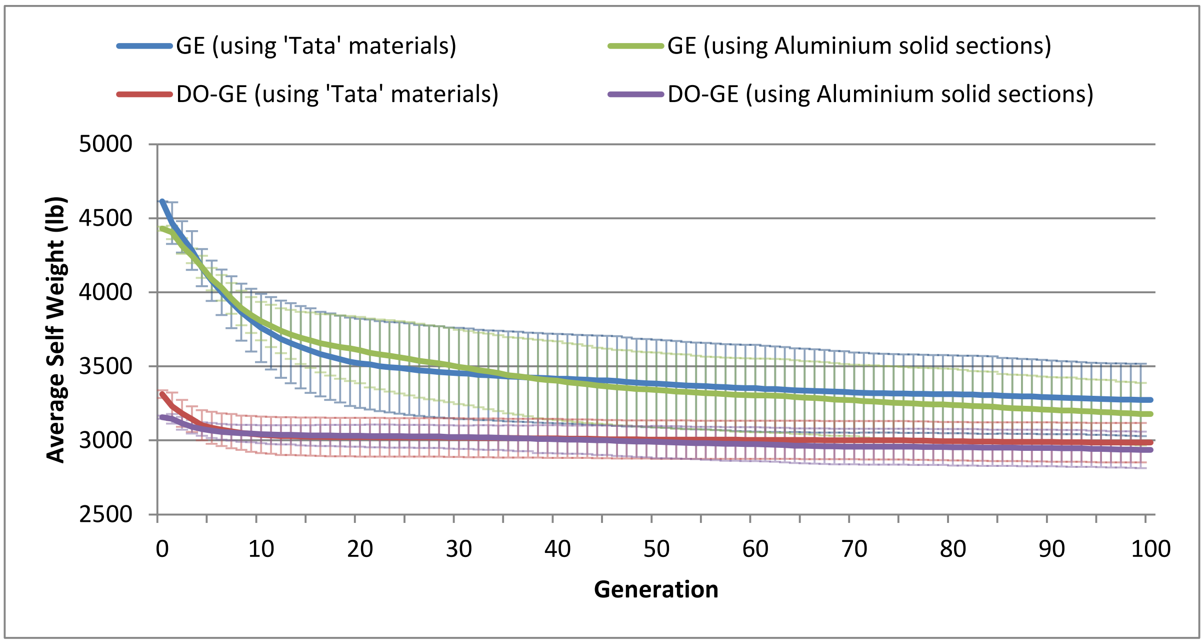


Figure 4.15: 17-bar truss optimisation. Rapid convergence and lesser standard deviation apparent with dual optimisation approach.

4.4. Deterministic Sizing Optimisation

Although the work presented in this thesis uses two Chromosomes to optimise sizing and topology simultaneously, sizing optimisation can also be seen as a deterministic problem when given an initial material configuration [7]. The optimum required member sizings for a particular topological layout can be determined by analysing the load paths and stresses in the loaded structure, as utilized by Shea and Smith [85]. The current stress in each edge can be multiplied by the original cross-sectional area of that edge to calculate the force in each edge (Eq. 4.8). The minimum required cross-sectional area of that edge such that it will pass all of its stress constraints can then be calculated as the force divided by the observed stress, as shown in Eq. 4.9.

$$Force = Observed\ Stress * Original\ Area \qquad \text{Eq. 4.8}$$

$$Minimum\ required\ cross\ sectional\ area = \frac{Force}{Observed\ Stress} \qquad \text{Eq. 4.9}$$

This minimum required cross-sectional area can then be compared with the section sizes of all available construction elements (i.e. different material sizes), and the most appropriate material can then be reassigned to that edge such that the stress in each edge is maximised with regards to its allowable limit:

$$m_i^{area} \le Required\ Area \le m_{i+1}^{area}$$

$$Required\ Area = m_{i+1}^{area}$$

$$\forall\ m_i \in Materials\ List$$

Once all members in an individual have been optimised, the structure must be re-analysed and the individual must again have its fitness evaluated, as a change in the member sizings induces a change in the stress state of each edge. Due to the highly optimised nature of the problem (with all constraints at or just below their limits), these changes, while potentially small, could push the structure over its constraint limits. This would mean that the "optimised" structure could remain unfit and would need to be re-optimised and once again have its fitness evaluated. Since this loop can continue indefinitely if not managed properly, a termination criteria needs to be set, either in the form of a maximum number of optimisation iterations (i.e. a limit to the optimisation depth), or some stopping criteria (i.e. an improvement over the original fitness has been found, or if convergence has occurred and no further improvement in the fitness is observed). A trade-off must, therefore, be reached between computational time and the efficiency of the deterministic optimisation algorithm. Notably, various experimental runs demonstrated that on average optimised fitness results stabilize and achieve convergence after the 5th optimisation step.

An experiment was performed to compare both GA-GE and the dual chromosome DO-GE method previously described in this chapter against this deterministic sizing optimisation method. The same benchmark problems described in Section 4.2 were attempted, except this time only one chromosome was used: Ch.A, which defined the structural topology. All member sizings were directly calculated using the deterministic optimisation method described above, and the results were compared with those of previous experiments which used two chromosomes to solve the same problems.

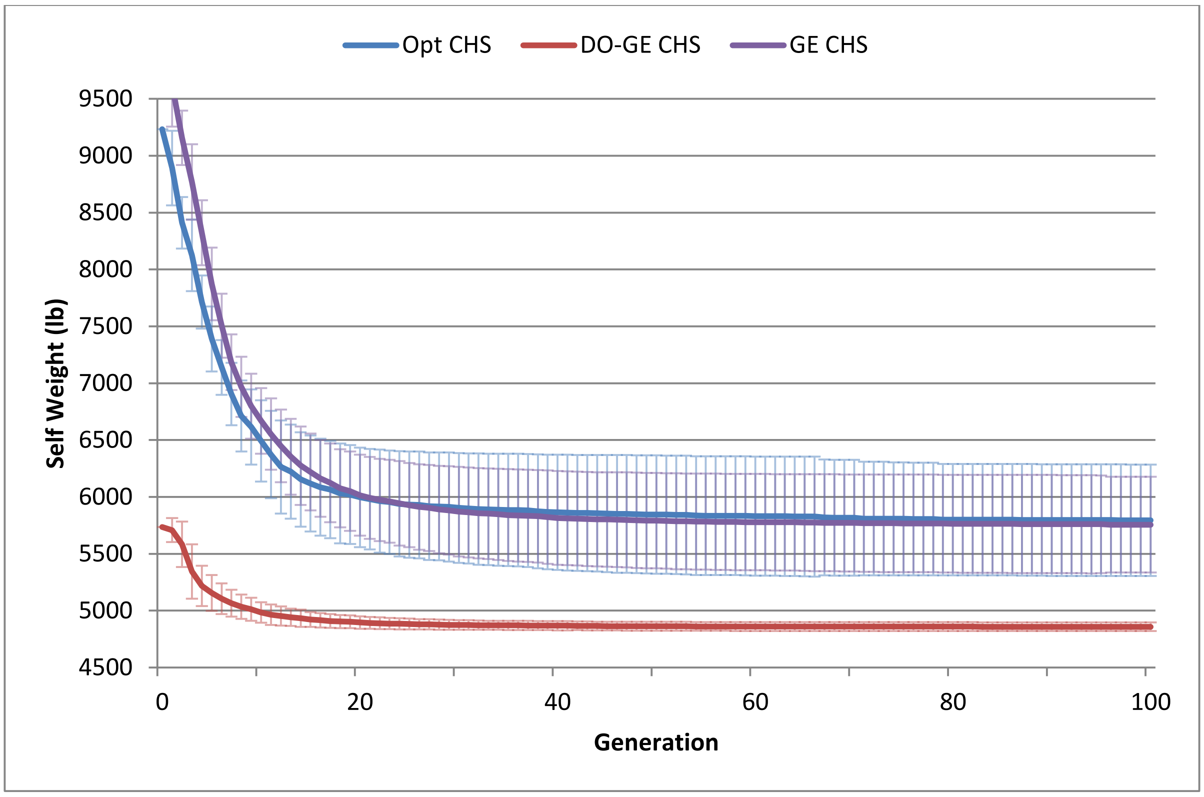


Figure 4.16: 10 Bar Truss, Load Case 1 (CHS materials). Comparison between regular GA-GE, DO-GE, and GE running a deterministic member size optimiser.

As shown in Figure 4.16, GA-GE performs identically whether using a deterministic optimiser or using an evolutionary process to optimise member sizings. However, while the overall optimised results for deterministic optimisation and GA-GE were nearly identical in both cases, optimising the member sizings at each step vastly increased the computational effort required, leading to a five-fold increase in the run time for a single evolutionary run with identical settings. Since deterministic optimisation produces identical results to GA-GE but with increased computational time, it offers no performance benefits for truss optimisation problems. Conversely, the full dual-chromosome optimisation method shows much faster convergence with a lower standard deviation and an overall better average fitness. Although only one load case is shown here, these results were seen across all three groups experiments performed, regardless of material choice or the loading scenario. This corroborates Luh and Lin's assertions [77] that the most optimal solutions for discrete methods can only be found by simultaneously considering optimisation of size, shape, and topology as each has an effect on the others.

4.5. Discussion

While the best solution for each load case found using DO-GE was consistently heavier than the best solutions from previous works, this is due to both the use of different materials (aluminium was used in the case of the 10-bar truss) and a more restricted range of available materials. Traditional optimisation methods [7, 73, 12, 13, 64, 65, 76] calculate the minimum required cross-sectional area, while the DO-GE method presented here matches the closest commercially available element (based on a predefined list) to that minimum.

When the results are analysed in more detail, some interesting points become apparent. These are directly related to the application of the method as a structural design tool and highlight some important short-comings of the more traditional optimisation methods.

For the literature examples presented in Section 4.2, compressive and tensile stress limits were identical and were set at 25 ksi in the case of the 10-bar truss and 50 ksi for the 17-bar truss. In structural design practice this is far from the case, as many section-dependent factors govern the material stress limits, including relevant design codes of practice [57, 58, 59, 60] and manufacturer’s specifications [118]. This discrepancy is particularly relevant in the case of axial compression. Both standard codes of practice [59, 60] and manufacturer specified compression resistance limits [118] are considerably more conservative as they take into account various factors of safety. These are not used in traditional optimisers. Also these variable stress limits apply regardless of the material used, as compressive stress limits are a function of the length of the member, its cross-sectional area, and its thickness.

In the manufacturer’s specifications [118], along with the current standard for structural steel design [60], methodologies are available to calculate the allowable maximum resistance of axial compression members. These are based on the gross cross-sectional area, the effective length of the member, and the given maximum compressive strength of the material (which itself is a function of the section geometry). DO-GE addresses these issues by building dictionaries of section data for each structural member, including section geometry and properties, compressive and tensile stress limits (based on section geometry, design codes of practice, and manufacturer specifications), and permissible Euler buckling limits. A summary of the maximum permissible stress for each element in compression is given in Table 4.8.

Table 4.8: Summary of compression elements in sample problems

Element	**Length (in)**	**Second moment of area (in^4)**	**Max allowable stress (ksi)**
		10-bar truss (load case 1)	
3	360	153.040	29.810
4	360	72.075	18.505
8	509.12	153.040	19.420
10	509.12	2.883	3.416
		10-bar truss (load case 2)	
2	360	0.235	0.758
3	360	153.040	29.810
4	360	101.146	22.552
8	509.12	72.075	18.505
		17-bar truss	
3	100	749.583	45.112
4	100	0.074	3.785
7	100	44.927	45.016
11	100	101.146	49.362
12	100	4.132	35.991
14	100	31.713	47.272
15	141.42	9.442	32.392
16	141.42	7.520	22.902
17	141.42	12.757	32.011

Inspection of the data shows that there is considerable variation in the maximum permissible compressive stresses, with values ranging from 0.75 to 49.4 ksi. When the approach used by traditional optimisation methods (to take a fixed constant value for this property) is considered, the implications of this approach with respect to compressive resistance and buckling are very clear. The results also highlight a very significant shortcoming of traditional optimisation methods.

For illustrative purposes, all experiments were run a second time with a wider array of materials representative of those of other works (solid aluminium and steel material sets, as described in Table 4.1. Evolved solutions were found to be much closer to the previous best solution reported in the literature, using the standard commercially available cross sections. Comparisons between element cross-sectional areas evolved using DO-GE and those of previous methods (Table 4.2 to Table 4.7) confirm that a dual optimisation approach for evolving both structural topology and member sizing simultaneously is fully capable of matching other optimisation methods.

4.6. Conclusions

Simultaneous optimisation of structural topology and member sizing was hypothesised to be capable of generating highly optimal viable solutions. Through the use of two unique chromosomes, it was shown that evolution of two separate datasets for a single objective was possible. This dual optimisation method was compared against popular methods from the literature and was found to be capable of producing similar or superior solutions. Comparison was also made against a deterministic sizing optimisation method, which was found to have a five-fold increase in the computation time required for evolution. This corroborates the reasoning for using a heuristic approach over a deterministic approach, as stated in Chapter 1; heuristic approaches trade the outright accuracy of deterministic approaches for computational speed.

This chapter also demonstrated that the majority of reviewed discrete optimisation literature use unrealistic assumptions about material properties and physical structure layout. Although numerous notable truss optimisation methods exist, it was found that the most popular methods are not fully appropriate for everyday use in the construction industry, as they focus purely on optimising minimum element cross-sectional area, neglecting crucial section properties and material specifications. Furthermore, the widespread use of identical tensile and compressive stress limits on the material, and the absence of design codes and standards of practice in such optimisation methods gives a false impression of both the efficiency of the algorithm and the importance of the achieved results. Experiments showed that realistic construction materials, when applied to the same problem, cannot produce the same results. The conclusion was drawn that the heavily optimised results of the literature give a false impression of the efficiency of the algorithms in question. There is a tendency in the literature to focus on minute improvements to existing techniques rather than introducing new paradigms. While the use of commercially available elements and constraints based on building code standards produces consistently heavier results than those of idealised optimisation methods, the approach described in this chapter only creates designs which conform to standard design codes of practice, and hence all fit individuals can be considered viable for construction.

Although fixed length chromosomes were used in this chapter, Chapter 6 will expand on this idea by exploring the recursive capabilities of GE in structure generation, allowing for variable-length chromosomes to be used. The addition of recursion to the grammar should theoretically vastly increase the representation capabilities of the program, allowing for an extensive increase in the number of potential derivable solutions. A potential problem arises however in that a larger representation space increases the importance of constraint handling in unfit individuals. Chapter 5 will, therefore, examine the implications of constraint handling in unfit individuals, including the impact that a larger representation space will have on evolutionary performance.

Chapter 5. Constraint Handling and Evolutionary Selection Pressure

Selection of appropriate techniques for handling different constraints is a key part of evolutionary optimisation and in all disciplines of heuristic optimisation algorithms. This particularly applies to the field of population-based evolutionary structural engineering truss optimisation where multiple conflicting constraints are often present. These constraints include standard engineering parameters such as stress, strain, deflection, buckling, and weight. They can, however, also include more complex constraints such as an accurate estimate of the cost of the structure or a subjective assessment of the architectural form. Some constraints may be hard limits which cannot be exceeded (e.g. stress limits or fire resistance), while other constraints may be more flexible (e.g. cost or aesthetic appearance). An individual solution can only be considered "fit" for purpose if it passes (i.e. does not violate) *all* constraints imposed upon it. The selection of appropriate functions for measuring these constraints, and the subsequent management of these parameters, is a crucial part of the evolutionary process.

Structural engineering optimisation will often require the designer to satisfy multiple parallel objectives, such as minimising the self-weight and cost of the structure, while maximising its structural rigidity [7]. There may also be overlaps between both constraints and objectives, such as minimising deflection, subject to a specified maximum deflection limit. In the initial phase of evolution where the population may be dominated by unfit solutions, understanding the interaction between these constraints and the overall metric by which the fitness of the individual is judged will have a significant impact on the quality of the designs produced. As such, a key challenge for designers when using evolutionary approaches is to find an accurate metric that will allow the designer to:

a) judge individual constraints, and
b) transform the performance of the individual (with respect to those constraints) into a single coherent value for use by the fitness function.

When an individual fails a constraint, its fitness must be amended such that it reflects the unfit nature of the individual, rendering it less likely to pass on genetic material to subsequent generations.

Traditional ground structure methods (such as those described in Chapter 4) consist of well-defined problems. With all potential solutions having to be specified *a priori,* a relatively small/limited number of potential solutions are capable of being generated. The probability of evolving a fit solution in the initial stages of the evolutionary process is high, as there are only a small total number of potential available solutions through which to search. In this instance, handling of constraint violations within the fitness function, while still important, is not a crucial issue, as fit solutions are easily obtained (and often appear in the first generation).

A possible issue arises, however, when the scope of the problem is increased. An increase in the total number of potential derivable solutions of the evolutionary algorithm can theoretically improve the quality (i.e. fitness) of the derivable solutions. However, this increase means the evolutionary process consequently has a much larger field through which to search (Figure 5.1). Whereas the handling of constraint violations within the fitness function may not be of particular importance in methods with limited representation capabilities (such as discrete "ground structure" topology optimisation methods discussed in Chapter 4 [10, 14, 69, 73, 77, 117]) due to the relatively high probability of evolving a feasible solution from a limited representation set, the increased difficulty inherent with an increased representation capabilities elevates the importance of managing violated constraints.

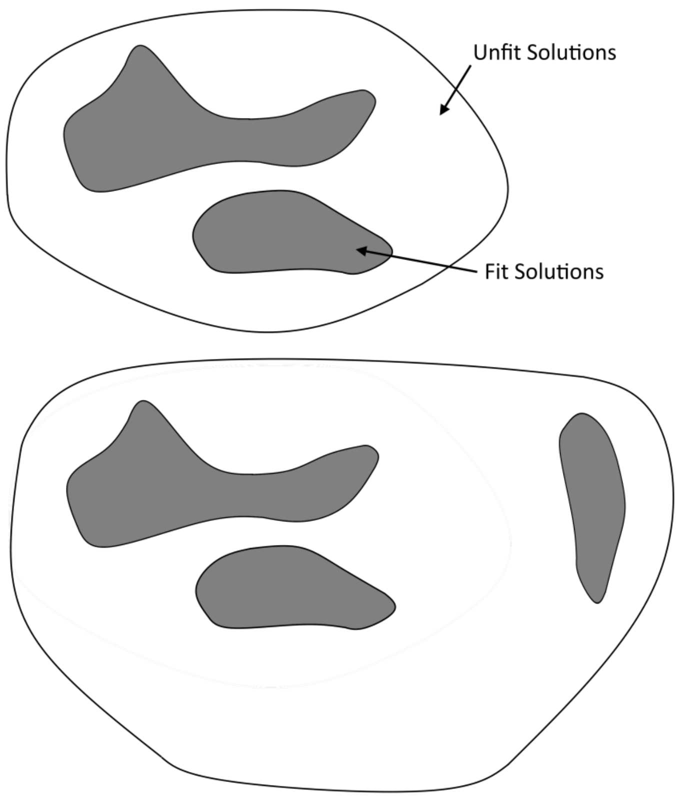


Figure 5.1: An increase in the representation space size may uncover more potential fit solutions, but the overall number of unfit solutions will see a greater increase.

This chapter does not set out to find the most appropriate method for evolving the fittest overall solution but rather to find the most effective way to evolve *any* fit solution where the probability of such is extremely low (i.e. the most efficient way to navigate to an area of the search space populated by a higher concentration of possible solutions). A number of questions are asked in order to ascertain this:

- Does the number of applied constraints have a positive or negative effect on the evolutionary process?
- Should constraints be seen as a necessary design component or a tool to improve search?
- What is the most efficient way to relate constraint violations to the overall fitness of the individual?

A method for applying accurate penalty values for individuals with multiple failed constraints is introduced, based on a quantifiable appraisal of the severity of any constraint violation. This method is compared against several methods from the literature using benchmark test cases of varying degrees of difficulty. The effect of differing constraint limits on the overall population evolution is analysed. The differences between varying degrees of hard and soft limits are discussed, as are the implications of their use in different scenarios.

The remainder of this chapter is structured as follows. Section 5.1 provides a background to the main themes discussed in this chapter. Section 5.2 details how changes to the definition of a problem can have greater or lesser impacts on the fitness landscape, thereby affecting the difficulty of the problem. This section includes a discussion on pre- and post-fit evolution in Section 5.2.2, and the development of a penalty function for unfit individuals in Section 5.2.3. This penalty function is compared against a number of benchmark numerical examples from the literature in Section 5.3, and the implications of these experiments are discussed in Section 5.4. The conclusions are drawn in Section 5.5.

5.1. Introduction

Terry Jones' work [4, 79] set in place a standardised model for understanding the terms associated with search space and landscapes. The terminology associated with certain components of heuristic algorithms in general must be understood correctly before the whole picture can be grasped.

1. The ***search space*** (**S_S**) contains the full set of all possible solutions. It is not limited in any sense by what the search algorithm is capable of producing and includes all conceivable (and non-conceivable) problem solutions regardless of their viability.
2. The ***representation space*** (**S_R**) is a subset of the search space, and encapsulates all potential design solutions which are capable of being generated and represented by the algorithm. The solutions in the representation space are derived from all permutations and combinations of genes in the chromosome, regardless of their actual merits as viable designs. Each point in the representation space represents a particular genotypic string which, when applied to a grammar, creates a phenotypical translation of that point. The larger the representation space, the greater the percentage of the search space that is covered and, therefore, the greater the search capabilities of the algorithm.
3. The ***fitness space*** (**S_F**) represents the S_R when mapped through some valuation of each particular solution. This is most commonly called the fitness function. Each point in the S_F is a quantifiable appraisal of a corresponding design solution from the S_R.

A landscape is a term often used as a visual metaphor for the relationships between the fitness values of a population of individuals. It is defined by Jones [4] as a combination of five essential components:

$$L = f(\mathrm{S_R}, \theta, F, S_F, >_F) \qquad \text{Eq. 5.1}$$

where,

S_R	= Representation space
θ	= Operator (crossover, mutation, etc.)
F	= Fitness Function
S_R	= Fitness Space
$>_F$	= A partial order over the Fitness Space

Jones's work mainly focussed on manipulation of θ and the observance of changes apparent in the landscape. The focus of this particular chapter is the manipulation of F and consequently the observable effects on $\mathrm{L_F}$, the fitness landscape. By extension, if all other variables are kept constant (R, θ, $>_F$), any variation in F will signify a corresponding variation in L_F. From this point forwards, the term ***fitness landscape*** can be thought of as equivalent to the $\mathrm{S_F}$, as all other variables are held constant.

5.2. Variation in Fitness Landscapes

The fitness landscape represents the performance aspects of each point within the $\mathrm{S_R}$. Individuals within the fitness landscape can be divided into two basic categories:

1. Fit individuals

   Individuals which pass all constraints or limiting factors imposed on them are deemed to be **fit** for their designed environment, in that they adequately perform all of the tasks required of them.

2. Unfit individuals

   Individuals which fail *at least* one constraint or limiting factor imposed on them are deemed to be **unfit** for their designed environment, in that they fail to adequately perform the tasks required of them.

While the $\mathrm{S_R}$ is concerned with the phenotypic appearance of an individual (i.e. physical dimensions and topological layout of a structure) and, as such, remains constant for any given design envelope/grammatical representation, variations in the fitness function can lead to a dynamic fitness landscape. The fitness landscape will often have terminology applied to it, which refers to peaks, troughs, valleys, and hills. These address the differing fitness values of points in the space. A peak would represent a local optimum; a point where all surrounding points have a worse fitness value. Conversely a trough will be surrounded by points with a better fitness value.

As a problem becomes more difficult (higher loading conditions, larger spans, thinner available members, etc.) the fitness landscape correspondingly changes. While there are still the same total number of potential solutions defined by the problem, the percentage of fit solutions in the representation space (hereafter called the ***fitness ratio***) reduces and it becomes increasingly difficult for the evolutionary algorithm to find a fit solution. A simple analogy would be to think of a fitness landscape as a series of islands in a sea, with the sea level representing the boundary above which all individuals are feasible. If the sea level rises, the islands correspondingly shrink in size (i.e. the total number of feasible solutions diminishes), and it becomes more difficult to find feasible solutions. In this case, the process is tending towards a random search, where the benefits of the evolutionary approach are lost. This correlation is shown experimentally in Section 5.3 where the S_R is kept constant, but the loading conditions are changed, leading to a decrease in the fitness ratio. To minimize evolutionary difficulties inherent with this effect, careful manipulation of constraints is essential. This concept is important to grasp, and the correlation can be demonstrated by a simple experiment.

5.2.1. Experiment: Increasing Problem Difficulty and its Effect on the Fitness Space

Consider a theoretical problem where an evolutionary algorithm has to find a suitable solution for the optimum size of a simply supported beam with a single point load at centre span. The only constraint relates to the bending moment capacity of the beam. The problem description consists of a solid rectangular steel beam (density 7,850 kg/m^3) of variable cross-sectional area, of fixed length L = 10m, supporting a variable vertical point load of P (N) at its centre. There are 10 available beams to choose from, as shown in Table 5.1:

Table 5.1: Material properties

Beam id	Breadth (mm)	Depth (mm)	Self-Weight (kg)
1	20	40	62.8
2	30	60	141.3
3	40	80	251.2
4	50	100	392.5
5	60	120	565.2
6	70	140	769.3
7	80	160	1,004.8
8	90	180	1,271.7
9	100	200	1,570
10	110	220	1,899.7

There are 8 test cases in consideration, each with a higher load than the previous:

Table 5.2: Test cases with variations in loading

Test Case	1	2	3	4	5	6	7	8
Load = P (N)	100	2,000	5,000	10,000	15,000	25,000	35,000	50,000

Traditional beam theory [57, 60] states that the maximum bending moment for a simply supported beam with a point load is:

$$M = \frac{PL}{4} \qquad \text{Eq. 5.2}$$

The elastic section modulus S for a rectangle is given as:

$$S = \frac{bh^2}{6} \qquad \text{Eq. 5.3}$$

The moment capacity of a beam is given in the standards [60] as:

$$M_c = p_y S \qquad \text{Eq. 5.4}$$

Taking grade S275 steel, with a design strength p_y of 275 N/mm^2, it is then possible to calculate the moment capacity M_c for any given section. Now, consider a representation space

R comprising of the selection of 10 solid rectangular beams of increasing size shown in Table 5.1. The bending moment applied to each beam for each given load can then be calculated. The fitness function in this instance merely subtracts the actual bending moment from the moment capacity, as shown in Eq. 5.5. If the number is positive, then there is a surplus moment capacity, and the beam is fit for purpose. If, however, the number is negative, then the beam does not have the requisite moment capacity to carry the load, and it is deemed to be unfit for purpose.

$$Surplus\ Capacity = (p_y \frac{bh^2}{6}) - \frac{PL}{4} \qquad \text{Eq. 5.5}$$

Table 5.3 shows that as the load increases (i.e. the fitness function *f* changes), the fitness landscape L_F changes and the fitness ratio (i.e. the percentage of fit solutions in the representation space) reduces, while the actual representation space (the list of available beams) remains constant. As the fitness ratio decreases, the likelihood of finding a fit individual correspondingly decreases.

Table 5.3: Surplus moment capacity (kNm) for increasing loads. Fit individuals are green, unfit are red

Section Properties		**Surplus Moment Capacity (kNm) for test cases**							
Beam Id	Moment Capacity (kNm)	1	2	3	4	5	6	7	8
1	1.47	1.22	-3.53	-11.03	-23.53	-36.03	-61.03	-86.03	-123.53
2	4.95	4.7	-0.05	-7.55	-20.05	-32.55	-57.55	-82.55	-120.05
3	11.73	11.48	6.73	-0.77	-13.27	-25.77	-50.77	-75.77	-113.27
4	22.92	22.67	17.92	10.42	-2.08	-14.58	-39.58	-64.58	-102.08
5	39.6	39.35	34.6	27.1	14.60	2.1	-22.9	-47.9	-85.4
6	62.88	62.63	57.88	50.38	37.88	25.38	0.38	-24.67	-62.12
7	93.87	93.62	88.87	81.37	68.87	56.37	31.37	6.37	-31.13
8	133.65	133.4	128.65	121.15	108.65	96.15	71.15	46.15	8.65
9	183.33	183.08	178.33	170.83	158.33	145.83	120.83	95.83	58.33
10	244.02	243.77	239.02	231.52	219.02	206.52	181.52	156.52	119.02
Fitness ratio		**100**	**80**	**70**	**60**	**60**	**50**	**40**	**30**

5.2.2. Constraints: Pre- and Post-Fit Evolution

The evolutionary process can be divided into two stages: pre-fit evolution and post-fit evolution. Pre-fit evolution refers to the evolutionary process *before* any fit individuals have been found by the algorithm, while post-fit evolution begins as soon as a single fit individual appears within the population. In the pre-fit evolutionary phase, the interaction between objectives and constraints is of vital importance in navigating through the initial unfit-laden search space to find fit individuals. This is achieved by managing the relationship between the constraints and the objectives of *unfit* individuals within the fitness function.

Constraints provide high-level information about the performance of all elements and aspects of an individual, while the fitness represents the overall performance of the individual relative to both its peers and the overall evolutionary objective. The addition of more constraints does not necessarily improve search characteristics. Indeed the opposite could be argued. More constraints theoretically mean a lower quantity of fit individuals in the overall population, with a “worse” overall average fitness for those individuals with more constraints over those with fewer constraints. Supposing a second constraint is added to the original example from Table 5.3: a vertical deflection limit. The formula for calculating vertical deflection δ is given [57, 60] as:

$$\delta = \frac{PL^3}{48EI}$$
Eq. 5.6

Limits on deflection for a steel beam are given in the standards [60] as:

$$\delta_{max} = \frac{Span}{200}$$
Eq. 5.7

With a span of 10m, the deflection constraint becomes:

$$\delta_{max} = \frac{10{,}000mm}{200} = 50mm$$
Eq. 5.8

With a modulus of elasticity E of 205,000 N/mm^2 for grade S275 steel, we can calculate the maximum observed deflection as:

$$\delta = \frac{P\ * (10{,}000)^3}{48 * 205000 * \frac{bd^3}{12}}$$
Eq. 5.9

$$\delta = 101{,}626 * \frac{P}{\frac{bd^3}{12}}$$
Eq. 5.10

Using these formulae, the fitness function can now calculate the actual deflection of each beam for each given load and compare it to the allowable deflection limit of 50mm, as shown in Table 5.4.

Table 5.4: Addition of deflection constraint reduces number of fit individuals. Fit individuals are green, previously fit but now unfit individuals are orange, and originally unfit individuals are red.

Section Properties		**Actual Deflection (mm) for load cases**							
Beam Id	I (mm^4)	1	2	3	4	5	6	7	8
1	106667	95.27	1905.49	4763.72	9527.44	14291.16	23818.6	33346.04	47637.2
2	540000	18.82	376.39	940.98	1881.96	2822.95	4704.91	6586.87	9409.82
3	1706667	5.95	119.09	297.73	595.46	893.2	1488.66	2084.13	2977.33
4	4166667	2.44	48.78	121.95	243.9	365.85	609.76	853.66	1219.51
5	8640000	1.18	23.52	58.81	117.62	176.43	294.06	411.68	588.11
6	16006667	0.63	12.7	31.74	63.49	95.23	158.72	222.21	317.45
7	27306667	0.37	7.44	18.61	37.22	55.82	93.04	130.26	186.08
8	43740000	0.23	4.65	11.62	23.23	34.85	58.09	81.32	116.17
9	66666667	0.15	3.05	7.62	15.24	22.87	38.11	53.35	76.22
10	97606667	0.1	2.08	5.21	10.41	15.62	26.03	36.44	52.06
Fitness ratio		**90**	**70**	**50**	**40**	**30**	**20**	**10**	**0**

Comparisons between Table 5.3 and Table 5.4 show that the fitness ratio has reduced in all cases with the addition of a second constraint, with fewer overall fit individuals for every load case. Sections highlighted in orange represent members that were fit when only the single constraint of moment capacity was being considered but which are deemed unfit when both moment capacity and deflection are considered.

In engineering optimisation the addition of constraints is more often than not a design necessity rather than an evolutionary choice. The existence of more constraints thus represents better design practice. In this regard, constraints can (and should) be used similarly to design codes of practice [59, 60, 117].

The number of constraints C and the number of fitnesses F will remain constant between individuals. Each element E in an individual I must pass $c \in C$ constraints depending on the state of that element. For example, a member in compression might have constraints such as buckling, web bearing, compression limits, etc., while elements in tension will have different constraints applied; differing conditions will incur variable constraints per element E. Constraint application does not necessarily need to be limited to individual members within an individual I and can be applied broadly to the entire individual itself (such as an overall cost or weight limit) or to portions thereof (such as groups of members forming joints). If E ($\forall E \in I$) passes all constraints $c \in C$, then the fitness f of the individual is left unchanged. If, however, any constraint c is violated (or indeed if multiple constraint violations occur), then that individual must be assigned a penalised fitness F_p such that it becomes correspondingly less likely for that individual to succeed to the next generation. In this instance the interaction between C and F_p becomes critical.

5.2.3. Constraint Handling in Unfit Individuals: Development of a Reliable Penalty Function

There are multiple ways of handling unfit individuals, but the most common is to add a penalty value to the original fitness of the individual [111]. Once an individual fails a constraint, it is deemed unfit for purpose and is assigned a penalised fitness. Constraint handling reviews from the literature (e.g. [109, 80]) recommend that the penalty to be applied to the fitness be kept as low as possible. It should be right above the feasible/infeasible boundary, or the Near Feasibility Threshold (NFT) (similar to the description of the boundary by Li *et al.* [64]); this method is often called the *minimum penalty rule*. However, in the case of structural engineering optimisation this might not be possible as there may be no clearly defined boundary between fit and unfit individuals when multiple constraints are present (indeed, Coello Coello acknowledges that this is apparent in most applications [109]).

To continue with the beam optimisation example presented in the previous section, the constraints on the problem consist of the bending moment capacity and the vertical deflection of the beam. The fitness of each beam is determined as the self-weight of the beam, and the problem definition is concerned with minimising this self-weight for any particular load case. For load case 1 in Table 5.4, beams 2 and 10 are both fit for purpose, with beam 2 the “fitter” of the two options having a self-weight of 141.3 kg compared to 1,899.7 kg for beam 10.

Suppose, theoretically, that the load for test case 1 suddenly increases from 100N to 300N, as shown in Table 5.5.

Table 5.5: Actual deflection for beams 2 and 10, load case 1

Section Properties		**Actual Deflection (mm) for load**	
Beam Id	Self-Weight (kg)	100 N	300 N
2	141.3	18.82	56.46
10	1,899.70	0.1	0.31

The actual deflection of beam 10 remains below the limit, at 0.31mm. However, the deflection of beam 2 is now 56.46mm, and it is no longer fit for purpose, as it fails the deflection constraint. Beam 2 must now be penalised such that its fitness cannot remain "better" than that of beam 10 (which already has a fitness over 10 times worse than that of beam 2 even though beam 10 is actually fit). There must be assurance that the fitness of unfit individuals cannot be better than the fitness of potential fit individuals, i.e. there must be a distinction made between fit and unfit individuals.

Coello Coello [109] asserts that:

"Penalties which are functions of the distance from feasibility are better performers than those which are only functions of the number of violated constraints."

Since the NFT cannot be clearly defined, the amount by which a constraint is violated must instead by calculated in order to gain an accurate estimate of the *distance to feasibility* (DTF) for that particular individual (i.e. some metric which indicates the number/severity of constraint failures). An accurate estimate of the DTF for each individual is theorised to improve the evolutionary selection pressure for fitter individuals, thereby improving the overall evolutionary process. What is developed here is a method of developing an accurate and quantifiable description of the severity of any constraint violation for any individual. This method compiles various recommendations from methods referenced from the literature and combines them to create an engineering-appropriate penalty function. This method is then compared against various methods from the literature in Section 5.3.

Normalising Constraint Violations

Since multiple different constraints may operate on different scales (i.e. deflection may be of the order of a few millimetres, while stress may be off by a few hundred Newtons per square millimetre), direct comparison of different constraint violations would be non-sensical. A much more logical method of comparing differing constraints can be achieved by normalising each constraint by its own limit. This can be obtained by assuming that the limit of a constraint is 100%, and then everything over that limit represents the percentage by which that constraint has failed (as shown in Eq. 5.11). This penalty mechanism provides a level metric by which to judge all constraint failures across all individuals in a population.

$$Failure = abs\left(\frac{(value - limit)}{limit} \times \frac{100}{1}\right) \qquad \text{Eq. 5.11}$$

In the presented example, the deflection limit is 50mm and beam 2 now deflects by 56.46 mm. The failure of beam 2 can now be quantified as:

$$Delection\ Failure = abs\left(\frac{56.46 - 50}{50} \times \frac{100}{1}\right)$$
$$= 12.92\% \qquad \text{Eq. 5.12}$$

Thus, beam 2 exceeds its deflection constraint by 12.92% of its capacity.

Using this technique, the amount by which any constraint has failed for any individual and for any test case can be rapidly assessed, which provides a level metric by which all failed constraints can be compared.

Multiple Constraint Failures

While normalising constraints can allow for unbiased comparison between separate individuals with single constraint failures, a new issue arises with individuals with multiple constraint failures. In test case 4 (Table 5.4), both beams 4 and 5 have failed. However, beam 4 fails in both bending and deflection, whereas beam 5 only fails in deflection. The failure percentages for beam 4 are:

$$Bending\ Failure = abs\left(\frac{22{,}916{,}666.67 - 25{,}000{,}000}{25{,}000{,}000} \times \frac{100}{1}\right) \qquad \text{Eq. 5.13}$$

$$= 8.33\%$$

$$Deflection\ Failure = abs\left(\frac{243.90 - 50}{50} \times \frac{100}{1}\right) \qquad \text{Eq. 5.14}$$

$$= 378.8\%$$

while the percentage failure for beam 5 is:

$$Deflection\,Failure = abs\left(\frac{117.62-50}{50}\times\frac{100}{1}\right)$$
$$= 135.25\% \qquad \text{Eq. 5.15}$$

These failures must be processed in such a way that they not only accurately describe the performance of the individual for the task at hand, but also such that they sufficiently differentiate failed individuals. In addition to the most popular constraint handling method of static penalties, three separate methods of penalty function generation by processing constraint violation information from the literature can be further developed using the normalisation method introduced in the previous section in order to relate them to the overall fitness of the individual [109, 80, 130, 111]:

1. Number of Failed Constraints

   The first method merely counts the number of violated constraints and assigns a penalty value to that individual proportional to the number of failed constraints, but not to the severity of those violations.

2. Maximum Failure

   The second method takes the single constraint that produces the greatest percentage violation of its limitation. This means that an individual that might violate multiple constraints will only be assessed on the basis of the worst violated constraint [62]. This method can be readily modified to use the normalised percentage-based method described above.

3. Summation of All Failures

   The third method sums all constraint violations by which each element E of an individual I has failed any constraint c. This means that individuals with multiple constraint violations will have a correspondingly poor fitness, as they cumulatively have a higher DTF. This method can be readily modified to use the normalised percentage-based method described above.

Penalty Function

Once the violation amount has been quantified, the final step is to transfer that information to the fitness via a penalty function of some form. While both Coello Coello [110] and

Michalewicz [80] advocate the addition of a constraint violation penalty to the fitness, this might not sufficiently differentiate individuals as the violation can be very small and the range of possible fit solutions may be very large (i.e. a fit solution for a truss may exist at 1,000 kg, while a different fit solution may also exist at 20,000 kg). The end result may be that an unfit individual, even when penalised, could still theoretically end up with a “better” fitness than a truly fit individual since there is no clear boundary between fit and unfit individuals. For this reason all constraint violations and associated penalties in this thesis are multiplied by the fitness, as advocated by Kawamura et al. [73]. This multiplicative constraint violation is correspondingly hereafter known as the *“violation multiplier”*.

Due to the wide range of possible fitness values for *fit* individuals, an additional multiplication buffer is advised [62] in order to ensure that individuals are sufficiently differentiated from the total range of plausible fit individuals. The overall fitness of unfit individuals is therefore multiplied by both a constant (set at 100 in this thesis) and the violation multiplier described above, as shown in Eq. 5.16. If no constraints are broken then the fitness of the individual is unaffected. Most importantly, this multiplication method acknowledges Coello Coello’s findings that the penalty be a function of the DTF.

$$f_p = (100 * f * \text{violation multiplier}) \tag*{Eq. 5.16}$$

The evolutionary process must also be prevented from “cheating”. The violation multiplier has been defined above as a percentage of the normalised constraint limits, which is then multiplied by the original fitness. Thus, the evolutionary process can produce an individual that violates the constraints by only a very small fraction of a percentage. This fraction would then be multiplied by the original fitness value to generate a total penalised fitness, which would be deemed “better” than the original fitness by virtue of being a lower value, even taking the multiplication buffer of 100 into account.

For example, consider a bridge with a weight of 5,000 kg and a deflection of 100.01mm. If the fitness of this bridge is its self-weight and given the sole constraint of a deflection limit of 100mm, this bridge would violate its constraint by 0.01mm, corresponding to an overall violation of 0.01%. The penalised fitness of this bridge would then be:

$$f_p = (100 * 5000kg * 0.01\%) = 5000 \tag*{Eq. 5.17}$$

From Eq. 5.17 it can be seen that if the overall fitness of the bridge is multiplied by (100 * 0.01), contrary to penalising it there would not be any net effect on the fitness of the individual. This could then create an individual with a seemingly better fitness than other truly fit individuals, due to the multiplication of the overall fitness by a fraction of a percentage.

Because of this potential anomaly, this thesis recommends that all normalised constraint violations must have a (+ 1) value added to the given violation multiplier. This extra buffer ensures that only positive, non-fractional violation multipliers can be used for penalty factors.

Combining all the various recommendations thus far, the two main DTF methods of handling constraint violations mentioned earlier (Maximum Failure and Summation of All Failures) can be heavily modified to incorporate these findings. The resultant penalty functions are novel and can be considered well suited to discrete evolutionary structural engineering as they are designed to work well with multiple conflicting objective constraints of varying magnitude. The final proposed maximum violation penalty value to be used for unfit individuals is:

$$f_{p\,max} = 100f * \left(\left(\max \left(abs \frac{\left(c_o^i - c_l^i\right)}{c_l^i} \times \frac{100}{1} \right) \right) + 1 \right) \qquad \text{Eq. 5.18}$$

$$\forall\, E \in I$$

while the final proposed summation violation penalty value to be used for unfit individuals is:

$$f_{p\,sum} = 100f * \left(\left(\sum_{\mathrm{i}=1}^{\mathrm{C}} \left(abs \frac{\left(c_o^i - c_l^i\right)}{c_l^i} \times \frac{100}{1} \right) \right) + 1 \right) \qquad \text{Eq. 5.19}$$

$$\forall\, E \in I$$

where:

c_o^i	= the observed value of constraint i
c_l^i	= the limiting value of constraint i

The bridge failure example above would now have a penalised fitness of:

$$Penalised\ Fitness = 100 * 5000kg * (0.01 + 1) = 500{,}050$$

Eq. 5.20

This additional 1% buffer now separates all individuals according to their constraint violations, on a sliding scale depending on the severity of the violation. All unfit individuals are sufficiently differentiated from fit individuals by the multiplication buffer of 100.

5.3. Constraint Handling Experiments

In order to observe the impact of the differing methods of handling broken constraints, a set of experiments was conducted. The procedure for these experiments was to steadily increase the load on varying structures, thereby reducing the fitness ratio and increasing the difficulty of the problem. In the following experiments, three of the most popular methods of handling constraints (as described on page 124) from the literature were compared against the Summation Violation technique presented here:

1. Summation violation (Sum Fail, as detailed in Eq. 5.19)
2. Maximum violation (Max Fail, as detailed in Eq. 5.18)
3. Number of failed constraints (Num Fail)
4. Static penalty (Default Fit)

Three benchmark test cases from the literature were investigated, hereafter named Test Case 1 (TC1), Test Case 2 (TC2), and Test Case 3 (TC3). In order to demonstrate the variation in effectiveness of differing constraint handling methods on problems of varying difficulty, different loading cases were examined for each test case. Each test case contained three load cases - light, medium, and heavy loading (hereafter named Load Case 1 (LC1), Load Case 2 (LC2), and Load Case 3 (LC3)), thereby representing successive increases in the problem difficulty. Loads far in excess of those specified in the literature were selected such that the evolutionary process would have great difficulty in evolving any fit solutions.

The objective of these experiments was to ascertain which of the given constraint handling methods is the most suitable for use with high-difficulty problems (i.e. problems with a very low fitness ratio).

Fitness ratios were calculated for each problem by performing a partial enumeration of the search space. In each case, 1,000,000 randomly generated solutions were evaluated (in line with methods described by Michalewicz [130]) and the percentage of fit individuals in the representation space was calculated. This fitness ratio gives a direct numerical index for the difficulty of a problem [130]. A larger number indicates a larger proportion of fit individuals in the representation space, which means that the problem is easier and could potentially be solved by random search. A high fitness ratio also means that there is a high probability that fit solutions will be generated in the randomly initialised first generation, meaning there is less of a challenge for evolution.

The hypothesis for these experiments was that for any load case of any test case, there should be a notable difference between the effectiveness of varying methods of handling broken constraints. At the highest loading levels (representing the highest difficulty levels and the lowest fitness ratios), it was expected that fit individuals would no longer appear in the randomly initialised first population for any constraint handling method. At the lowest loading level it was expected that most methods would have little difficulty in evolving fit solutions. In each test case a popular benchmark structural engineering design envelope was generated to represent standard evolutionary design problems. All four constraint handling methods were tested against all three loading conditions for each of the three test cases, giving 36 sets of experimental runs in total. The aim of these experiments was not to evolve the best individual for any particular test case, but rather to find the quickest way to evolve a single fit solution in an evolutionary run. This allows for comparison of constraint handling methods and assessment of the most appropriate method to use in cases where:

- the representation space is very large, and
- the fitness ratio is very low.

Since standard truss optimisation representation spaces are quite small (with a limited number of potential available solutions based on permutation and recombination of pre-defined truss layouts, as detailed in previous chapters), a new method of evolving truss structures by allowing variation in node placement rather than variation in node connections was developed. This new method allows variation in the placement of nodes rather than variation in node connections. In this fashion, all that is required in the definition of the problem is the boundary description, or design envelope (i.e. all necessary external forces and reactions, but no information about the internal layout of the structure). This new method vastly increases the representation capabilities of the search algorithm. A detailed discussion of this method is given in Chapter 6.

The measure of success for these experiments was determined as whether or not a single fit individual was evolved at any stage in the evolutionary run. This metric is hereafter called *the probability of evolutionary success*, with the reasoning that if a single fit individual has been evolved in a population, the algorithm has succeeded in finding a suitable solution for the problem (i.e. a solution which passes all imposed constraints). The probability of evolutionary success for a single loading condition on a particular test case will reach 100% if all algorithm runs for a particular constraint handling method evolve fit individuals. Evolutionary runs for each constraint method were performed 100 times per load case per test case (i.e. 100 runs for "TC1, LC1, Max Fail", 100 runs for "TC3, LC2, Default Fit", etc.). The test cases and load cases employed are described in subsequent sections. Experimental variables were set at:

- Population size: 1000
- Generations: 100
- Mutation Probability: 0.01
- Crossover Probability: 0.75
- Tournament Selection
- Tournament Size: 1% of Population Size
- Elite size: 1% of Population Size
- Generational Replacement

5.3.1. Test Case 1: Simply Supported Truss

The design envelope for TC1 (as shown in Figure 5.2) is taken from a common optimisation problem given by Li and Chen [51] and Wei *et al.* [89]. It describes a generic simply supported truss with a span of 18,288mm (720 inches), a depth of 9,144mm (360 inches), a pinned support at one end, and a rolling bearing at the other. A single point load F1 is applied vertically at the midpoint of the truss. Three load cases were tested:

1. 10MN (fitness ratio of 7.7003)
2. 15MN (fitness ratio of 0.4664)
3. 20MN (fitness ratio of 0.0364).

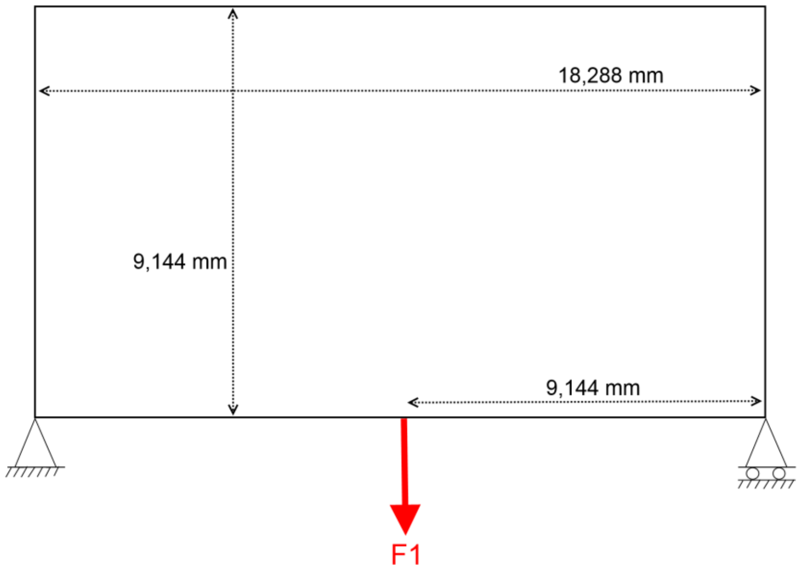


Figure 5.2: Basic simply supported truss example for constraint comparison.

Figure 5.3 shows that for LC1 (10MN) all three DTF methods (Sum Fail, Max Fail, and Num Fail) effectively produce the same results, with all 100 runs evolving fit individuals, indicating a 100% probability of evolutionary success.

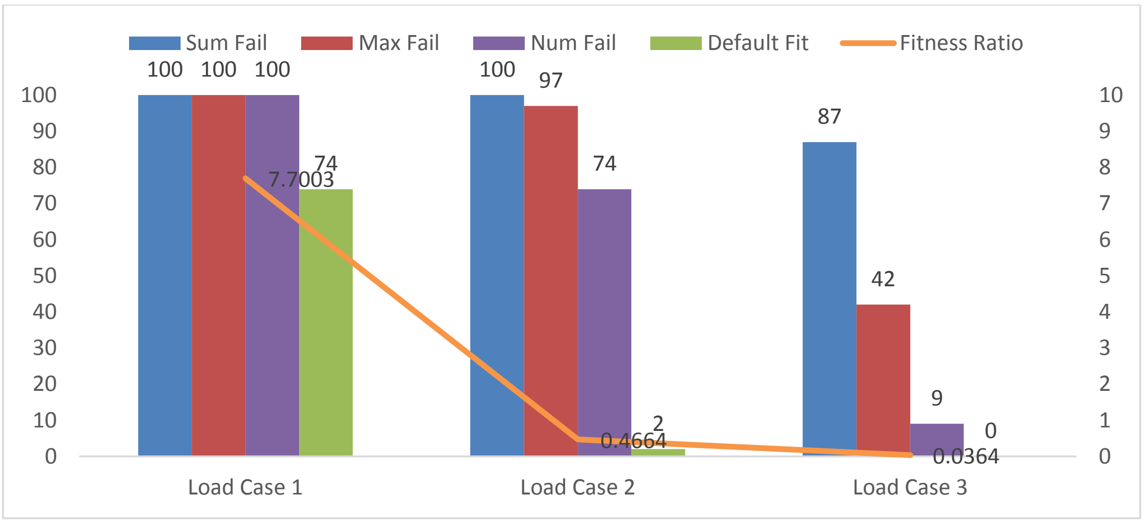


Figure 5.3: Number of successful runs for TC1 out of 100 total runs for increasingly difficult problems

The use of a static penalty, however, performed noticeably less well than the other three methods and was only able to evolve fit solutions in 74 out of 100 runs, indicating a maximum probability of success of 74%. When the problem is made more difficult (i.e. the load increases from 10MN to 15MN, as shown in LC2), this probability of success drops to only 2% for the static penalty method, while the Num Fail method reduces to 74%. Both the summation and maximum failure methods perform extremely well, with all 100 runs using the summation of all failures metric succeeding in finding fit solutions, and only 3 runs using the maximum failure metric failing to find a single fit solution. For the final LC3, the Default Fitness fails to produce any fit individuals, and the Num Fail method only registers a 9% chance of success. The Max Fail method produces less than 50% success (with only 42 successful evolutionary runs), whereas by contrast Sum Fail is still able to deliver 87%.

Further information on the comparative performance of these constraint handling methods across varying degrees of difficulty can be seen in Table 5.6. Therein, the total number of successful generations (out of a total number of 10,000 generations – 100 generations per run for 100 runs) for each constraint handling method and each load case for TC1 can be seen. By using the formula developed in Eq. 5.21, the average number of generations required to evolve a single successful solution for each constraint handling method and each load case can be calculated.

$$Avg.no.gens = \frac{(Total\ successful\ runs * 100) - Total\ successful\ gens}{Total\ successful\ runs} \quad \text{Eq. 5.21}$$

In all load cases, the static penalty method (Default Fit) does not perform as well as the DTF methods (Sum Fail, Max Fail, and Num Fail). This is evident from the lower numbers of both successful generations and successful runs from this method compared to the others. Further discussion is in Section 5.4.

Increasing the problem difficulty for TC1 has a corresponding effect on the efficiency of the varying methods of constraint handling with unfit individuals. The number of generations required to evolve a fit solution increases steadily, while the probability of evolving a fit solution decreases. This correlation is not seen with the use of a static penalty however, as there is no evolutionary pressure towards the generation of more fit solutions. With this approach, evolution of a fit solution is purely based on the initialisation method from the first generation. This means that if a fit solution is not randomly discovered in the first few generations, it will likely not be discovered.

Table 5.6: Constraints - Experimental results from TC1

		Total no. successful gens (out of 10,000)	Total no. successful runs (out of 100)	Avg. no. gens needed to evolve a successful solution
Load Case 1	**Sum Fail**	9833	100	1.7
	Max Fail	9830	100	1.7
	Num Fail	9817	100	1.8
	Default Fit	7222	74	2.4
Load Case 2	**Sum Fail**	9401	100	6.0
	Max Fail	9176	97	5.4
	Num Fail	6593	74	10.9
	Default Fit	194	2	3.0
Load Case 3	**Sum Fail**	7498	87	13.8
	Max Fail	3509	42	16.0
	Num Fail	702	9	22.0
	Default Fit	0	0	N/A

5.3.2. Test Case 2: Cantilevered Truss

The envelope for TC2 is illustrated by Li *et al.* [64], Lee and Geem [76], Khot and Berke [128] and Adeli and Kumar [129]. It describes a simple cantilevered truss with a span of 10,160 mm (400 inches) and a depth of 2,540 mm (100 inches), as shown in Figure 5.4. A single point load F1 was placed at the end of the structure, and the load was varied to see the evolutionary effects of increased difficulty. The three load cases tested were as follows:

1. 1.5MN (feasibility ratio of 0.3811)
2. 3MN (feasibility ratio of 0.0000)
3. 3.5MN (feasibility ratio of 0.0000)

All of the above loading conditions are considerably higher than the benchmark loading of 444,800 N from the literature (which carries a feasibility ratio of 16.7738).

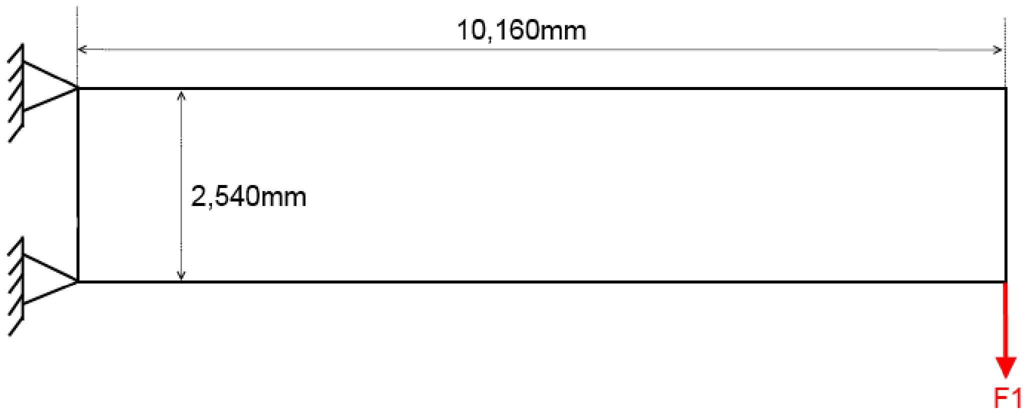


Figure 5.4: Basic cantilevered truss design envelope

As with TC1, an increase in the problem difficulty for TC2 led to deterioration in the efficiency of various constraint handling mechanisms (shown in Figure 5.5). For the first load case, all DTF methods achieved a perfect success rate, with all 100 runs evolving fit individuals, while only 2 Default Fitness runs failed to evolve a single fit individual. Increasing the load from 1.5MN to 3MN for LC2 corroborates the results from TC1 by showing a dramatic drop in the efficiency of both Num Fail and Default Fitness, while both Sum Fail and Max Fail remained unchanged.

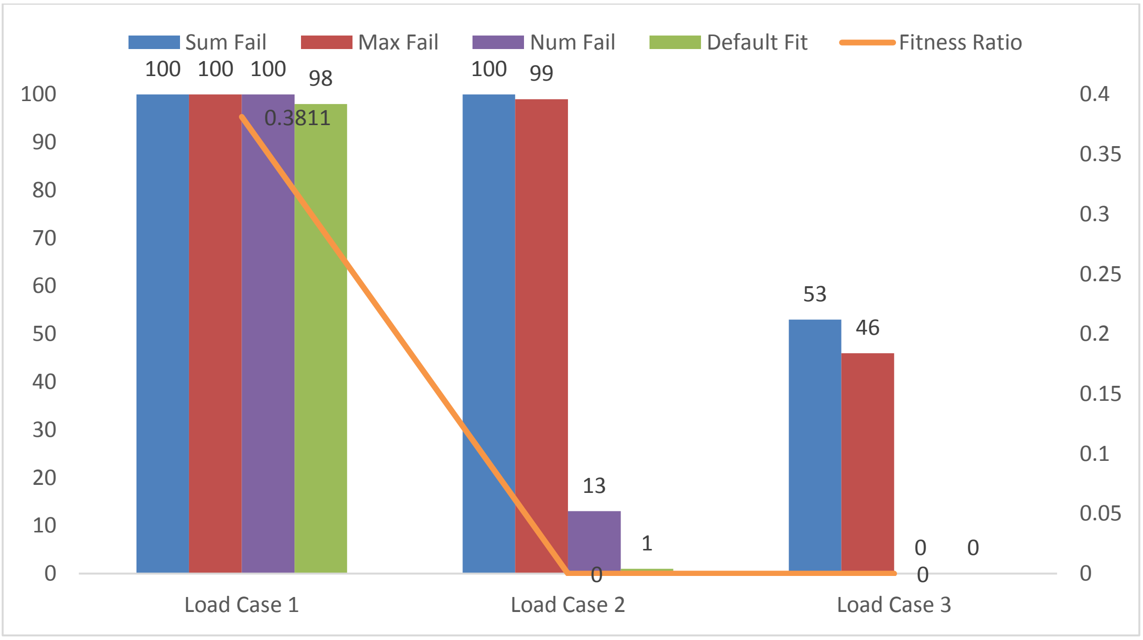


Figure 5.5: Number of successful runs for Test Case 2 out of 100 total runs for increasingly difficult problems

Increasing the load on the cantilevered truss to a maximum load of 3.5MN reduced the probability of evolutionary success when using either the number of constraint violations or the static penalty function to 0. Over 100 runs, each for 100 generations, no fit individuals were found using either method. The best performing DTF method (Sum Fail) reduced the success rate observed in LC2 to 53%, and the average number of generations needed to evolve a successful solution increased more than five-fold from 4.7 to 23.09, as shown in Table 5.7. The use of Max Fit performed poorer still, with only 46 successful runs out of 100.

Table 5.7: Constraints - Experimental results from TC2

		Total no. successful gens (out of 10,000)	**Total no. successful runs (out of 100)**	**Avg. no. gens needed to evolve a successful solution**
Load Case 1	**Sum Fail**	9894	100	1.06
	Max Fail	9892	100	1.8
	Num Fail	9746	100	2.54
	Default Fit	9663	98	1.4
Load Case 2	**Sum Fail**	9530	100	4.7
	Max Fail	9395	99	5.1
	Num Fail	1023	13	21.31
	Default Fit	93	1	7
Load Case 3	**Sum Fail**	4076	53	23.09
	Max Fail	3591	46	21.93
	Num Fail	0	0	N/A
	Default Fit	0	0	N/A

5.3.3. Test Case 3: Cantilevered Truss, Two Loads

The final test case evolved a cantilevered truss with two loads, as described by Deb and Gulati [14], Hajela & Lee [69], Luh and Lin [77], Li *et al.* [64], and Kaveh and Talatahari [65]. The truss sports the same dimensions as those of TC1, with a span of 18,288 mm (720 inches) and a depth of 9,144mm (360 inches). Two identical point loads are placed on the bottom chord of the truss, one at the midpoint and the other at the extreme end of the truss, as shown in Figure 5.6. The three load cases tested examined forces F1 of:

1. 2.5MN (feasibility ratio of 0.1135)
2. 3.5MN (feasibility ratio of 0.0028)
3. 5MN (feasibility ratio of 0.0000).

As with TC2, these loading conditions are far greater than the original design load of 444,800N from the literature (which has a feasibility ratio of 22.6274).

At an initial force F1 of 2.5MN (i.e. a combined load of 5MN), all 100 runs using each of the three DTF constraint handling methods successfully managed to evolve fit individuals, within nearly identical average evolution times of 1.47 and 1.46 generations for the Sum Fail and Max Fail methods respectively. The Default Fitness method did not perform as well, with only 76 successful runs, indicating a probability of success of 76%.

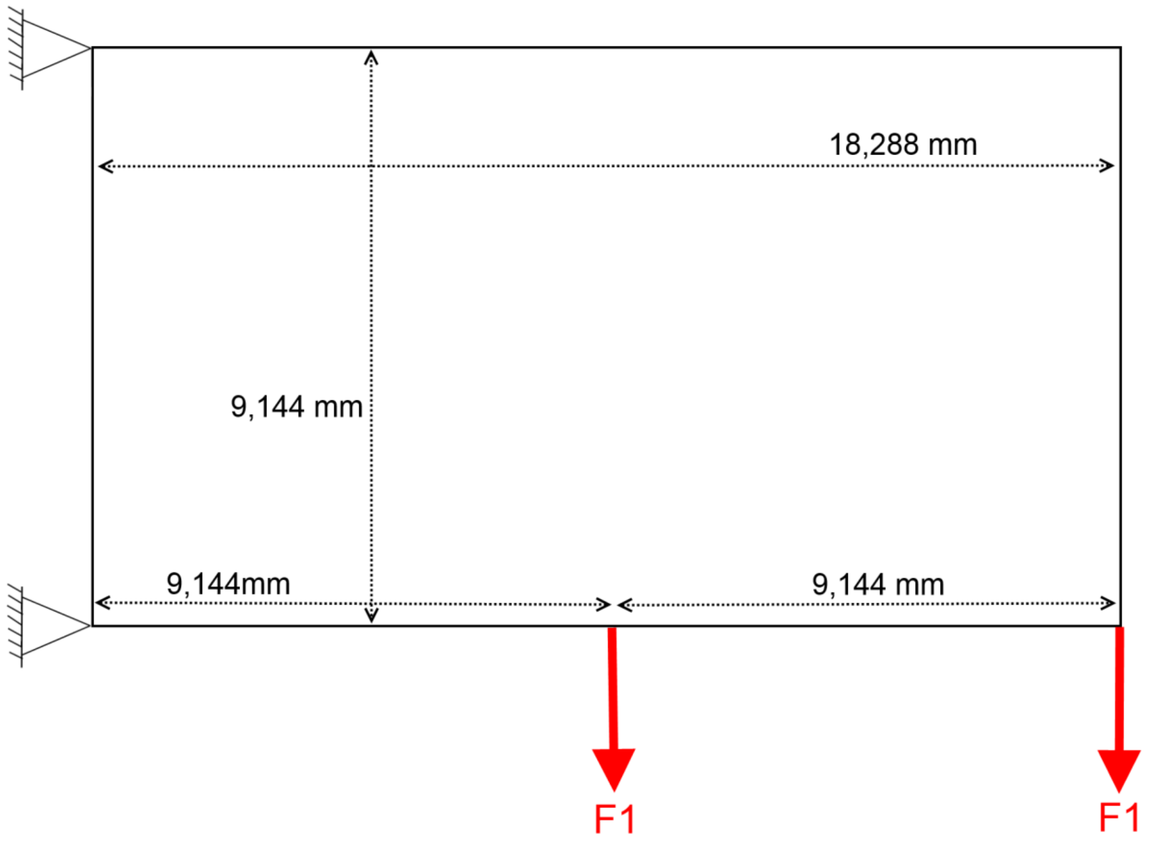


Figure 5.6: Dual-load cantilevered truss envelope

Increasing the force to 3.5MN for LC2 (a combined load of 7MN) saw the Sum Fail and Max Fail methods still maintaining a 100% probability of evolutionary success, but with a marginal dip in the efficiency of the evolution, with the average number of generations needed to evolve a fit solution increasing from 1.5 to 4.2 for both (Table 5.8). The Num Fail method reduced slightly down to 95%, while the static penalty method of Default Fitness being applied to individuals with violated constraints saw a drop in performance from 76 successful runs to just 1 successful run out of 100 total runs.

For a final maximum force F1 of 5MN (a total combined load of 10MN), the difference between the Sum Fail and Max Fail methods of constraint handling becomes clearer, with the former posting a 52% probability of evolutionary success compared to the 37% of the latter.

Just like with the previous two test cases, the number of failures method and the static penalty method reduces down to 2% and 0% success rates, respectively, with a corresponding increase in the difficulty of the problem. This indicates that these are the least appropriate methods to use for handling of violated constraints.

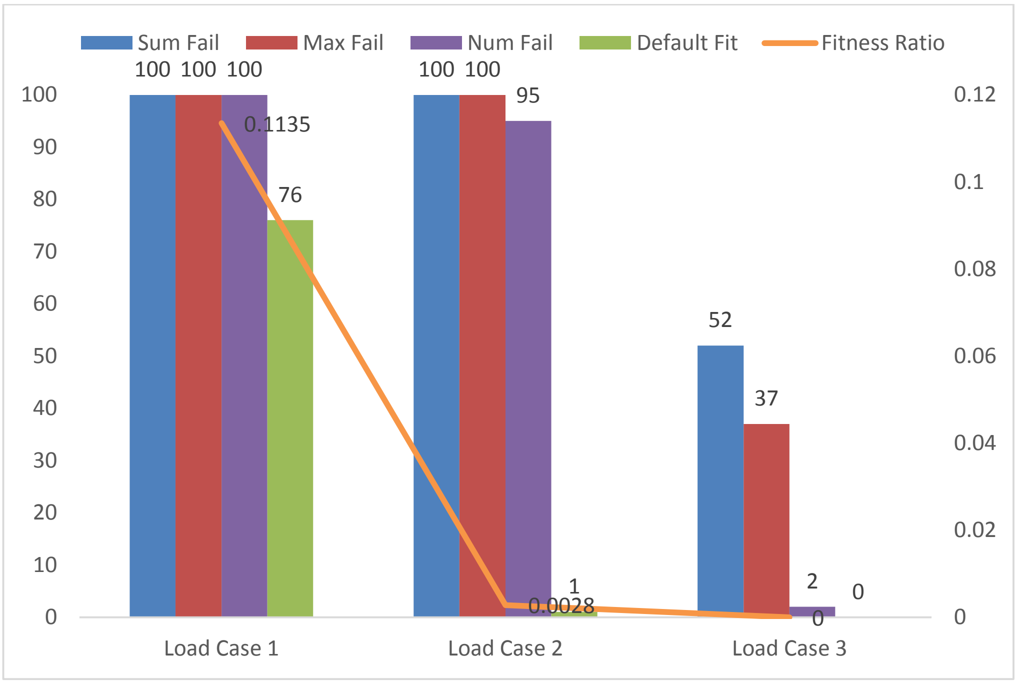


Figure 5.7: Number of successful runs for TC3 out of 100 total runs for increasingly difficult problems

Table 5.8: Constraints - Experimental results from TC3

		Total no. successful gens (out of 10,000)	**Total no. successful runs (out of 100)**	**Avg. no. gens needed to evolve a successful solution**
Load Case 1	**Sum Fail**	9853	100	1.47
	Max Fail	9854	100	1.46
	Num Fail	9836	100	1.64
	Default Fit	7462	76	1.82
Load Case 2	**Sum Fail**	9575	100	4.25
	Max Fail	9580	100	4.2
	Num Fail	8898	95	6.34
	Default Fit	97	1	3
Load Case 3	**Sum Fail**	4083	52	21.48
	Max Fail	2900	37	21.62
	Num Fail	170	2	15
	Default Fit	0	0	N/A

5.4. Discussion

A number of observations can be gleaned from the experiments performed in this chapter. Firstly, the partial enumeration experiments performed for all experimental settings clearly show that increasing the loading on a structure decreases the fitness ratio (as shown in Table 5.9).

Table 5.9: Fitness ratios for all problems and load conditions

Problem Name	Loading	Fitness Ratio	Probability of fit solution in 1st generation
Test Case 1	10 MN	7.7003	45.5%
	15 MN	0.4664	0.75%
	20 MN	0.0364	0%
Test Case 2	Original	16.7738	100%
	1.5 MN	0.3811	74.25%
	3 MN	0	0.25%
	3.5 MN	0	0%
Test Case 3	Original	22.6693	100%
	2.5 MN	0.1135	53%
	3.5 MN	0.0028	1%
	5 MN	0	0%

Table 5.9 also shows that the lower the fitness ratio, the less the probability that the random initialisation method will be able to generate a fit solution in the first generation. An interesting observation is that different problems have different difficulty thresholds. A similar fitness ratio across different problems does not necessarily constitute a similar difficulty level. For example, at a fitness ratio of just 0.3811 LC1 of TC2 has a probability of fit initialised individuals of 74.25%, whereas LC2 of TC1 (with a similar fitness ratio of 0.4664) has a far lower probability of just 0.75%. Furthermore, Figure 5.5 shows that for TC2 LC1 this fitness ratio sees a probability of evolutionary success of 98% for the Default Fit constraint handling method, while Figure 5.3 shows that TC1 LC2 (with a similar fitness ratio) has a probability of evolutionary success of just 2% for the same method.

As mentioned in Chapter 2, the death penalty technique of simply rejecting unfit individuals outright is the most popular constraint handling method. However, when the difficulty of the problem increases such that the probability of discovering a fit individual in the initial generation (i.e. the fitness ratio) tends towards zero (as is the case with the highest loading cases for all of the test cases listed in Section 5.3), this method can no longer be applied.

The next most popular method over the death penalty [109] is the use of a static penalty. This technique has a number of advantages over DTF methods, primarily in that it is extremely easy to implement. However, with a static penalty a single, uniform, “bad” fitness penalty is given to the individual regardless of the actual number of constraint failures or severity of any single failure. This penalty, therefore, remains constant across every individual. This means that for a population of *unfit* individuals all individuals will have the same fitness value and will be treated as performing identically. Since no information on the relative performance of individuals is passed through to the selection and replacement part of the algorithm (as detailed in Section 3.9), the fitness landscape (i.e. the total landscape of fitnesses of each individual solution once evaluated by the fitness function) is zero-dimensional; all individuals in the fitness landscape occupy a single point. This means that there is no way for the evolutionary process to differentiate between separate unfit individuals as each unfit individual is deemed identical in terms of evolutionary performance. Pre-fit evolution is therefore essentially a random search (based on a random initialisation) as the population cannot be sorted in any meaningful manner.

Since there is no selection pressure for any particular solution, it can be seen from the experiments in Section 5.3 that (in the pre-fit phase) a static penalty method will either randomly generate a fit solution within the first 5 generations, or it will fail to evolve a fit solution at all. If the problem is very difficult, such that the probability of randomly producing a fit solution in the first 5 generations is extremely low, then a static penalty method may fail to find a fit individual entirely. It can be seen from Figure 5.3, Figure 5.5, and Figure 5.7 that the performance trend of the Default Fit method would appear to be directly correlated with the trend of the fitness ratio (i.e. the lower the fitness ratio, the less the probability that a static penalty method will successfully evolve a fit solution). In order to uncouple this correlation, constraint handling methods in the fitness function must be a function of the Distance To Feasibility (DTF).

In Section 5.3 it was seen that all DTF methods out-performed the static penalty method in all load cases for all test cases. Furthermore, Figure 5.3, Figure 5.5, and Figure 5.7 show that such methods are not correlated with the fitness ratio, and can hence be considered more applicable to a wider range of more difficult problems. However, of the three DTF methods investigated, the Num Fail method of handling broken constraints was outperformed by the Max Fail method, which in turn did not perform as well as the Sum Fail method. This discrepancy is due to the fact that insufficient information about constraint failures is being given to the fitness function. Consequently, the algorithm becomes stuck in a local optimum. This is due to the fact that the evolutionary process cannot differentiate between individuals with a similar *number* of failed constraints. For example, if two individuals both have one failed constraint, but one fails by 20%, while the other fails by only 3%, they are both treated as equal in terms of fitness. This stalls the search process.

While the use of the Max Fail DTF method does produce a multi-dimensional fitness landscape (allowing the algorithm to distinguish between relative failed individuals), in some cases there is still not enough information being transferred to the final fitness value for each individual on the failure of all constraints. Because of this, the evolutionary process cannot differentiate between individuals with multiple constraint failures. For example, if one individual fails only one constraint by 10%, while another individual fails one constraint by 10% and five constraints by 9%, the evolutionary process will treat both individuals as having equal performance, even though one is far worse than the other. With the use of a Max Fail-style constraint handling method, individuals can be evolved with multiple counts of very low constraint violations. However, these individuals may occupy an area of the search space that is not near any truly fit solutions. As such, the evolutionary process gets stuck in a local optimum searching around the best known (but still unfit) solution and cannot navigate to a different area of the search space without a large increase in the mutation rate. Although mutation rates which adapt and react according to fitness values have been described [131], a simpler and more effective solution in this case is to merely provide more information to the fitness function on the failure of all constraints in an individual.

All performed experiments indicated that the best constraint handling method for structural engineering truss optimisation was to pass on as much information as possible on all failed constraints to the final fitness of the individual. This minimises the risk of the algorithm getting stuck in a local optimum in the pre-fit evolutionary phase (although this still can occur with very high difficulty problems and is a characteristic of heuristic algorithms) and maximises the probability of evolutionary success. In every case described, the summation failure method developed for this application in this chapter either out-performed or matched all other methods.

5.5. Conclusions

In cases where the representation space is very large and the feasibility ratio is very low (i.e. the problem difficulty is such that very few feasible individuals exist in the space), a static penalty approach for constraint handling (while still enjoying popularity in less theoretically difficult problems) is not the most effective, as the fitness landscape is zero-dimensional. Experimental results show that there exists a theoretical difficulty limit above which the use of a static penalty will not evolve any feasible individuals. In order to create an n-dimensional fitness landscape (where n represents the number of fitnesses used), information on the distance to feasibility of failed individuals must be passed through to the fitness function. If a constraint is violated, the penalty to be applied to the overall fitness must be related to the severity of that violation.

Caution is to be urged, however, when adding relative performance information into the fitness of each individual. As shown Section 5.4, failing to provide enough information can actually have a negative effect on the evolutionary performance of an algorithm, allowing the evolutionary process to get stuck in a local optimum. To this extent, the method of accurately generating penalty values for failed individuals by calculating the normalised value of any failed constraint shown in Eq. 5.19 is preferable. This method, developed for this application, allows the evolutionary algorithm to accurately distinguish between differing constraint violations between individuals. Experiments conducted in this chapter concluded that the more information on the distance to feasibility of an individual the fitness function is given, the better the chance that the selection and replacement mechanism of the evolutionary algorithm will have of differentiating between individuals with multiple constraint failures in a highly competitive fitness landscape.

5.6. Summary

A key challenge for designers when using evolutionary approaches is the need to find an accurate metric that will provide sufficient evolutionary pressure to evolve viable solutions. A need was, therefore, identified to find an effective method of applying such evolutionary pressure on unfit individuals in order to evolve fit individuals. Through a series of experiments evolutionary pressure was shown to be more important as the problem becomes more difficult.

An increase in problem difficulty can come through a change in loading conditions, a change in the constraint limits, or any combination thereof. Variation in the fitness landscape can occur, even if the representation space remains constant. This variation was observed through a number of experiments, which demonstrated the effectiveness of varying methods of penalty function applications. Appropriate methods for handling both single and multiple failed constraints were discussed, and a new method of calculating penalty values for unfit individuals was introduced for this application, based on the normalised values of all failed constraints. Finally, a recommendation was made that as much information as possible on the magnitude of the failures of all aspects of an individual be included in the overall applied penalty to the fitness of that unfit individual, in order to minimise the risk of the evolutionary algorithm becoming stuck in a local optimum in pre-fit evolution. The following chapter puts these recommendations into practice by employing an extremely large representation space in order to solve ill-defined problems.

Chapter 6. Discrete Truss Optimisation via Node Placement and Variation

Continuum topology optimisation in design methods represent the current cutting edge in engineering design optimisation [12, 132, 133]. In large scale civil and structural engineering projects however, manufacturing solid structures fully optimised using these techniques is generally prohibitively expensive and difficult [12, 7]. Discrete beam structure optimisation methods are more appropriate for large scale designs, as they allow regular elements and construction methods to be used, leading to savings in cost and weight over more traditional construction methods. Classical discrete topology optimisation, such as those methods discussed in Chapter 2 and Chapter 4, follows a ground structure approach, with all possible node and beam locations being specified *a priori* and the algorithm selecting the most appropriate configuration from the given list of options [10]. These methods have been shown to be inherently restrictive as they severely limit the representation space to what is defined; a larger representation space can be more effectively navigated to find the global optimum [14].

While continuum topology optimisation is arguably the most effective optimisation method, it has inherent flaws [11, 7], primarily in that the computational cost of generating a solution increases exponentially with the physical size of the structure. As such, large-scale structures cannot yet be optimised due to the current limitations of computing power [7]. This is an area in which ground structure optimisation excels [12], yet there are still limitations in the techniques employed. Luh and Lin [77] make particular note of the fact that the most optimal solutions for discrete methods can only be found by simultaneously considering optimisation of size, shape, and topology, as each has an effect on the others. This concept was proven in Chapter 4, where it was demonstrated that simultaneous evolution of structural shape, structural topology, and member sizing was not only possible with the use of two chromosomes, but that improvements over traditional ground structure methods could be thus achieved.

Rozvany [13] explicitly stated that globally optimal solutions for discrete optimisation problems *cannot* be found with enforced layouts and a small number of members, i.e. the representation with a ground structure approach is too constrained to effectively find the global optimum of the overall search space, regardless of the size of the representation space. Deb and Gulati [14] confirmed this by proving that even limited variation of node locations in a ground structure approach led to improved fitness results. Chapter 5, however, showed that a problem arises in that an increase in representation capabilities spreads the search process over a much wider area than before. To counteract this, an improved technique for applying penalty functions to unfit individuals was demonstrated in Chapter 5, thereby enabling the search process to quickly and efficiently navigate through the initial unfit-laden fitness space towards areas populated by fitter solutions.

Further unconstrained discrete representations from the literature such as the principal stress line method provide even better results [13], but can only be used with compliance minimisation problems [51]. Therefore a wholly unconstrained discrete optimisation method that can improve any fitness value is likely to be a more effective tool. In order to develop such a method, a number of hypotheses are proposed:

- That it is possible to create an unconstrained grammatical representation, given minimal information for the desired solution;
- That such an unconstrained representation will be able to evolve viable solutions;
- That minimal information should be required for a viable solution to be evolved;

This chapter aims to test these hypotheses, along with whether better solutions achievable with more or less provided information.

The remainder of this chapter is structured as follows. Section 6.1 provides an introduction to the topics discussed in the chapter. Section 6.2 describes the importance of the design envelope in both discrete and continuum optimisation. The approach to node generation within this design envelope is discussed in Section 6.3, including a number of different methods of describing node locations using a grammatical representation. Methods of connecting nodes are discussed in Section 6.3.6, and the GE ripple effect is described in Section 6.3.7. A number of benchmark numerical examples from the literature are detailed in Section 6.4, including proof-of-concept examples (Section 6.4.1) and non-regular truss forms (Section 6.4.4). The implications of these experiments are discussed in Section 6.5, and the conclusions are drawn in Section 6.6.

6.1. Introduction

Standard practice for ground structure optimisation follows a binary-style approach whereby the full set of possible solutions are specified beforehand with the algorithm adding or removing pre-existing elements (Figure 6.1). However, this limits the search space to only what is explicitly defined in the algorithm. Potential fit solutions that lie outside the representation space are never considered.

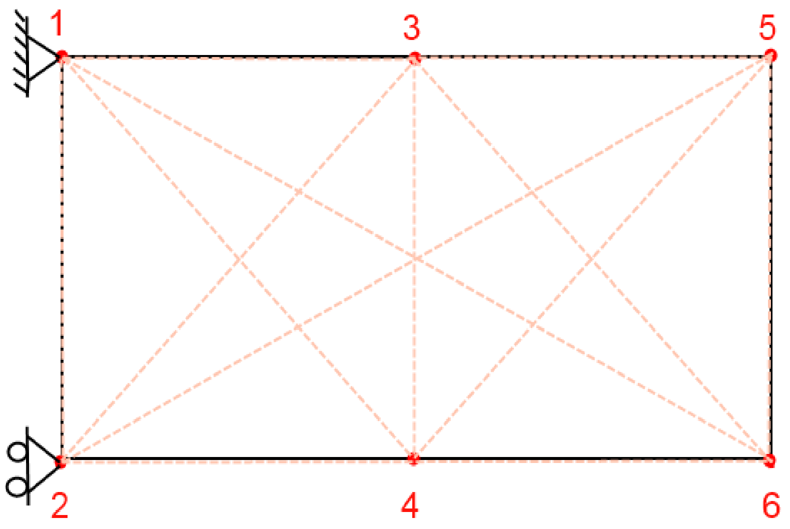


Figure 6.1: Traditional 16-member, 6-node ground structure optimisation approach [14]. Red lines represent possible edge locations.

If even a slight variation of the placement of nodes can lead to a better fitness, then a question must be asked about whether the ground structure-style approach of evolution and optimisation of node connections with fixed node locations is the most efficient method of discrete structure generation. Since a larger representation space can be efficiently managed by carefully manipulating the fitness values of unfit individuals, the size of the representation space should be of non-primary concern to discrete methods.

Another important omission to note is that traditional 2-dimensional ground structure optimisation methods roundly ignore the fact that co-planar members which intersect *by definition* create an extra nodal connection. Along with idealised material specifications (including unrealistic stress limits and cross-sectional area optimisation that exceeds manufacturing precision capabilities, as demonstrated in Chapter 4), there is an apparent assumption that solid members can pass through one another with no structural effect. Classical ground structure optimisation problems, such as the 10-bar truss problem described in Section 4.3.1 or the 15-member structure shown in Figure 6.1 are therefore unrealistic. In reality, the 6-node, 10-bar truss from Figure 4.2 should be more accurately described as an 8-node, 14-bar truss, as shown in Figure 6.2.

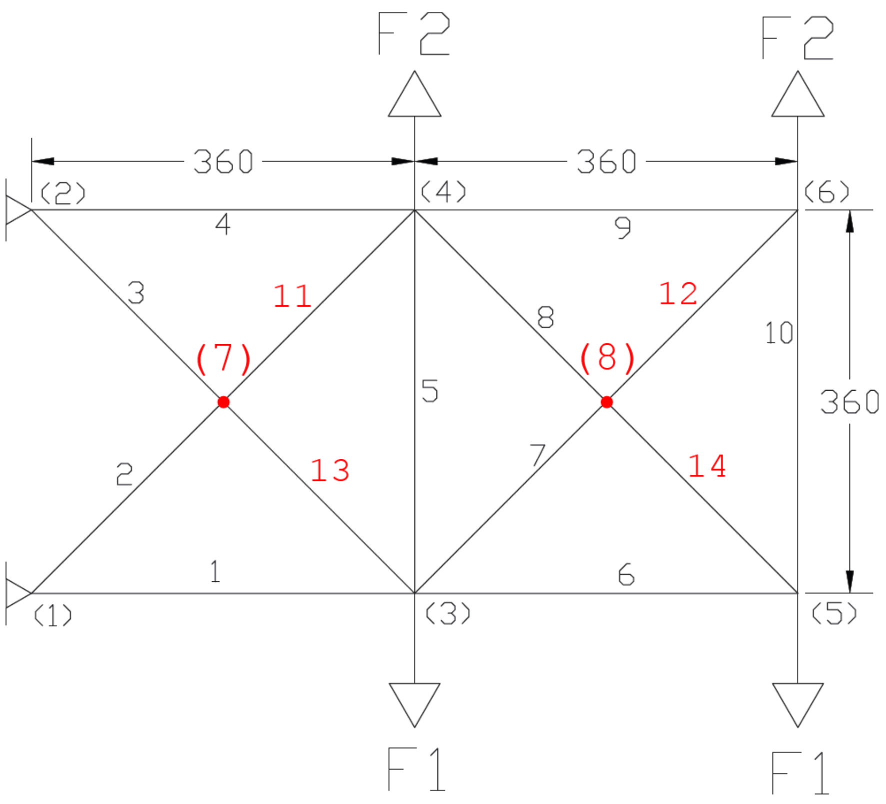


Figure 6.2: 10-bar truss problem more accurately described as a 14-bar, 8-node truss problem. Extra nodes and edges indicated in red.

This 14-bar truss has completely different characteristics from the 10-bar example, as the four longest members are now effectively halved in length. This has a notable effect on buckling of compression members, as these long diagonal members are now braced at their mid-points. In order to generate and optimise realistic structures, every meeting of multiple members should be treated as a nodal connection, splitting all connected members into shorter members and changing the solution outcome.

In conclusion, if even a slight variation in the location of nodes can lead to improvements in fitness, a question must be asked over whether or not methods which use fixed representations can truly be considered the most appropriate discrete optimisation methods. A radical suggestion would be to effectively invert this process by to selecting and indicating the presence and location of the actual *nodes* themselves, rather than the edges connecting fixed (or minimally variable) points. Variable nodes could then be connected using a simple and repeatable method such that the emphasis were on the node locations themselves rather than the interconnectivity between them. The central hypothesis that this chapter aims to examine is whether or not this method would demonstrate any advantages or disadvantages over the traditional connection-focussed ground structure method.

6.2. Design Envelope

Kawamura *et al.* [73] noted that, as with observable nature, the environment has the greatest effect on the evolution of an individual, and that any individual placed in a well-defined environment should evolve successfully to suit it. The problem definition in continuum optimisation begins with the definition of this environment: the design envelope. This envelope defines the maximum and minimum boundaries in all dimensions for the location of any element of the design; elements (be they nodes, edges, or design material) that lie outside the design envelope are not considered, and indeed in most cases are not permitted to be generated. Discrete optimisation does not technically have a similar concept, but the design envelope could be said to be described by the outermost node locations of the structure [51]. In order to replicate the continuum method using discrete elements, the design envelope must incorporate limits for all possible element locations. These limits can be set by either penalising designs that fall outside the limits, or more effectively, by only allowing elements to be placed within (or on the boundary of) the limits.

Considering that the hypothesis of this chapter is that a node location-based discrete approach will be capable of generating fitter solutions than a fixed node connectivity-based discrete approach, a representation must be built with the capability of placing nodes anywhere within the design envelope. A GE approach introduces the possibility of defining a design envelope by providing dimensional limits for the nodes within the structure. As defined in Chapter 3 and demonstrated in Chapter 4, two chromosomes can be used to define the structure itself [117]. The first, Chromosome A (Ch.A), defines the topological form, or shape, of the structure. This can be done by using the genes in Ch.A to map to the locations of any number of nodes within the physical problem space, or to select more nodes to add to the structure (Figure 6.3). Once node locations are set by Ch.A, a simple and repeatable algorithm can be used to connect nodes together to form the truss structure. Once connections are made, the second chromosome (Ch.B) is used to set member sizings from a pre-defined list of available material options.

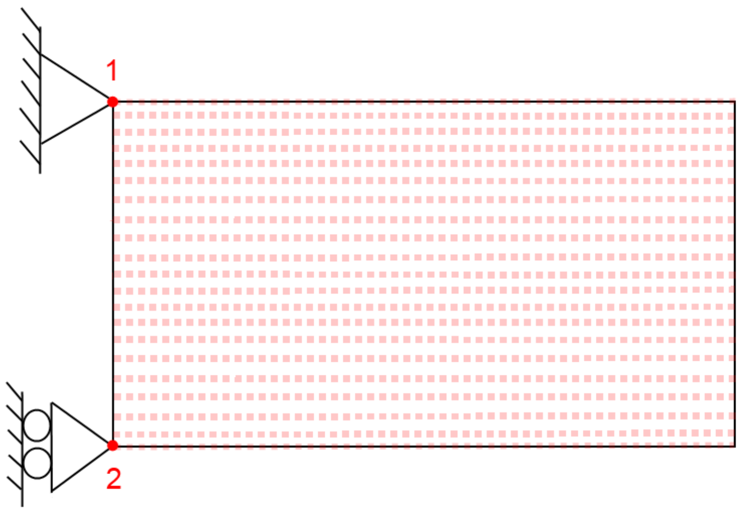


Figure 6.3: Node location-based optimisation approach. Possible node locations are indicated by red dots.

6.3. Node Generation

Deb and Gulati [14] broke down the node characteristics of an individual into two sections: essential nodes and non-essential nodes (they use the phrases “basic” and “optional”). Essential nodes are those nodes that are necessary for the problem definition. These represent the location of all fixed and loaded points in the structure. Any other nodes are viewed as non-essential (i.e. a theoretical solution to the problem can be found using only the essential nodes, as shown in Figure 6.4).

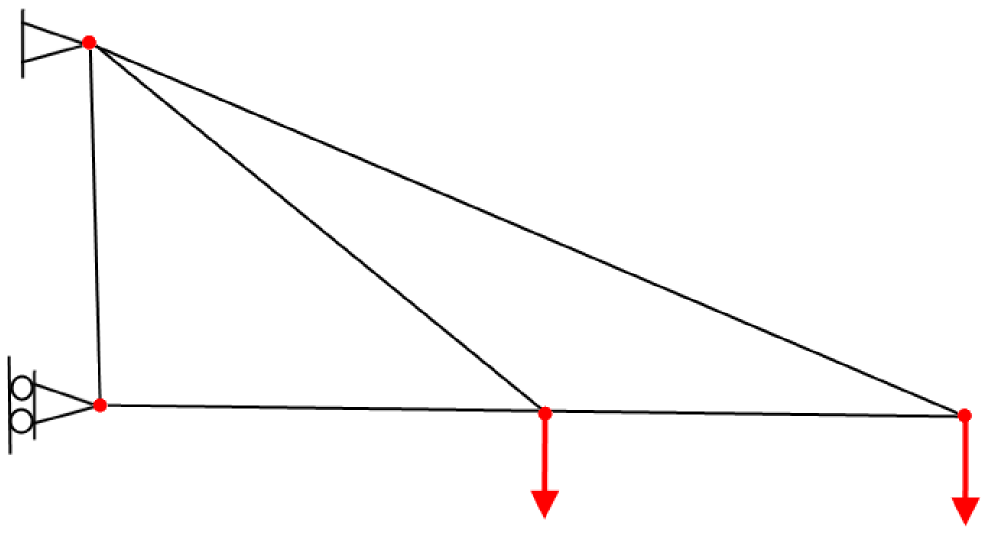


Figure 6.4: Stable truss configuration, using only essential nodes (no non-essential nodes present)

To use this analogy, the essential nodes must first be defined: all fixed nodes and loaded nodes. These are the only fundamentally necessary nodes in the structure. There is no need to specify “boundary” nodes at the edges of the design envelope. If these are deemed necessary by the structure, they will be subsequently evolved.

Loads can be applied to nodes as a three-dimensional vector (with x, y, and z co-ordinates), allowing for a point load of any size and direction to be placed on any node in the structure. For example, a vertical downward force of 100 N would be described as:

```
[0, -100, 0]
```

Figure 6.5: Loading by force vector

In this manner any point load in any direction for any load can be specified, simply by specifying the node location and force vector.

In a similar manner to point loading, fixed node locations can be indicated by a triple tuple of Boolean values, with each indicating a fixing in that particular dimension. For example, a fully pinned support would be indicated by:

```
[True, True, True]
```

Figure 6.6: Node fixing indicated as triple tuple of Boolean values

while a pinned support with a rolling bearing on the x-axis would be indicated as:

```
[False, True, True]
```

Figure 6.7: Pinned / rolling bearing node fixing

All other nodes in the structure (i.e. those nodes which are neither loaded nor providing reactionary support) can, therefore, be deemed to be non-essential, with one caveat: if all essential nodes were co-linear (that is, if all essential nodes were to exist on the same line), it would be possible for the grammar to generate non-optimal, kinematically unstable, one-dimensional structures, as shown in Figure 6.8.

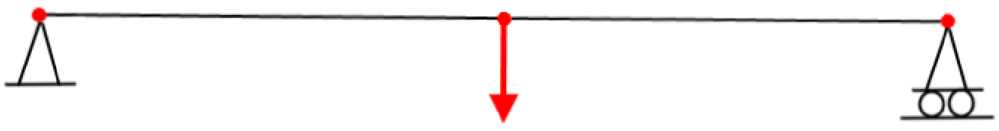


Figure 6.8: Kinematically unstable structure resultant from co-linear essential nodes

Because of this, there should be at least one pre-specified node for each structure which is not co-linear to all other essential nodes. This would ensure that all generated structures are at least two-dimensional and, as such, are all kinematically stable and theoretically valid (actual fitness notwithstanding). While three dimensional structures are indeed possible, unless stated otherwise, all experiments in this chapter are performed in two dimensions.

Once essential nodes are defined, the grammar selects any number of non-essential nodes for the interior of the structure. The recursive capabilities of Grammatical Evolution can easily be employed to allow the grammar to select irrespective of the number of required nodes, by allowing a simple production rule to either add a single node or to add a single node and to call itself again, as detailed in Figure 6.9.

```
<node_iter> ::= <node> | <node>,<node_iter>
<node> ::= [<x>, <y>, <z>]
```

Figure 6.9: Simple recursive grammar segment showing potential for multiple node selection through the production rule `<node_iter>`

The grammar next needs to be able to indicate the actual locations for potential non-essential nodes. This requirement leads to a slight complication. Specifically, the grammar needs to be able to select from within a range of numbers (i.e. the boundaries of the design envelope) in such a way that it can effectively cover the entire design envelope. This is a problem in itself. A major drawback of the BNF grammar format is that it cannot specify a range of numbers from which to choose from, without first defining every individual available number. Thus, for a very large range of numbers this procedure would be extremely tedious and inefficient. From a coding point of view this would require adding every selectable value, as any small change in the geometry of the design envelope would lead to the addition or removal of large amounts of code from the grammar. To overcome this, an effective method of selecting a single value from a within a range of values using a BNF grammar format must be created. This method must be repeatable (i.e. the same Ch.A input will generate the same phenotype), fast, and accurate. It must also use as little genetic information as possible and must provide as broad a representation as possible. Sections 6.3.1 to 6.3.4 explore various approaches of achieving this.

6.3.1. Direct Chromosome to Phenotype Mapping

The simplest way to generate node locations within a variable space would be to map directly from the chromosome to the design space, in the style of a GA, with each codon on the chromosome representing either an x or y value (two dimensions are shown here for simplicity but each method discussed can be extended to three dimensions). This would entail setting the maximum permissible codon value greater than or equal to the greater dimension of the design space such that each dimensional node value will equal the codon value modulo the maximum permissible value for that vertex, as shown in the grammar excerpt in Figure 6.10.

```
<span> ::= 18288
<depth> ::= 9144
<node> ::= [<x>, <y>]
<x> ::= <GECodonValue> % <span>
<y> ::= <GECodonValue> % <depth>
```

Figure 6.10: Grammar excerpt showing direct mapping from chromosome to design space

While this method requires a very small chromosome (comparatively speaking) in that only a single codon is required per dimensional value of each node (i.e. one codon per x value, one per y value, one per z value), a few key problems arise. One is that mutation of a single gene would equate to mutation of an entire dimensional value of a particular node. Since genetic mutation is completely random, this would have drastic effects on the phenotype of the individual, thereby prohibiting small mutation steps. At a much higher level however, this method is severely limits the representation capabilities of the grammar. Figure 6.10 shows a maximum span of 18288 mm (in the x-dimension), with a maximum depth of 9144 mm (in the y-dimension). Since this grammar can only select numbers between 0 and these values for the respective dimensions, it can by definition only allow for regular square or rectangular design spaces to be generated. If used in three dimensions, this would result in the generation of a cube or rectangular box. Since the creation of each dimension is independent of the other (i.e. neither is a function of the other) no complex shapes can be produced. If, for example, a non-standard design envelope such as a curved design envelope were to be required, a direct chromosome mapping would not be possible without heavy manipulation of the code. To accommodate for variations in the geometry and size of the design envelope with different problems, the grammar would need to be re-written for every problem, which would be exceedingly tedious and inefficient.

6.3.2. Non-Terminal Mapping

Another suggestion would be to treat the design space as a sequence of non-terminal integers, as shown in the grammar excerpt in Figure 6.11:

```
<node> ::= [<x>, <x>]
<x> ::= int(<n><n><n><n><n>)
<n> ::= 0|1|2|3|4|5|6|7|8|9
```

Figure 6.11: Grammar excerpt showing node generation using non-terminal mapping

This particular method would be highly efficient at localised hillclimbing, as any node mutation of any single dimensional value would be well segmented, thereby allowing for a high degree of evolutionary control. For example, mutation of the first digit would create a large phenotypic change of the order of tens of metres, but mutation of the fifth integer would have a comparatively small change, in the tens of millimetres. As such, given the right design problem, this approach would be a highly efficient method of quickly finding the global optimum.

However, this method has a number of significant drawbacks. First, it requires a relatively long chromosome to be used, as each node would require five genes for an x-value and a similar number for a y-value (with more again for a z value if a three-dimensional representation space is employed). Furthermore, the method will only work effectively, if the maximum value (i.e. the range/span of the design) is a multiple of 10,000 (dependant on the number of digits in the maximum permitted value).

A grammar like this would have difficulty representing a search space if, for example, the maximum value of one dimension was 18,288 (as is the case with the 10-bar truss optimisation problem detailed in Figure 4.2). The first digit would be simple (either 0 or 1), but all subsequent digits are less so. If the first digit were 0, then the second digit could be any integer between 0 and 9. If, however, the first digit were 1, then the second digit could only vary between 0 and 8. This would have a cascading effect on the remainder of the digits in the sequence. Very quickly this would become a highly complex tree. For a single problem, a grammar could be created to generate an efficient solution, but if the problem definition changed, it would require a complete re-writing of the grammar, which would render the method highly inefficient.

6.3.3. Random Number-Based Node Generation

A random number generator (RNG) such as the Mersenne Twister [123] can be used to select from a pre-specified range of numbers using the python call `random.randrange(from, to, step)`. `Step` in this instance refers to the resolution, or step size that the random number generator can select from, within the range (`from, to)`. With the use of an RNG, problems arise with the repeatability of each action. Unless a random seed is set, a different number will be returned every time the RNG is used, and no genetic information would be transferred between generations. To counteract this, each time a node was to be generated a random seed would need to be set for each separate coordinate. This can be done by taking a codon directly from the chromosome using `<GECodonValue>`, as shown in Figure 6.12.

```
<span> ::= 18288
<depth> ::= 9144
<node> ::= [<set_rand >, <x>, < set_rand >,
<y>]
<set_rand> ::= random.seed(<seed>)
<seed> ::= int(<GECodonValue>)
<x> ::= random.randrange(0, <span>,
<resolution>)
<y> ::= random.randrange(0, <depth>,
<resolution>)
<resolution> ::= 10
```

Figure 6.12: Grammar excerpt showing RNG-based node generation, 10mm resolution

Importantly, a sufficient quantity of seeds must be provided such that the random number generator can create enough nodes to populate the entire design envelope. If the maximum permissible size of each gene is too small (see Section 3.6.1), then there will not be enough unique random seeds such that all possible node locations generated by terminals within the grammar are covered. Due to this, the maximum permissible codon value must be set greater than or equal to the largest dimension of the design envelope. This will ensure that there are *at least* as many unique individual random seeds as desired node locations. To illustrate this, Figure 6.13 shows a basic rectangular design envelope, with a span of 10 and a depth of 7.

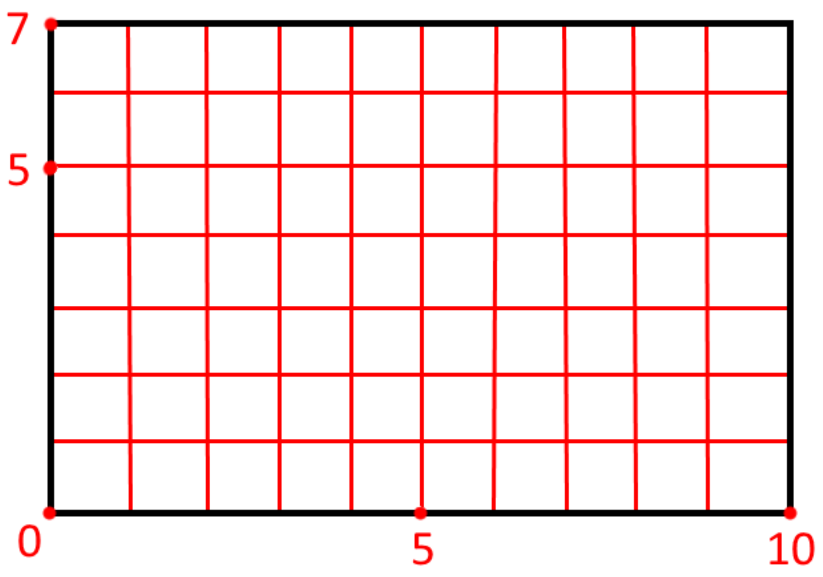


Figure 6.13: Simple design envelope for random number-based node generator

In this instance, if the maximum codon value were set to 10, there would be 11 potential codon values: 0 to 10 inclusive, and hence 11 unique random seeds. Since there are less than 11 possible choices for the y-dimension (8 choices, 0 to 7 inclusive), and there are exactly 11 choices for the x-dimension, this maximum codon value is capable of generating a sufficient quantity of unique random seeds to cover the entire space. However, if the maximum codon value was set to 7, then, while there would be sufficient quantity of random seeds to cover the y-dimension, there would not be sufficient unique seeds to fully explore the x-dimension, and some values would remain unused.

Since this random number-based method essentially places nodes on a grid pattern within given boundaries, the grid resolution itself can be easily changed. If a smaller node resolution is set, e.g. a node every *n* mm, then the max codon value can be set to *span/n* of the greater dimension (as shown in Eq. 6.1).

$$\text{CODON\_A\_SIZE} \geq \frac{\max(\text{span}, \text{depth})}{\text{resolution}} \qquad \text{Eq. 6.1}$$

Using an RNG to generate structures has both positive and negative aspects. As with the direct mapping method described in section 6.3.1, a single codon from Ch.A is responsible for each dimensional coordinate for each node. This means that mutation of any single codon would have a completely random effect on the phenotype of the individual, giving the algorithm little or no hillclimbing ability. There is no way for the program to differentiate between a “large” mutation or a “small” mutation (in terms of phenotypic effect) as all actions will have equally random effects. Since mutation in GE is carried out on a per-codon basis (i.e. each codon has the same percentage chance of mutating), and since mutation of any codon which maps to a random seed for a node’s x or y value would completely change the phenotype of the structure, mutation is effectively amplified. Conversely, since Grammatical Evolution is a heuristic method, this increased random search would theoretically mean that a wider area of the representation space is covered, with less chance of the algorithm getting stuck in a local optimum.

Reducing the Representation Space

With a heuristic evolutionary method an overly large representation space can foil the search process before it has even begun, as there are simply too many possible outcomes to search through, especially when the ratio of fit to unfit individuals is very small. Michalewicz conducted a study [130] to study the evolutionary effects of a comparatively small ratio (between 0% and 0.5% in all test cases) of fit individuals to unfit individuals in the representation space. That study found that such a small ratio can make effectively finding any fit individuals in the representation space extremely difficult for the evolutionary algorithm, without careful management of the constraints (as detailed in Chapter 5). Bohnenberger *et al.* [72] also proposed the idea of reducing the representation space, but their solution was simply to limit the representation capabilities of their algorithm, which potentially had a detrimental effect on the quality of their overall solutions. When seeking to reduce the size of the representation space, techniques should be used which endeavour to preserve the overall quality of potential solutions. This can be accomplished by both limiting the resolution of the available node locations in the initial generations and by setting universal materials across each individual in the first generation.

With python's random number-based node generation, this method would be capable of placing a node anywhere within the given structure boundaries on a grid of 1mm using the `step` operation from the `random.randrange(from, to, step)` function. The representation space could be effectively reduced by limiting the step size to a grid of 100mm at the start of the evolutionary process. If a predefined limit were to be reached (in this particular instance once a single fit individual has been found), the grid could be expanded to a mesh resolution of 10mm, increasing the representation space. The net result of this would be that the overall end representation space would remain the same, but the search would move more quickly towards areas of the representation space populated by a higher percentage of fit individuals.

6.3.4. Percentage-Based Node Generation

Combining the methods of RNG-based node generation from Section 6.3.3 and non-terminal mapping from Section 6.3.2 creates a grammar which generates nodes from within a select range of numbers, *and* with a well-segmented representation, which gives fine-grained control to mutation, thereby allowing for localised hillclimbing. If, instead of allowing a node to be placed at a grid of every millimetre, node locations were instead based on a *percentage* of the overall dimensions of the structure, this would then eliminate the need for a random number generator to be used, as the grammar would no longer be selecting from within a range of numbers but would be selecting a percentage of a given maximum value. Instead of mapping directly from the genotype to the phenotype, the grammar would specify that both the x- and y-values of the node would be a percentage of the order of _ _._ _% of the maximum permissible value for that dimension (the span or the depth of the structure). The basic grammar excerpt shown in Figure 6.14 illustrates the principle.

```
<span> ::= 18288
<depth> ::= 9144
<i_node> ::= (<percentage>, < percentage>)
<percentage> ::= <n><n>.<n><n>
<n> ::= 0|1|2|3|4|5|6|7|8|9
<node> ::= (<x>, <y>)
<x> ::= <span>/100 * <i_node>[0]
<y> ::= <depth>/100 * <i_node>[1]
```

Figure 6.14: Grammar excerpt showing percentage-based node generation

The grammar representation shown in Figure 6.15 only has a range of between 00.00% and 99.99%; this grammar will never be able to generate a full 100%, and no node will be able to be placed at that very boundary of the design envelope. This can be rectified with the use of a multiplier, which can be added to the percentage value such that an evolved value of 99.99% would equal 100%, but 0% would still remain 0%. This multiplier is defined in Eq. 6.2:

$$\mathrm{x}' = \frac{\mathrm{span}}{100} * (\mathrm{x} + (\mathrm{x} * \frac{100 - 99.99}{99.99})) \qquad \text{Eq. 6.2}$$

This addition ensures that the full range of 100% of the design space can be covered by the grammar, with an equal bias for all percentage values, as shown in the amended grammar in Figure 6.15.

This percentage-based grammar, newly developed by the author for this specific application in this thesis, now utilizes the best aspects of all previous node generation options. The grammar is entirely non-biased towards any particular outcome, and it exhibits high locality. The coordinates for each node are well-defined, and the relatively high number of codons required for each coordinate allows for fine-grained mutation and crossover control. This itself allows for local hillclimbing in the search space.

```
<span> ::= 18288
<depth> ::= 9144
<i_node> ::= (<percentage>, < percentage>)
<percentage> ::= <n><n>.<n><n>
<n> ::= 0|1|2|3|4|5|6|7|8|9
<node> ::= (<x>, <y>)
<x> ::= <span>/100 * (i_node[0]+(
i_node[0]*0.00010001))
<y> ::= <depth>/100 * (i_node[1]+(
i_node[1]*0.00010001))
```

Figure 6.15: Grammar excerpt showing percentage-based node generation with modification to allow for 100% range.

6.3.5. Grammatical Bias and Regularity of Structures

One of the major benefits of GE's use of a formal grammar structure in defining its individuals is the ability to add bias to the grammar [48]. The question of bias in the grammar is an interesting one. The percentage-based grammar described in Figure 6.15 has no bias towards any particular outcome with regards to the locations of nodes; it is equally likely that that node can appear anywhere within the structure.

Bias can be added to this grammar by adding extra options to the node generation process. Line 4 of the grammar shown in Figure 6.15 shows only a single terminal option; the rule `<percentage>` will always return a figure in the range 00.00% - 99.99%, with an equal probability for all options. This rule could be changed to add bias towards different outcomes, as shown in Figure 6.15:

```
<percentage> ::= 0 | <n><n>.<n><n> | 99.99
```

Figure 6.16: Line 4 from Figure 6.15, amended to add bias towards boundaries

Now the grammar is biased towards placing nodes at the boundaries of the design envelope. There is a 1 in 3 chance that a node will be placed at any of the three options, meaning that cumulatively the probabilities of node location are as follows:

Table 6.1: Probabilities of node placement locations

0% (lower boundary)	33.33%
0.00% - 99.99%	33.33%
99.99% (upper boundary)	33.33%

Broken down even further, each location in the range 00.00% - 99.99% now has a probability of 0.0033% of selection (see Eq. 6.3).

$$33.33\% \div 10{,}000 = 0.0033\%$$

Eq. 6.3

This means that cumulatively, the probability of the grammar placing a node at either the upper or lower boundaries of the structure is:

$$33.33\% + 0.0033\% = 33.3366\%$$

Eq. 6.4

The bias in this grammar now is heavily towards placing nodes at the boundaries of the design space, with a 33.3366% probability that a node will be at either boundary, compared to a probability of 33.3266% that a node will be at any other location within the design space.

With certain, well-defined structural optimisation problems, the forms of the solutions are well-known, as they have been exhaustively covered by the literature. Taking a well-defined problem where know the structure of the solution is known, e.g. the cantilevered truss problem as defined in Figure 4.2, this domain knowledge can be used to write a grammar in such a way that it is biased towards a particular set of solutions. In the case of this 10-bar cantilevered truss, this problem would seem to favour a regular solution, i.e. structures with nodes at midpoints of the depth and quarter points of the overall span. Amendment of the percentage-based grammar from Section 6.3.4 to only account for regular node locations at quartile instances is trivial, as shown in Figure 6.17:

```
<span> ::= 18288
<depth> ::= 9144
<i_node> ::= (<percentage>, < percentage>)
<percentage> ::= 0 | 25 | 50 | 75 | 100
<node> ::= (<x>, <y>)
<x> ::= <span>/100 * i_node[0]
<y> ::= <depth>/100 * i_node[1]
```

Figure 6.17: Grammar excerpt showing quartile-based node generation

This new grammar now only places nodes in regular quartile locations associated with the accepted solution. This is actually an interesting case, as theory insists that this type of grammar will produce a more correct solution for a specific problem. However, the literature has already shown that a limited increase in variation of node locations in a ground structure approach can to better fitness results [14], and severely limiting the representation of the grammar effectively turns the approach into a ground structure method. If, as predicted by the literature presented at the start of this chapter, an unconstrained representation performs better than a constrained representation, then the hypothesis presented in this chapter can be considered to have been upheld.

6.3.6. Node Connection

The recursive capabilities of Grammatical Evolution [46, 115, 114] allow for variations in the number of nodes between individuals, resulting in a variable amount of the chromosome (Ch.A) being used. Kawamura *et al.* [73] found that representation of truss structures by combinations of triangles led exclusively to kinematically stable structures with a high degree of optimality, being able to reduce (or even eliminate entirely) the number of unnecessary members. By passing the list of nodes through a Delaunay triangulation algorithm [134], connections between nodes (edges) are defined. Delaunay triangulation operates by triangulating a set of points in a plane such that no point lies within the circum-circle of any triangle (as shown in Figure 6.18).

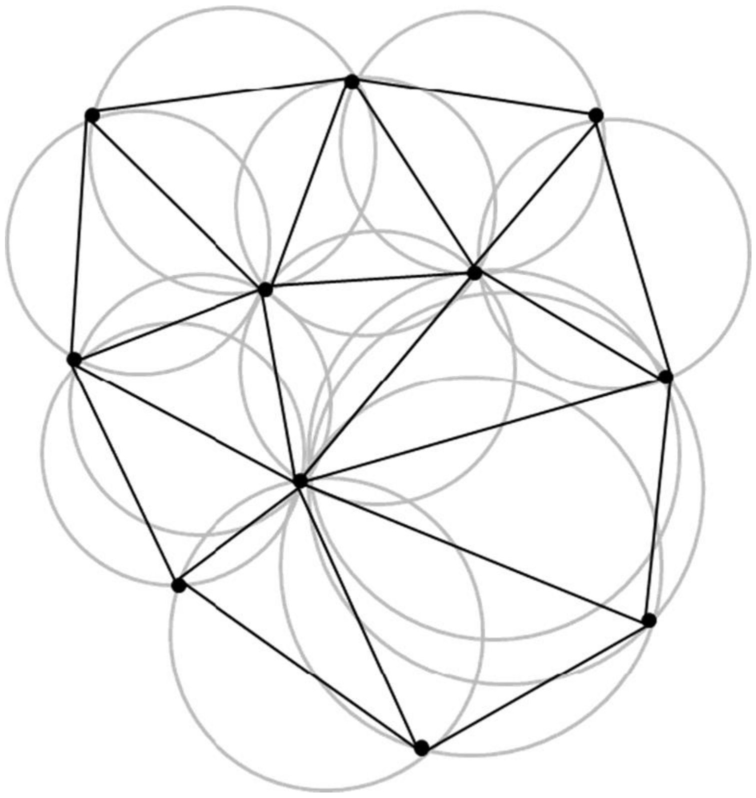


Figure 6.18: Delaunay Triangulation [135]

This triangulation method of connecting a given set of nodes is guaranteed to result in a kinematically stable, fully-triangulated structure. Materials for each of these edges are then assigned by directly mapping the genes from Chromosome B (Ch.B) to the corresponding graph objects. In this way evolution of the structural topology and shape and the sizing of individual members in tandem is possible.

The ability of DO-GE to delete unstressed members from its solutions can also be retained with this method. Though not an evolutionary feature, it nevertheless would give the method the potential to produce irregular quadrilateral shapes which might otherwise be unattainable, subject to a single caveat: the deletion of unstressed members can only be allowed to occur if a kinematic mechanism is not resultant. For a kinematically stable structure to be created, two conditions must be met:

i. Eq. 6.5 must be satisfied [9]:

$$r + m - 2n \geq 0 \quad \text{Eq. 6.5}$$

where:

r =	Number of reactions
m =	Number of members
n =	Number of nodes

ii. All nodes must have at least two edges attached

If both of these conditions are met, then an unstressed edge may be removed from the structure leaving a kinematically stable structure.

6.3.7. The GE Ripple Effect

A number of potential issues can be identified with this Delaunay-based GE method, mainly revolving around the use of two chromosomes and the subsequent effects of mutation and crossover on a dual chromosome setup with GE. Particular note should be taken on the topic of the GE ripple effect, as detailed by O'Neill *et al.* [122]. GE codons are said to have intrinsic polymorphism, with the exact meaning of any single codon being affected by the immediately preceding codons. With one chromosome, a mutation change early on in the gene sequence might change that gene's production value from a terminal to a non-terminal, which would mean the next gene in the chromosome would change its mapping, with a cascading/rippling effect on the remaining part of the gene sequence, potentially leading to significant alterations in the phenotype of the individual. This issue affects all applications of GE, not just those specified in this thesis.

The problem amplifies with two chromosomes, particularly when changes in Ch.A are taken into consideration. Since both the mapping and the amount of Ch.B that is used are entirely reliant on the phenotype derived from Ch.A, a single small change in Ch.A could completely change both the mapping for Ch.B and the portion of the chromosome that is used. If a single gene mutation occurs in Ch.A that changes a production rule from a terminal to a non-terminal (i.e. if a mutation change makes recursion possible), this would have a ripple effect which would completely change the derivation of the remaining portion of Ch.A, itself changing the mapping of Ch.B, potentially radically changing the phenotype of the individual.

These dual chromosome ripple effects are yet further with the addition of the Delaunay triangulation algorithm. Since the method described above generates node locations by selecting a percentage value of the overall limits of the design envelope, mutation of these percentage values could end up creating an entirely new coordinate value. With Delaunay triangulation, even if the total number of nodes remains constant, a change in the location of a single node could end up creating an extra edge, as shown in Figure 6.19. Once again, this would then have a chain effect on the remaining portion of Ch.B, shifting the subsequent material values to different edges.

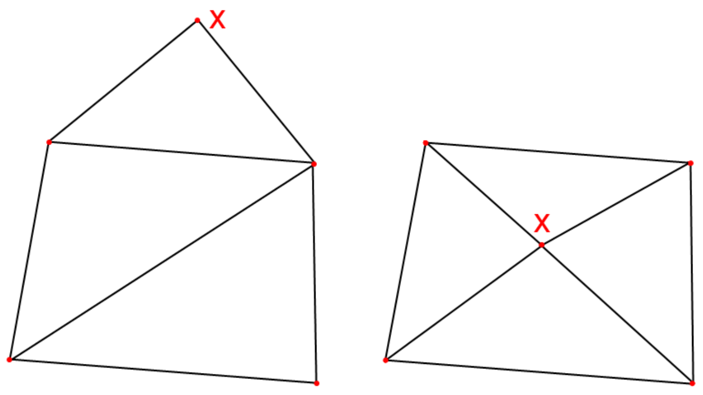


Figure 6.19: Five-node Delaunay triangulated structure, movement of node x from exterior to interior creates an additional edge

Crossover of either Chromosome of two individuals with differing amounts of edges would also lead to a GE ripple. Intrinsic polymorphism means that crossover of Ch.A would completely change the mapping of the tail end of the crossed genome, thereby changing the mapping of the latter portions of Ch.B. Similarly, crossover of Ch.B would obviously produce a different mapping process for the tails of Ch.B for both children, possibly rendering each design unfit.

O'Neill *et al.* [122] noted that there were no apparent detrimental effects from GE ripple. Indeed, the ripple effect seems to diminish convergence by performing a more varied global search, allowing for longer runs to be performed.

6.4. Truss Optimisation Examples

Continuum Topology Optimisation deals with minimisation of compliance, given a specified weight (a volume fraction of the original design space). Ground structure (beam) optimisation could be considered the opposite, as it generally deals with minimisation of self-weight, given specified deflection limit (deflection being linearly related to compliance). Since specifying a volume fraction or weight using a ground structure approach is impossible, the objective of these experiments is the minimisation of structure self-weight, given pre-specified deflection limits (unless stated otherwise). This new discrete optimisation is, hereafter, given the name SEOIGE – Structural Engineering Optimisation In Grammatical Evolution.

In this section, a number of experiments are conducted to demonstrate the capabilities of the SEOIGE technique. These experiments can be divided into three categories:

i. Proof of concept – simply supported truss example
ii. Regular truss forms and problems – benchmark tests
iii. Non-regular truss forms and problems

6.4.1. Proof of Concept: Simply Supported Truss

The first test conducted using the SEOIGE method was a "proof of concept" experiment, with the objective of evolving an arbitrary simply supported truss for minimal self-weight. The ultimate goal of this experiment was to validate the hypotheses presented at the beginning of this chapter: that it is possible to create a highly unconstrained grammatical representation (providing minimal knowledge about the form of the solution other than external inputs) which could generate a viable solution.

A generic simply supported truss design envelope was generated, with a total span of 40 m, a height of 10 m, and a single vertical force acting at centre span (as shown in Figure 6.20). Support conditions are pinned-roller. In accordance with the findings of Kicinger *et al.* [68], which stated that the most effective and efficient method of designing large structures is through the use of symmetry, a simply supported truss structure can be achieved by merely mirroring a cantilevered truss about its central axis. Thus, the algorithm only needs to evolve half of the structure.

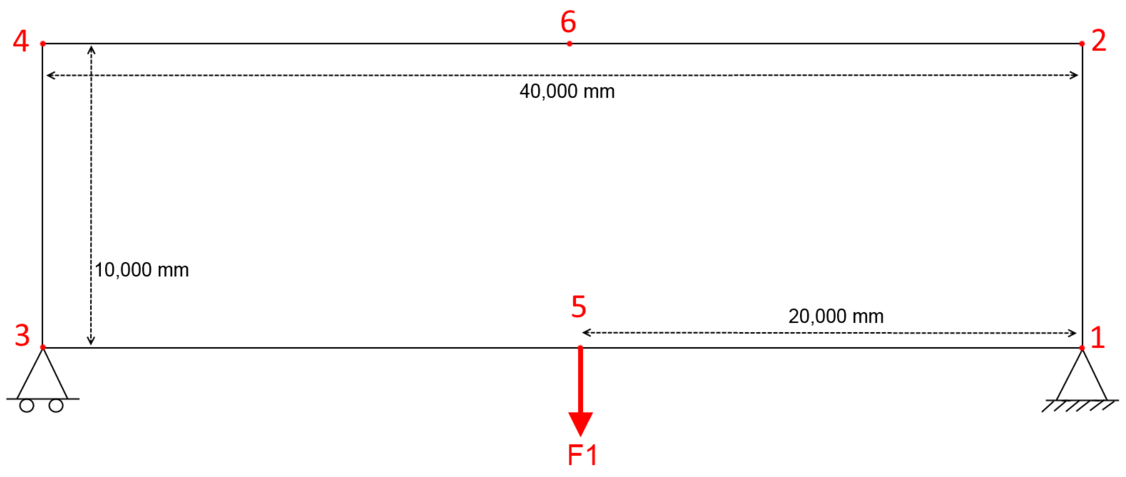


Figure 6.20: Design envelope for simply supported truss

In most cases for truss design, the truss itself will be required to conform to a specific outer shape (similar in concept to the design envelope itself). In such cases, addition of extra “design specific” nodes to the list of basic nodes is possible. These nodes will ensure that the actual envelope of the evolved solution will conform to that of the design envelope itself. These design nodes combine with essential load and support nodes to bring the total number of nodes to six essential nodes:

Table 6.2: Basic nodes for simply supported truss

Node Index	Node Location	Node Label
1	(20,000, 0)	Pinned support
2	(20,000, 10,000)	Design node
3	(-20,000, 0)	Rolling support
4	(-20,000, 10,000)	Design node
5	(0, 0)	Loaded (1,000,000 N)
6	(0, 10,000)	Design node

The validity of the hypothesis of this chapter can be summed in Figure 6.21, which shows a number of elite solutions taken from the initial generations of a single evolutionary run. A clear progression in the evolution of the fittest solutions is visible from generations 0 to 8. Initial generations begin with a random arrangement of internal nodes within the structure. As generations progress this nodal arrangement becomes less and less random as evolution drives the placement of nodes.

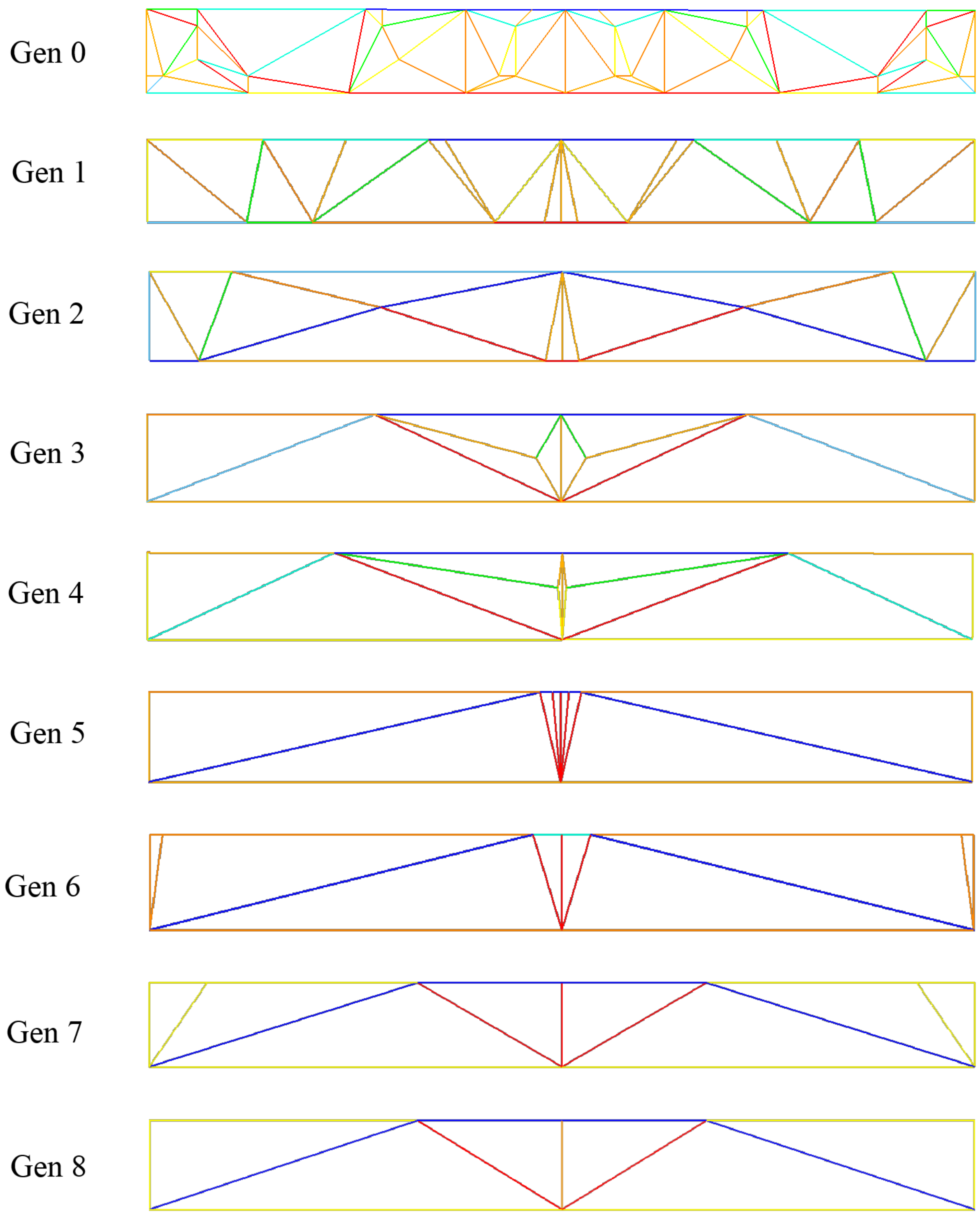


Figure 6.21: Evolution of a simply supported truss structure.

This experiment demonstrates that the SEOIGE method is indeed capable of generating viable solutions given only minimal knowledge of the problem itself. The following sets of experiments take this knowledge and compare solutions generated by SEOIGE for benchmark problems against those from the literature.

6.4.2. 10 – Bar Cantilever Truss

A cantilevered truss grammar (as shown in Appendix A.1) was created to match the dimensions and loads of the 10-bar cantilevered truss example from Section 4.3.1, as shown in Figure 6.22. The main conceptual difference between the SEOIGE method and the standard ground structure approach is that no information about the internal layout of the structure is required. A design envelope was generated by setting the maximum node dimensions to a span of 18,288 mm (corresponding to 720 inches) and a depth of 9,144 mm (corresponding to 360 inches). Any additional nodes defined by the grammar are generated via percentages of the span and the depth for the x and y coordinates respectively.

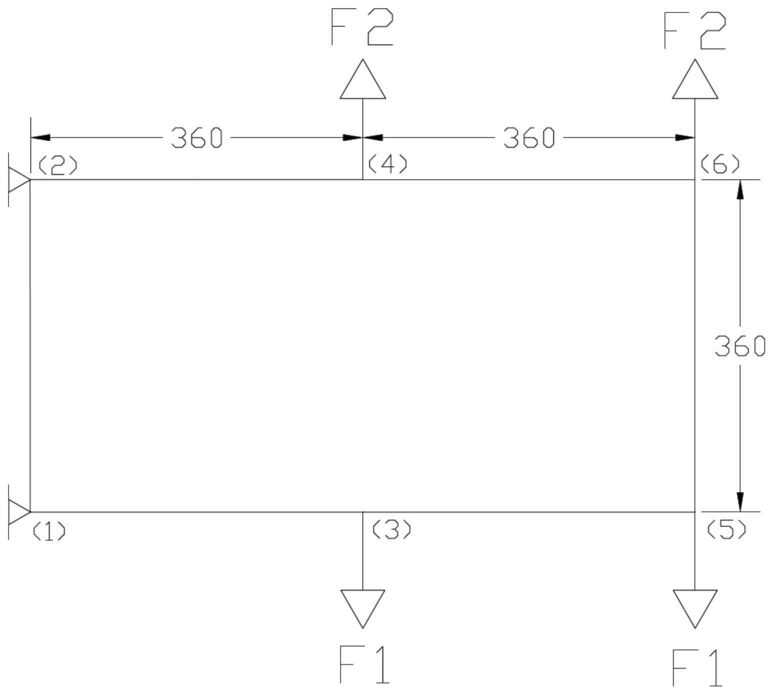


Figure 6.22: Design envelope for 10-bar cantilevered truss example, dimensions shown in inches

As with the problem defined in Section 4.3.1, two load cases were tested:

1. F1 = 444,800 N (100 kips)
   F2 = 0
2. F1 = 667,200 N (150 kips)
   F2 = 222,400 N (50 kips)

Load Case 1

For load case 1, only four essential nodes were defined (Table 6.3):

Table 6.3: Essential nodes for 10-bar cantilevered truss problem, load case 1

Node Index	Node Location	Node Label
1	(0,0)	Pinned support
2	(0, 9144)	Pinned support
3	(9144, 0)	Loaded (444,800 N)
5	(18288,0)	Loaded (444,800 N)

No other information on the design of the desired solution was detailed in the grammar. Options for either the adding of a single node or the addition of multiple nodes were given equal bias of 50% each. Node locations followed the percentage-based option as defined in Section 6.3.4, and unfit individuals were left unrepaired (i.e. no sizing optimisation was used to repair unfit individuals). A total of 100 runs were completed, each with a population of 1000 individuals over 100 generations. Mutation probability was set at 1%, and crossover probability at 75%. Both elite size and tournament size were set at 1% of the overall population size.

In terms of benchmark solutions against which to compare, Deb and Gulati [14] and Hajela & Lee [69] tackled the same problem as Luh and Lin [77], Li *et al.* [64], and Kaveh and Talatahari [65], but included a Boolean topological value in their genetic representation, allowing them to select the presence or absence of individual members. Their evolved optimal topologies (matching the globally optimum topology shown in Figure 4.4, needing only 6 bars as opposed to the original problem description of 10) resulted in greatly improved results over those portrayed in Table 4.2. As with the examples described in Section 4.2, two material sets were used for these experiments: steel Circular Hollow Sections (CHS) taken from the Tata blue book [118], and Aluminium Circular Solid Sections (CSS) as used in the literature.

Progressive stages of evolution for this problem are shown in Figure 6.23. The self-weight and load path are shown for best individuals sampled from the population at regular intervals.

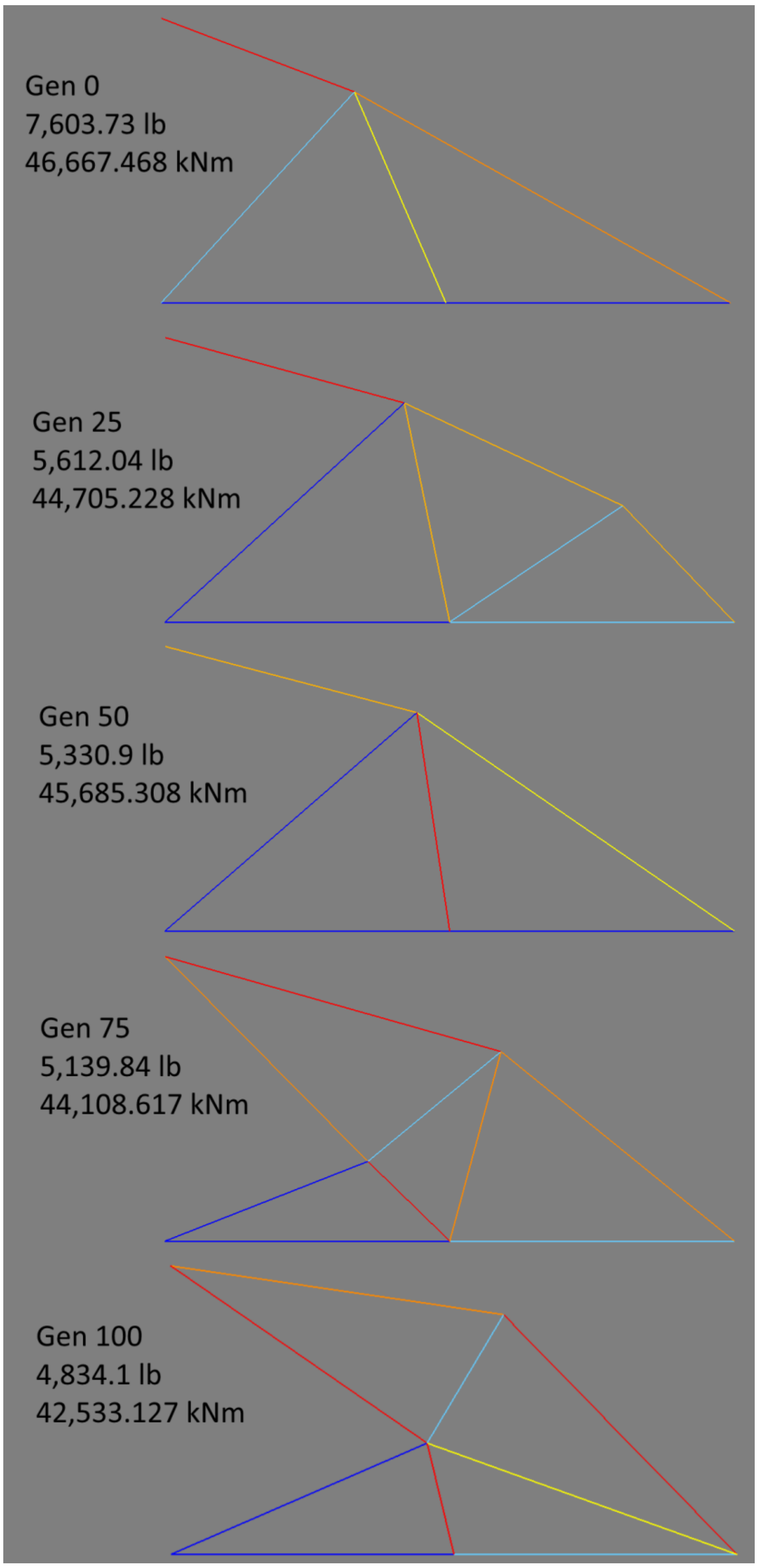


Figure 6.23: Progressive evolution of SEOIGE solution for load case 1 of the 10-bar truss problem

The optimum topological solution as derived by SEOIGE is shown in Figure 6.24. While there are still 6 nodes in the structure, there are now only 9 bars, with a substantially different topological and overall shape layout. The longest members in the structure (members 4, 3, and 9) are now all in tension, thus reducing the overall effect that lateral torsional buckling has on the structure. For this solution, SEOIGE evolved two new nodes, in addition to the four essential nodes listed in Table 6.3, as shown in Table 6.4:

Table 6.4: Non-essential nodes defined by SEOIGE for 10-bar truss, load case 1, CHS materials

Node "X" Percentage	Node "Y" Percentage	Actual Node Coordinate
58.77	83.11	(10749, 7600)
45.26	38.59	(8278, 3529)

New cross sectional areas for all members are listed in Table 6.5. Since the topological layout has changed dramatically, these cross-sectional areas cannot be compared with previous attempts (as in Table 4.2).

Table 6.5: Cross sectional areas for members shown in Figure 6.24

Element	CHS Area (mm^2)	CSS Area (mm^2)
1	4210	11871
2	4700	12968
3	4210	12129
4	4700	14839
5	2140	2710
6	153	N/A
7	3710	9290
8	4700	9484
9	3310	14194

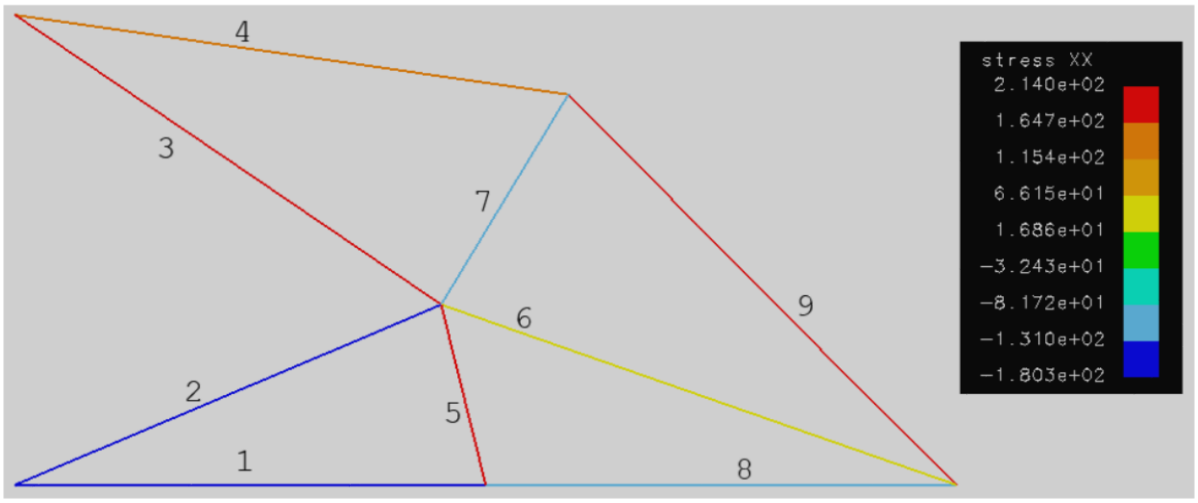


Figure 6.24: 10-bar cantilevered truss, load case 1 - optimum SEOIGE solution, CHS materials

When compared against previous results from the literature, Table 6.6 shows that SEOIGE is capable of producing results that are significantly lower than previously possible.

Table 6.6: 10-bar cantilevered truss: Evolved minimum truss weights for load case 1

	Hajela & Lee (1995)	**Deb & Gulati (2001)**	**DO-GE**		**SEOIGE**	
			Using Tata CHS steel	Aluminium solid sections	Using Tata CHS steel	Aluminium solid sections
Weight (lb)	4942.7	4912.85	5102.05	5056.88	4834.1	4888.84

While the best achieved result of DO-GE using CHS steel is 5102.05 lb, 189.2 lb heavier than the best achieved result of Deb and Gulati, SEOIGE was able to reduce this to 4834.1 lb, 78. 75 lb *lighter* than the best achieved solution of Deb and Gulati. These results can be yet further ratified by the fact that SEOIGE uses readily available CHS Steel [118] rather than the idealised solid aluminium sections of the literature.

An interesting case can be seen when the lighter Aluminium Solid Sections (as described in Table 4.1) are used. SEOIGE derives the same topological layout as shown in Figure 6.24, but member 6 was evolved with very low stresses in it, allowing for SEOIGE to delete it to improve the overall solution, as shown in Figure 6.25. Cross sectional areas are described in Table 6.5. What is interesting to note is that similarities can be seen between this solution and that of the literature [14], shown in Figure 4.4. SEOIGE's solution contains the same number of bars and the same topological layout. Although the shape itself has been distorted somewhat, the solution still improves upon those from the literature.

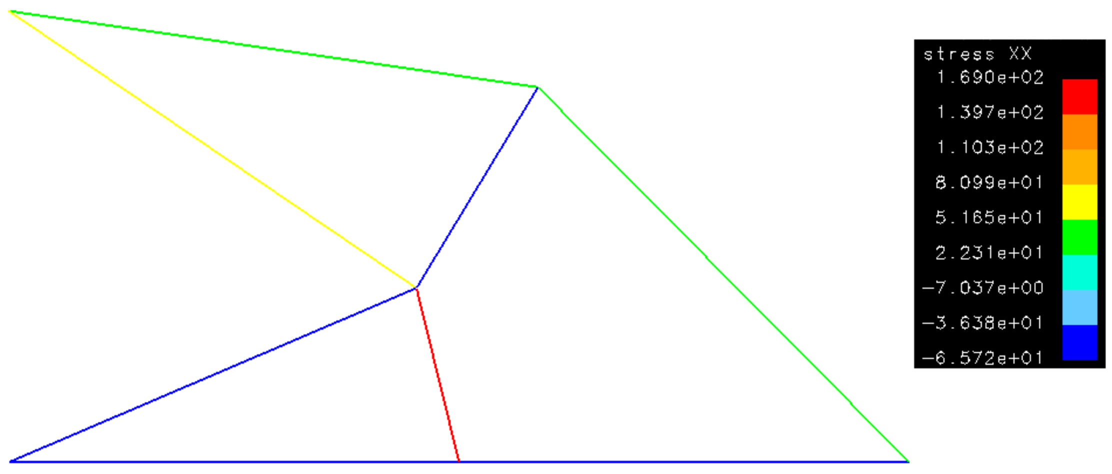


Figure 6.25: 10-bar cantilevered truss, load case 1 - optimum SEOIGE solution, CSS materials

Although it was able to find a directly comparable solution which was better than those of the literature (i.e. using the same material properties and improving on Deb & Gulati's solution by 24.01 lb), SEOIGE was unable to improve on the best solution found using the CHS steel.

Load Case 2

Load case 2 of the 10-bar cantilevered truss was not attempted by either Hajela & Lee or Deb & Gulati, and no examples could be found in the literature of the application of a topological bit-style approach to this particular problem. As such, the only comparisons that can be made are against the same literature as described in Section 4.3.1 – those of Li *et al.* and Kaveh and Talatahari [64, 65].

The addition of two more loads to the structure requires the addition of two more essential nodes, for a total of six essential nodes (as described in Table 6.7), the same number as with the original 10-bar truss model:

Table 6.7: Essential nodes for 10-bar cantilevered truss problem, load case 2

Node Index	Node Location	Node Label
1	(0, 0)	Pinned support
2	(0, 9144)	Pinned support
3	(9144, 0)	Loaded (667,200 kN)
4	(9144, 9144)	Loaded (222,400 kN)
5	(18288, 0)	Loaded (667,200 kN)
6	(18288, 9144)	Loaded (222,400 kN)

All other variables and settings remained the same as the previous load case.

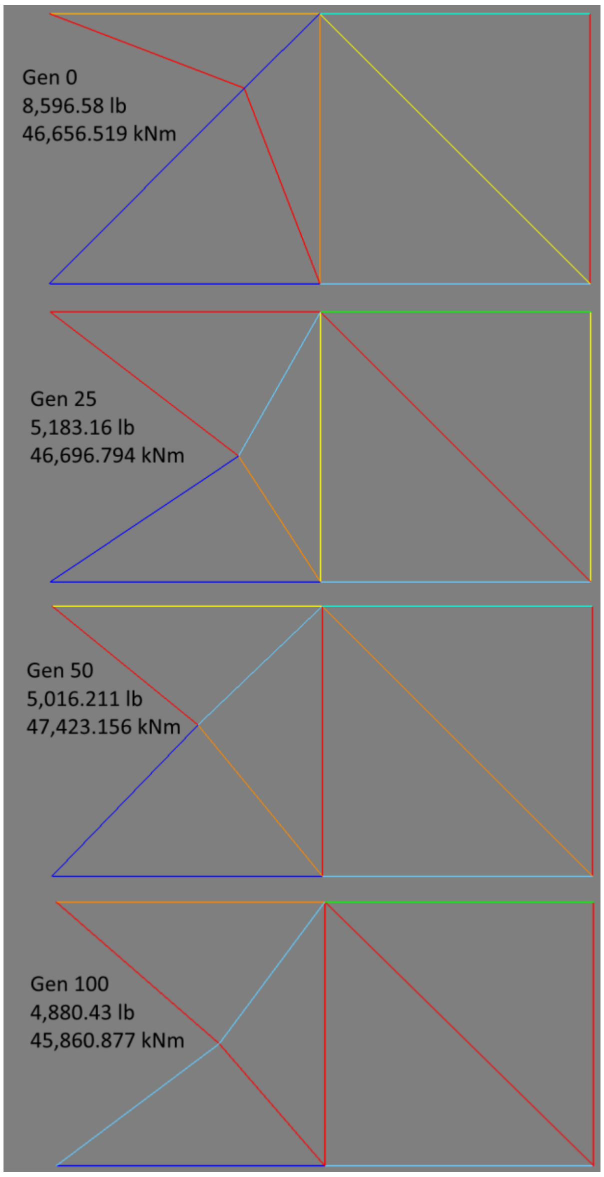


Figure 6.26: Progressive evolution of SEOIGE solution for load case 2 of the 10-bar truss problem

The progressive evolution of the best solutions from regular generations of a typical evolutionary run can be seen in Figure 6.26. What is interesting to note is that in this case the load path continually increases until the optimal solution is evolved. However, the self-weight of the solutions continuously decreases.

The evolved optimised topology for both material cases is shown in Figure 6.27:

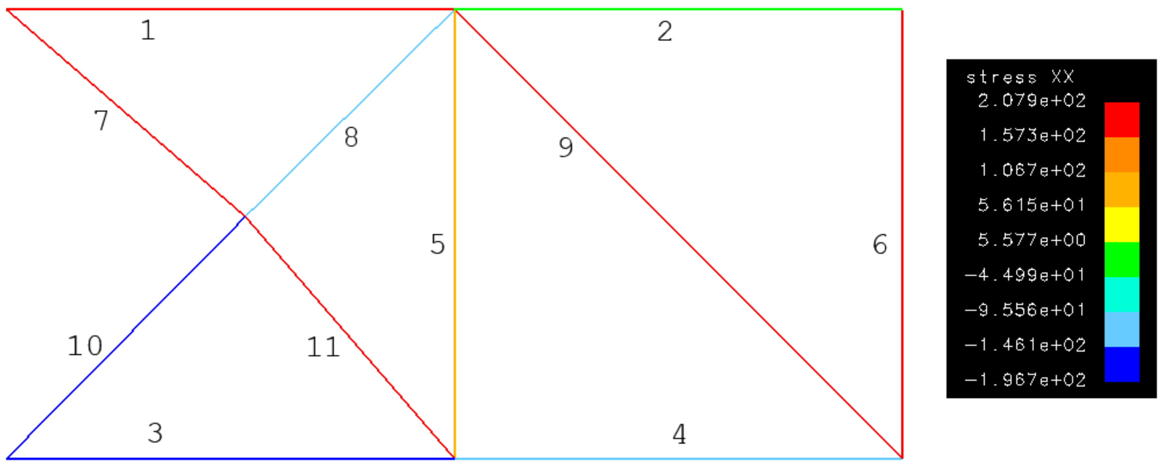


Figure 6.27: 10-bar cantilevered truss, load case 2 - optimum SEOIGE solution, CHS Materials

As with load case 1, it can be seen that the member with the highest risk of buckling has been broken into smaller segments to reduce the risk and to better dissipate the stresses. One optional node was added to this structure in addition to the six essential nodes listed in Table 6.7, as shown below in Table 6.8:

Table 6.8: Non-essential nodes defined by SEOIGE for 10-bar truss, load case 2

	Node "X" Percentage	Node "Y" Percentage	Actual Node Coordinate
CHS Materials	26.31	54.16	(4812, 4952)
CSS Materials	31.15	51.79	(5697, 4736)

As with load case 1, a different topological layout makes comparisons between evolved member cross-sectional areas of previous approaches difficult. Similarities, however, between

Figure 4.2 and Figure 6.27 show many common members, meaning direct comparisons are not impossible in this case.

The use of CSS materials evolved a very similar solution, as shown in Figure 6.28. Again, only one essential node was added to the solution, described in Table 6.8 above.

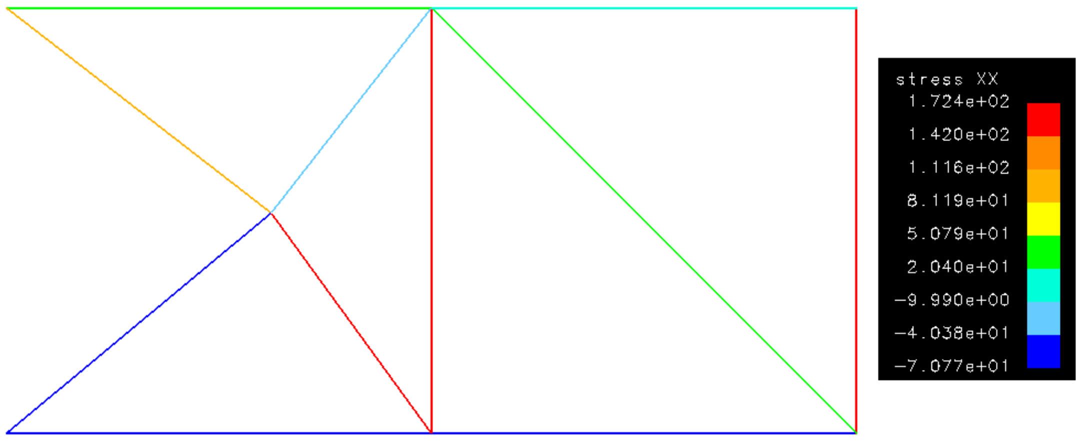


Figure 6.28: 10-bar cantilevered truss, load case 2 - optimum SEOIGE solution, CSS Materials

Cross-sectional areas for all members are given in Table 6.9 below. Member 5 for both solutions is of the lowest available cross-sectional area, and examination of the analysis results shows that there are actually zero stresses in the member itself, meaning that potential improvements to the overall structure can be made without that member.

Table 6.9: Cross-sectional areas for 10-bar truss, load case 2, SEOIGE solution

Element	CHS Area (mm^2)	CSS Area (mm^2)
1	4670	15161
2	153	64.52
3	5010	13032
4	3760	9161
5	153	65
6	1070	1290
7	4670	9097
8	2570	8581
9	3710	12774
10	3760	13032
11	4030	4774

However, while member 5 contains zero stresses, its deletion from the solution would create a kinematic mechanism whereby member 6 is only connected to one node and is free to rotate, as shown in Figure 6.29.

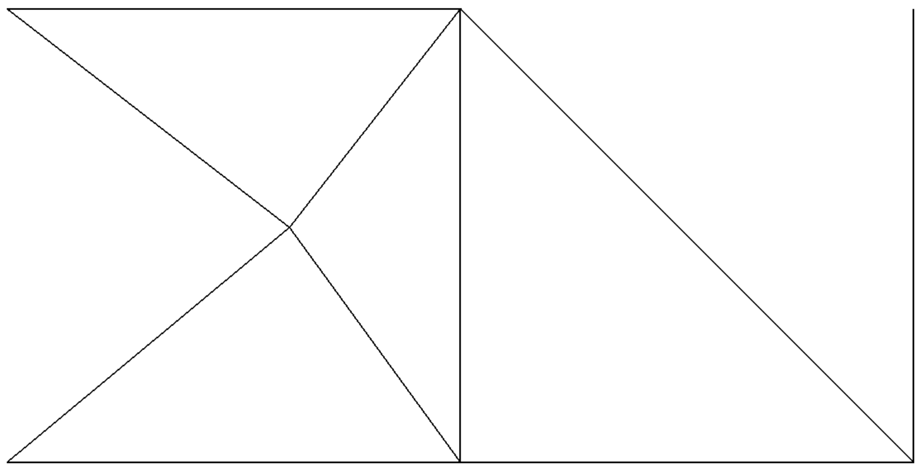


Figure 6.29: Kinematically unstable structure obtained with deletion of member 5

For this reason member 5 must be kept in the solution.

The optimisation results for load case 2 using both Tata CHS steel and the Aluminium Solid Sections are presented in Table 6.10:

Table 6.10: 10-bar cantilevered truss: Evolved minimum truss weights for load case 2

			DO-GE		**SEOIGE**	
	Li *et al.* (2007)	**Kaveh & Talatahari (2009)**	Using Tata CHS steel	Aluminium solid sections	Using Tata CHS steel	Aluminium solid sections
Weight (lb)	4677.3	4675.8	5016.3	4612.8	4880.43	4624.35

The best achieved result for DO-GE using commercially available CHS steel materials and material properties was 5016.3 lb, some 340.5 lb off the best achieved minimum weight of Kaveh and Talatahari. SEOIGE was able to improve on the achieved result of DO-GE by 135.87 lb, bringing it down to 4880.43 lb when using CHS steel, although still 204.63 lb off the best achieved result of Kaveh and Talatahari. Comparisons of DO-GE and SEOIGE using the idealised Aluminium Solid Sections show that although SEOIGE was able to achieve a better result than that of Kaveh and Talatahari (besting their solution by 51.45 lb), it was still 11.55 lb off the best achieved result by DO-GE. The reasoning for this is that the number and location of the essential nodes for Load Case 2 severely limits the scope of this problem for the SEOIGE method. Whereas SEOIGE performs well when there is little information given about the form of the solution (i.e. in Load Case 1), in Load Case 2 there is very little room for SEOIGE to evolve its own solution as the anticipated form of the solution is pre-set by the location of the essential nodes. This discussion is expanded on in Section 6.5.

6.4.3. 17 – Bar Cantilever Truss

Li *et al.* [64], Lee and Geem [76], Khot and Berke [128], Adeli and Kumar [129], and Fenton *et al.* [117] proposed solutions to a planar 17-bar cantilevered truss problem, as described in Section 4.3.3. A new design envelope for a cantilevered truss grammar was created to match the dimensions and loads of this problem, shown in Figure 6.30.

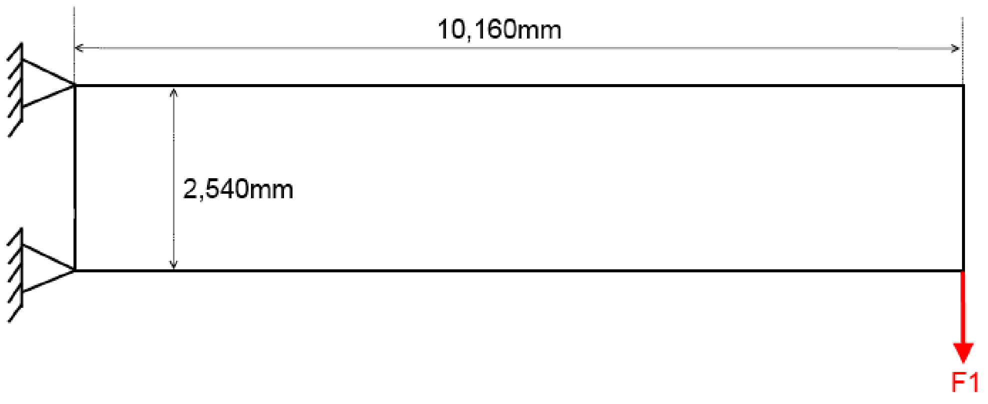


Figure 6.30: Design envelope for 17-bar cantilevered truss

The design envelope for this problem measures 400 inches by 100 inches (10160 mm by 2540 mm), with a vertical load of 100 kips (444,800 N) acting at the far end of the structure. Only three essential nodes need to be defined in this instance:

Table 6.11: Essential nodes for 17-bar cantilevered truss problem

Node Index	Node Location	Node Label
1	(0, 0)	Pinned support
2	(2540, 0)	Pinned support
3	(0, 10160)	Loaded (444,800 N)

In this instance both the shape of the design envelope and knowledge of prior successful solutions provide clues to the potential shape of the optimum solution. Thus, the grammar can be biased accordingly such that SEOIGE has a greater chance of generating a fitter solution. As described in Section 6.3.5, SEOIGE's percentage-based node generation makes it possible to easily add bias towards specific percentages to create more regularised structures. In the case of the 17-bar truss problem, previously successful solutions (as shown in Figure 4.12 and Figure 4.14) indicate that nodes placed at quarter points along the x-axis (i.e. at 25%, 50%, and 75% of the total span) will generate an optimum solution. These options were added to the grammar such that the production rule <node> has four options, as shown in Figure 6.31:

```
<node> ::= 25 | 50 | 75 | float("<n><n>.<n><n>")
<n> ::= 0|1|2|3|4|5|6|7|8|9
```

Figure 6.31: Node generation production rule for SEOIGE grammar, 17-bar truss case

Notably, this grammar, though biased towards placing nodes in regularized locations, still retains the capability of placing nodes at any location within the design envelope. In this manner SEOIGE can determine the optimum topological layout, with hints at what that optimum layout is suspected to be.

As with the previous examples, idealised materials (solid steel sections as described in Table 4.1) were used by the literature in solving this problem. The experiments were therefore run using Tata CHS steel and again with the idealised solid steel sections in order to benchmark against the performance of both the results from the literature and previous results from DO-GE.

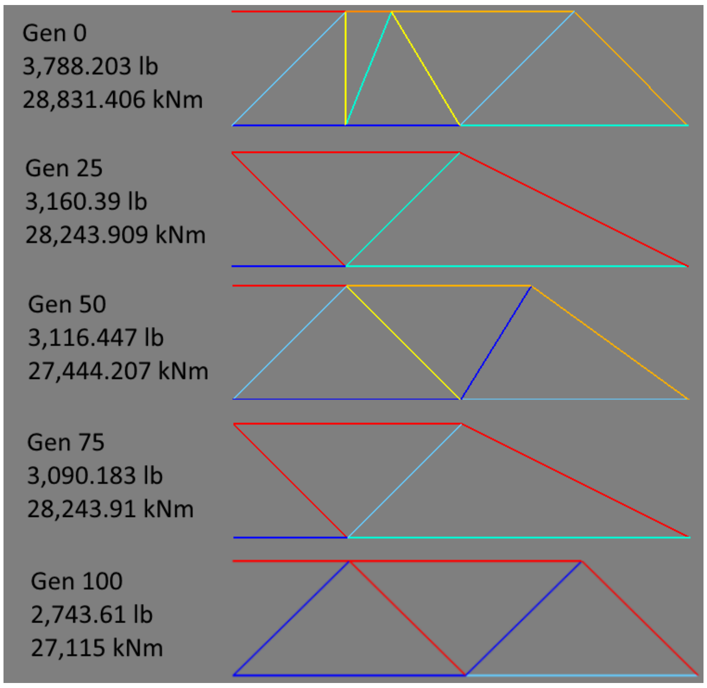


Figure 6.32: Progressive evolution of SEOIGE solution for 17-bar truss

Progressive evolution of the SEOIGE solution to the 17-bar truss problem can be seen in Figure 6.32. As with Figure 6.23 and Figure 6.26, Figure 6.32 shows that while the overall objective of the structural self-weight is continually minimised, the load path of the best solution sometimes increases incrementally. The final solution, however, is always the most superior result in terms of both self-weight and load path.

The optimum evolved topology for this problem can be seen in Figure 6.33. What is interesting to note is that this solution is visually identical to that derived by DO-GE, as shown in Figure 4.14 earlier. Even though the grammar retained the capability of placing a node at any location, the most optimum node locations as determined by the grammar were, as predicted, at quartile points.

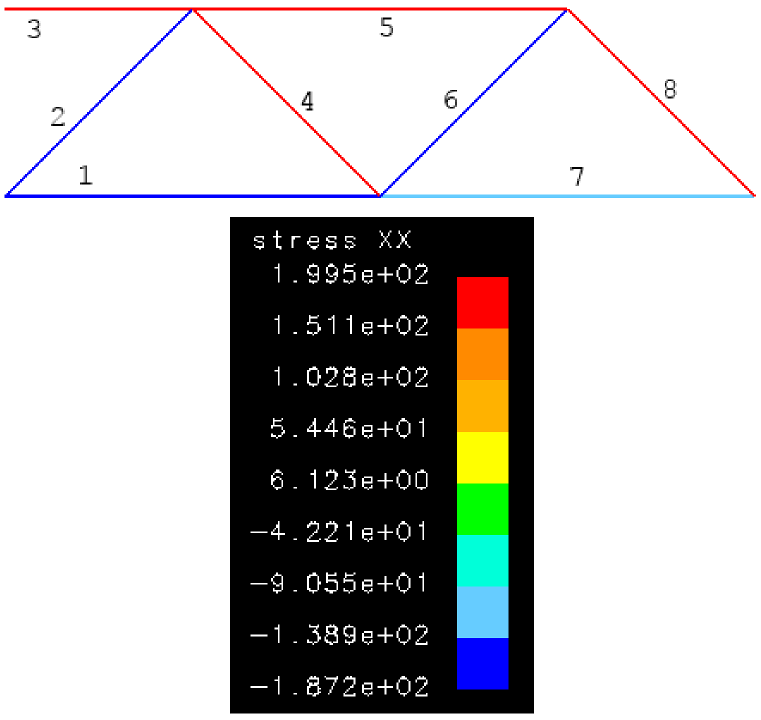


Figure 6.33: 17-bar cantilevered truss – optimum SEOIGE solution, CHS Materials

Evolved optional nodes are shown in Table 6.12.

Table 6.12: Non-essential nodes defined by SEOIGE for 17-bar truss

Node "X" Percentage	Node "Y" Percentage	Actual Node Coordinate
25.00	99.99	(2540, 2540)
50.00	00.00	(5081, 0)
75.00	99.99	(7620, 2540)

Cross sectional areas for all members using both realistic CHS steel and the idealised CSS steel sections are listed in Table 6.13:

Table 6.13: Cross sectional areas for members shown in Figure 6.33

Element	CHS Area (mm^2)	CSS Area (mm^2)
1	7920	8065
2	3360	3226
3	8920	10452
4	3760	3419
5	5010	5226
6	3360	3742
7	4030	2387
8	3360	4000

The results of these experiments are shown in Table 6.14.

Table 6.14: 17-bar cantilevered truss: Evolved minimum truss weights

	Khot	**Adeli**	**Li *et al.***	**DO-GE**		**SEOIGE**	
				Using Tata CHS steel	Steel solid sections	Using Tata CHS steel	Steel solid sections
Weight (lb)	2581.9	2594.4	2581.9	2642.1	2595.4	2743.61	2581.9

It must be noted from Table 6.14, however, that in this case SEOIGE was unable to match the exact minimum weight derived by DO-GE or of the literature. The interpretation of this is explained in the discussion in Section 6.5.

6.4.4. Non-Regular Truss Forms

Since the SEOIGE method has been validated and benchmarked against standard forms and tests from the literature in the above experiments, examination of its capacity to optimise non-standard truss envelopes would be useful. The constraints on structural design will often be driven by physical dimensions and limitations. Architects desire the freedom to design form over function, but are often limited by the nature and capacity of regular structural support mechanisms. Conversely, engineers are routinely pushed to design solutions for the ever more complex problems that arise from pushing the boundaries of shape and form. A design approach that could provide structural support solutions for arbitrary shapes and forms would therefore be of great value to both engineer and architect. This is a particular area where the node-based SEOIGE method can excel.

For design envelopes with non-regular shapes, limits can be set on either the x or y-values by using an equation that defines the boundary conditions, in the form of:

$$y = f(x) \tag{Eq. 6.6}$$

The design envelope for any shape which can be defined by an equation can then be thus described.

Figure 6.34 shows a design envelope for a long span curved truss, with a span of 100 m and three point loads of 1,000,000 N, at both quarter points and at midpoint.

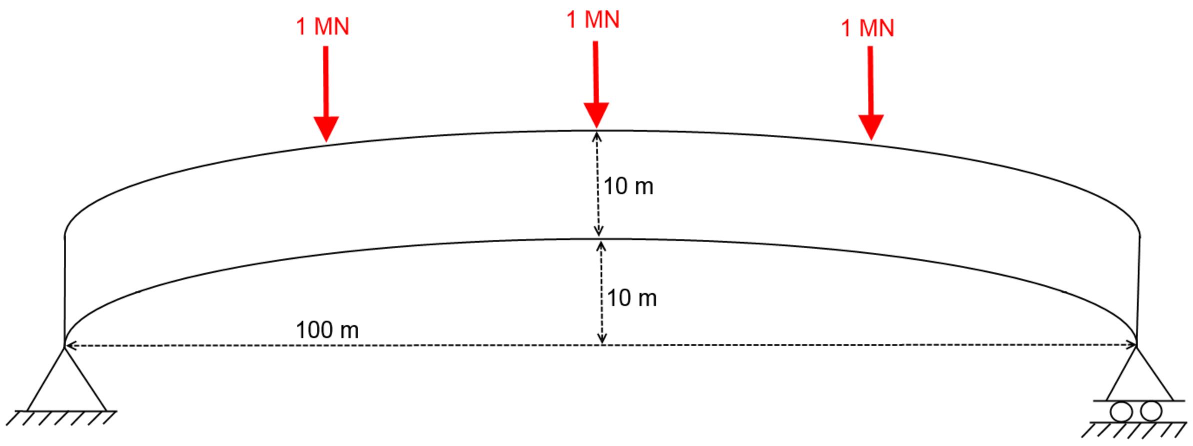


Figure 6.34: Design envelope for large span curved truss with high loading

The design envelope only limits the placement of nodes, and as such elements connecting nodes within the design envelope may stray outside of it, as shown in the solution in Figure 6.35. Removal of this capability from the program, however, is entirely possible by further defining the problem.

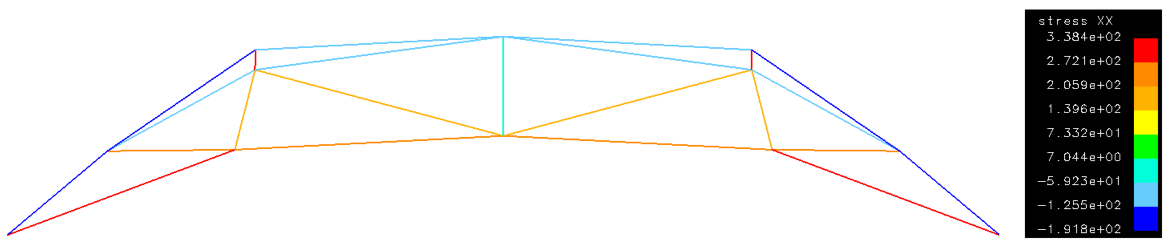


Figure 6.35: 100 meter curved truss, no limit to length of members

Inclusion of a maximum length limit for members is also possible in these structures. While this is not necessary (long members in tension pose no risk, long members in compression are evolved such that they are not at risk of buckling), the results are interesting. The 100 meter curved truss problem from Figure 6.35 was once again attempted, but a maximum limit for all member lengths was set at 20 meters. The resultant structure, shown in Figure 6.36, differs dramatically from the original solution.

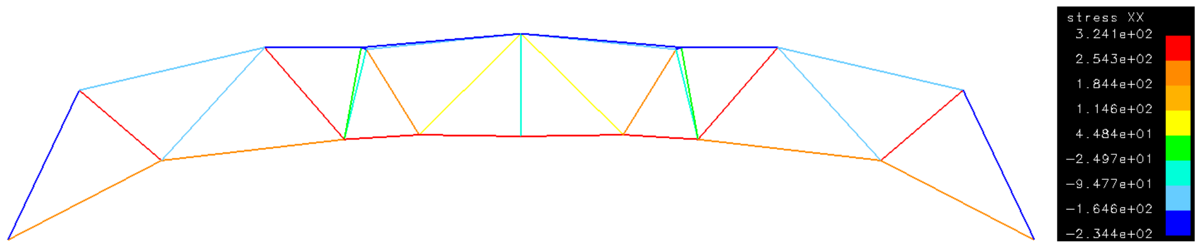


Figure 6.36: 100 meter curved truss, maximum member length limited to 20 meters

Other forms were also evolved, such as a circular arched structure, shown in Figure 6.37.

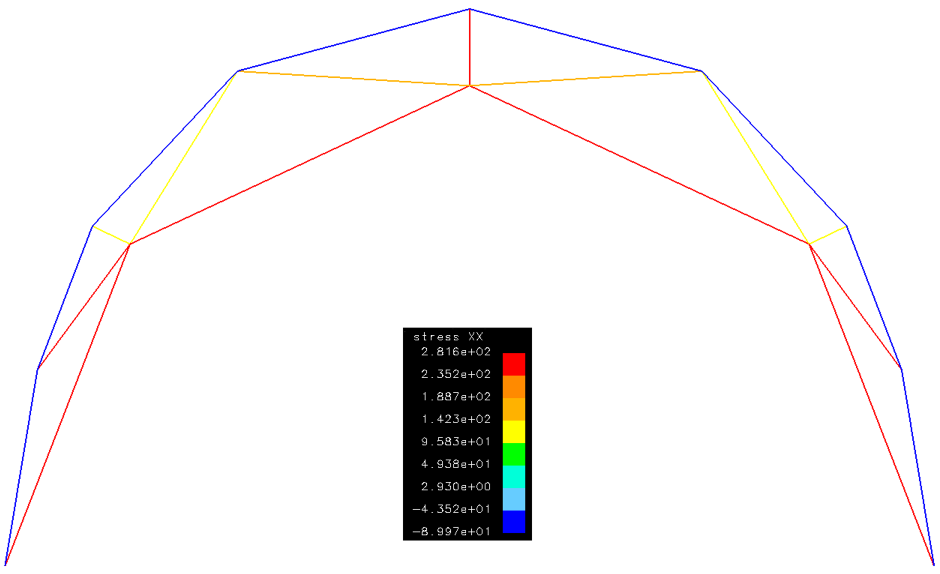


Figure 6.37: Truss shape derived from circular arch design envelope

These experiments demonstrate the capacity of the SEOIGE method to generate viable solutions for problems where there is little to no information known about the form of the solution, other than the external acting forces.

6.5. Discussion

Comparison of results from the various experiments performed in this chapter yield interesting conclusions. The benchmark experiments in Sections 6.4.2 and 6.4.3 show that in most cases, the SEOIGE method was able to improve on the results from ground structure approaches in the literature. SEOIGE was also able to improve on the results evolved by DO-GE in Chapter 4 for at least one of the given material cases in each of the benchmark tests attempted. For load case 1 of the 10-bar cantilevered truss benchmark problem, SEOIGE produces a significant improvement over both DO-GE and the results found in the literature. However, the method introduced in this chapter notably was not able to improve on traditional ground structure approaches for a number of cases. For load case 2 and the 17-bar case, SEOIGE's results aren't quite as impressive, only matching or marginally improving on results from the literature when using comparative materials (and failing to improve at all using commercially available materials). The reasoning for this can be thus explained.

SEOIGE can be seen to perform well where there is both a large design envelope area *and* where little information is known about the optimal solution. In the instance of load case 1 for the 10-bar truss, there are four essential nodes and a design envelope where one dimension is twice the length of the other, creating a broad area over which new nodes can be placed (and therefore a broad area over which new structures can be evolved). Since the SEOIGE method operates by addition and subsequent connection of newly defined nodes above and beyond predefined essential nodes, there is plenty of scope for adding new nodes and structure. This is promoted as there is little information given about the structure of potential solutions. An example of an unintuitive yet highly successful solution evolved using this method can be seen in Figure 6.24. This solution is very different to the previously accepted solutions to the problem.

However, the SEOIGE method does not perform quite as well when the problem is well defined (i.e. in cases with known solutions and little flexibility in the design space). Such is the case with both load case 2 for the 10-bar truss problem and the 17-bar cantilevered truss problem. Since SEOIGE operates by adding new nodes to form a structure, it performs best when it has control over not only the topological layout but more importantly the overall shape of the structure. With load case 2 for the 10-bar truss, the shape of the structure is pre-determined by the number of essential nodes on the boundary of the design envelope, and SEOIGE therefore does not have the freedom it needs to define its own solution. Additional nodes can still be added to the interior of the structure, but the optimal solution contains a minimal amount of edges, and therefore a minimal number of nodes. As such, SEOIGE cannot perform as well in these conditions.

The gulf between the performance of regular ground structure methods from the literature and the SEOIGE method can further be explained by the fact that SEOIGE's Delaunay triangulation approach to node connections is algorithmic in nature and lacks an appreciation for specific topological efficiency. The Delaunay triangulation method cannot specify which connections are required. With load case 2 for the 10-bar truss, there are six essential nodes defined (as shown in Table 6.7). Since the optimal solution for this problem contains a minimal number of edges, SEOIGE does not have the freedom it needs to evolve an optimal shape by adding new nodes. The only logical course of action in this case is to optimise the topology of the existing structure (i.e. vary the connectivity between fixed node locations to obtain a fitter solution). This means that this particular problem is well suited to a ground structure approach. A method which varies the number and location of nodes, and which automatically connects nodes in an algorithmic fashion, lacks the fine topological control that this particular problem requires and is consequently less effective.

A similar situation, although less dramatic, is seen with the 17-bar truss problem. In this instance however, the extreme dimensions of the design envelope, with one dimension being four times greater than the other, coupled with the fact that the most appropriate solutions are known (structures with regular node locations and elements) means that again SEOIGE's options are limited. Although SEOIGE is always given the option of deciding its own node locations in the regular fashion described in Section 6.3 (and, thus, retaining the full representation space capabilities), the inclusion of the regular quartile node locations at 25%, 50%, 75%, and 100% of the total span allows the program to sample a wide range of potential solutions whilst hinting at the potential correct solution. The fact that SEOIGE evolves optimal solutions using only these suggested quartiles and not using any nodes generated using the regular percentage-based method means that the problem can be considered well-defined in that the solution of the structure is known.

6.6. Conclusion

Traditional ground-structure-based discrete topology optimisation methods have been proven capable of deriving optimal solutions to benchmark problems [77]. However, the literature has suggested that the small representation spaces of ground structure methods limit their efficiency [14, 7, 13]. A hypothesis was posed at the beginning of this chapter that a method which could evolve the number and placement of nodes rather than the connectivity between fixed nodes would provide a vastly increased representation space over traditional ground structure methods. This would theoretically allow for a wider sample base, consequently increasing the potential for superior solutions.

The literature shows that the most optimal solutions for discrete structures can only be found by optimising the topology, shape, and member sizing of a structure simultaneously [77]. In choosing the location of its nodes, the method presented in this chapter gains fine control over both the shape and topology optimisation of the overall structure. This ability, combined with the use of two chromosomes (one for evolving topology and shape of the structure, and the other for evolving the member sizing) means that SEOIGE can truly optimise the structural shape, structural topology, and member sizing of a structure in tandem. Using this method, SEOIGE is able to out-perform various methods from the literature, producing superior solutions for an array of benchmark problems. As such, all research objectives can be considered to be successfully satisfied.

However, while the SEOIGE method can produce optimal solutions for problems where the solution is not known, it works best when both the structure of the solution and the design envelope (and more importantly the shape of the potential solution in the form of the number and locations of essential nodes) are ill-defined. The less essential nodes that exist (i.e. the looser the design brief), the better the solution SEOIGE can evolve. In cases where the structural shape is already set and connectivity between pre-existing nodes is paramount to finding the global optimum, SEOIGE has difficulty in improving upon existing solutions due to its lack of fine topological control over member connections.

6.7. Summary

A need was identified in the literature review (Chapter 2) to generate an unconstrained representation for discrete truss structures. The literature review also found that variation in node placement can lead to improved fitness results over traditional ground structure methods with fixed node locations. Bearing this in mind, this chapter introduced the idea of generating discrete structures through placement and variation of the nodes themselves rather than variation in connectivity of pre-existing nodes. This method, when combined with the novel use of a Delaunay triangulation algorithm to connect evolved nodes, is capable of generating repeatable fully triangulated, kinematically stable structures. The only input required for this method to function is the external actions acting on the structure (i.e. forces and reactions). No information about the interior structure of the solution is required to evolve viable solutions. The use of two chromosomes (a concept introduced in Chapter 4) allowed for the simultaneous optimisation of structural shape and topology, and of member sizing. These structures were then evolved to find minimum weight arrangements for benchmark problems, and in cases where the problem was ill-defined and the structure of the solution was not previously known the SEOIGE method was able to improve on the results from the literature.

Chapter 7. Conclusion

This chapter presents a summary of the work explored in this thesis, and relates this work back to the original research questions posed in Section 1.2.1. This chapter concludes with recommendations for future work in this field, discussed in Section 7.2.

7.1. Thesis Summary

The primary aim of this thesis was to investigate whether, given only boundary conditions such as loading and reactions, it is possible to evolve viable solutions using a discrete optimisation method. Based on a review of the literature, a number of research objectives were posed in order to ascertain whether this was possible. These aspects incorporated both structural engineering and evolutionary computation problems. As each objective is stated in this summary, the conclusions derived from that question are discussed.

7.1.1. Both sizing and topology optimisation conducted in tandem

The literature review presented in Chapter 2 found that simultaneous optimization of member sizing, structure shape, and structure topology theoretically produces the best optimization results. However, formulation is difficult with a constrained ground structure representation, and even more difficult to achieve effectively. In Chapter 4 a new technique for evolving multiple datasets simultaneously was introduced. This was enabled by the use of two separate chromosomes; one for evolving the overall shape and topology of the structure, and one for evolving the sizes of individual members in the structure. This method was compared, with identical materials and constraints, against various results from the literature, and it was found to be able to produce comparable results. This demonstrated that simultaneous optimisation of structural shape, topology, and member sizing is possible. Comparison was also made between this dual chromosome optimisation method and an optimisation method which used a deterministic approach to directly calculate the optimum member sizings for any given topology. The deterministic approach resulted in a five-fold increase in computational time and a lower overall average fitness value for evolved solutions.

7.1.2. A comparison of commercially available materials and code compliant constraints against their idealised counterparts

The very same comparison with the discrete optimisation literature discovered that the majority of the literature used unrealistic assumptions about material properties and physical structure layout. The literature review showed that there is a tendency in the literature to focus on minute improvements to optimization methodologies, while fundamental engineering principles are overlooked. When real world construction materials and building code-compliant constraints were utilized in the same problems, experimental data showed that the results, while still acceptable, were not as impressive as the initial set. The widespread use of identical tensile and compressive stress limits on the material and the lack of use of design codes and standards of practice in such optimisation methods, thus, gives a false impression of both the efficiency of the algorithm and the importance of the achieved results.

7.1.3. An efficient penalty function for violated constraints in Grammatical Evolution

A review of the literature indicated that a larger representation space would result in both an improved chance of evolutionary success and a better overall quality of individuals. Traditional ground structure discrete optimization approaches are also limited in their representation capabilities and thus cannot cover the search space as effectively as possible. Chapter 5 showed that an increased representation space coupled with an increased number of unfit individuals requires careful manipulation of individuals with violated constraints in order to evolve *any* fit solution. Although experiments in Chapter 5 demonstrated that the addition of more constraints further lessens the total amount of potential fit solutions in a population, the results from Chapter 4 indicate that design constraints in the form of building standards and codes of practice must be viewed as an engineering necessity. As such the use of more constraints can generally be regarded as representing better design practice. Finally, a new method of generating a penalty function for unfit individuals was generated, and this method was compared against various methods from the literature. Although most methods performed identically for less difficult problems, this new method provided an improvement over previous methods for highly difficult problems such as those with a very large representation space.

7.1.4. Creation of an unconstrained grammar representation

Once an efficient method of handling a large representation space was established the various methods of creating a grammar capable of generating such an unconstrained representation space were discussed in Chapter 6. Since the literature review found that allowance for small variation in node placement can lead to an improved fitness result in traditional ground structure optimisation, the hypothesis was posed that complete freedom in nodal positioning and placement would allow the heuristic process to get closer to the true global optimum. Various approaches were discussed with regard to generating nodal coordinates within a pre-specified design envelope, and the most effective method was the representation of potential new node locations as a percentage of the overall dimensional limits. A novel method of connecting existing nodes using a Delaunay triangulation algorithm was implemented, and repeatable, fully triangulated, kinematically stable structures were generated using this method.

7.1.5. Evolution of viable solutions given only boundary conditions and constraints

Chapter 6 also showed that the only design components necessary to evolve a viable solution using this new method were the nodal locations (and magnitudes) of external forces acting on the structure, including loading forces and reactionary forces. Experiments showed that viable solutions can be evolved given no prior information on the interior structure of the solution. Finally, using two separate chromosomes as defined in Chapter 4, and combined with the new penalty function described in Chapter 5, experiments conducted in Chapter 6 compared this new Delaunay-based node variant method against benchmark discrete ground structure optimisation methods from the literature. These experiments showed that the new method presented in this thesis produced superior results to those of the literature in cases where the problem was ill-defined and the structure of the solution is not known *a priori*.

7.2. Contributions

A number of contributions have arisen from this work. These include publications, which are listed in the “Publications Arising” section on page xvii.

Literature Review: A review of the literature in the area of Evolutionary Computation and Structural Design optimisation was presented in Chapter 2.

Applications of GE in Structural Engineering: A version of GE has been developed (as described in Chapter 3), which allows for the evolution of structural engineering designs using engineering parameters for both design constraints and as an objective function.

Realistic material design generation: The use of commercially available construction materials in evolutionary design was explored. In Chapter 4, it was found that solutions very close to leading solutions from the literature can be achieved using commercially available elements.

Code-compliant constraints: Current building codes of practice were translated into design constraints. When combined with realistic materials, these allow for critical knowledge of section geometry and orientation to be correctly used. In Chapter 4 it was found to be therefore possible to calculate fully accurate material stress limits as a constraint for structural design.

Twin chromosome GE: A version of GE was developed which used two chromosomes, allowing genetic information of two separate data sets to be transferred between generations. These separate chromosomes were used to evolve the member sizing and the overall topology of a structure in tandem, and it was shown in Chapter 4 that it is possible to simultaneously optimise the structural topology and member sizing of a structure.

Fitness landscapes and search: Definitions of the terms fitness landscapes and search were given in Chapter 5. Experiments demonstrated the properties of these landscapes as the attempted problems become more difficult. It was shown that an increase in the difficulty of a problem will correspondingly reduce the number of potential fit solutions in the overall fitness space.

Penalty Function: A new penalty function was developed in Chapter 5, for use on individuals with failed constraints (as shown in Eq. 5.19). This penalty function sums the normalised values of all constraint violations allowing for accurate comparisons between individuals with multiple constraint violations.

Unconstrained discrete truss grammar representation: A new grammar was developed with an unconstrained discrete truss representation, which allows GE to evolve its own solutions. This was detailed in Chapter 6, where it was shown that more representational freedom in a grammar can generate fitter individuals.

Edge generation and truss connectivity using Delaunay triangulation: A Delaunay triangulation algorithm was used within the grammar to connect nodes together to form truss members. This technique, as described in Chapter 6, is novel in this application, but has ties to the field of finite element analysis and mesh generation, where the use of Delaunay triangulation is prevalent.

7.3. Limitations

This thesis focussed on the development of a simultaneous optimisation technique for structural shape, structural topology, and member sizing for discrete truss structures. As such, certain limitations were inherent in the scope of the investigations. These limitations are explained below.

Although it is entirely possible for building codes of practice of any type to be used in conjunction with the work presented here, all experiments were performed using the British Standards due to the author's familiarity with the system. Use of Eurocodes, International Building Codes (IBC), etc. is entirely possible, but requires the user to apply the relevant constraints and limitations as detailed by the regulations.

Conduction of a complete parameter sweep of all evolutionary operators, including mutation and crossover rates, was not possible. Recommendations from the literature were adhered to as much as possible, and where such references were not available, popularly accepted settings were used. These settings remained consistent throughout the duration of the study.

All possible options for grammar generation were not explored. While this thesis only used traditional grammar forms, recently there have been advances in tree-adjoining grammars [136] and probabilistic grammars [137] which have been developed for GE. Similarly, only the most popular methods of selection and replacement (tournament selection and generational replacement) were used, although numerous different methods exist. These are examined in the literature review, but not implemented, as the focus of this thesis was on engineering optimisation.

All joints between members are treated as fully pinned in this thesis. Accurate modelling, along with comparisons of different jointing methods, was not possible as this was beyond the scope of this thesis. Jointing systems are however identified as a potential area for future research in Section 7.4.

7.4. Future Work

Although the field of structural engineering optimisation is well-established, new technological innovations mean it is constantly growing and evolving as new fields of research are discovered. A number of potential future research areas have been identified during the course of this thesis.

I. From the review of the literature, it was noted that Zhou [100] evolved continuum structures with the objective function of optimising particular fundamental frequencies for the structure itself. One interesting area of potential research would be to look at optimising discrete structures for this same purpose. An example application would be to evolve a bridge with specific fundamental frequencies which would limit the effects of heavy traffic or human footfall. This lessened impact would have a consequent beneficial effect on the maintenance and overall lifespan of the bridge by reducing fatigue.

II. Michell [5] and Chan [6] noted that if it is possible to design a structure whereby all members are either in tension (such as catenary structures) or all members are in compression (such as arch structures), then the global optimum for that particular problem has been found, and no better solution can exist. An optimisation method which evolves pure tensile or pure compressive structures warrants further investigation.

III. Large structures are difficult to truly optimise in a discrete fashion, as their physical size renders the requisite representation space infeasibly large. However, they have been successfully evolved with techniques which use simple repeatable elements or bays [68]. The inclusion of a recursive/fractional component to a grammar would allow it to generated repeatable periodic features, in a similar fashion to Huang and Xie's work [105]. Combined with the techniques described in this thesis, this method could prove very powerful for design of large structures where it may not be computationally economical to evolve structures over the complete design domain.

IV. Chapter 6 demonstrated that the SEOIGE method can be readily extended to three dimensions. An interesting future application would be to evolve tensegrity structures using this method. Tensegrity structures require struts for members in pure compression and cables for members in pure tension. Class 1 tensegrities are considered extremely difficult to design; they are defined as structures with only one compressive member per nodal joint, with all other members being cables in tension.

A heuristic discrete search algorithm such as SEOIGE would be well suited to evolving viable solutions, especially since the form of the solution is not known beforehand.

V. Joint connections between structural members could be more accurately modelled to further increase the analytical accuracy of the method. At present all joints between members are treated as fully pinned. Since it was not possible to accurately model, compare, and contrast different jointing methods due to the scope limit of this thesis, the method could benefit from this work in the future.

VI. The use of variable mutation and crossover rates [66, 131] has been proven to improve evolutionary capabilities with other methods. This area warrants investigation as it could yield additional performance benefits.

VII. The ground structure problems attempted in this thesis are very small. Ground structures with a very high number of members (> 100,000,000) have been successfully solved in the literature [87]. Evolution of structures on a similar scale using techniques such as those described in this thesis would be a difficult challenge, and would be worthy of further study.

VIII. Multiple objective optimisation has been successfully applied to GE [56, 50]. Design of multiple load cases for structures could be attempted using individual load case fitness values as multiple objectives.

IX. An investigation into varying connection algorithms other than the Delaunay triangulation method described in Section 6.3.6 could not only yield increases in performance, but could potentially be used for architectural and aesthetic purposes.

X. Design of structures in three dimensions would be a particular challenge with the SEOIGE method described in Chapter 6. At present structures can be generated in 3D, but the Delaunay triangulation method is more suited to surface mesh generation (e.g. in the generation of finite element meshes or surface textures). A particular challenge would be to find a method capable of generating internal structures for 3D shapes.

Appendix A: Grammars & Phenotypes

1. SEOIGE Grammar - Cantilevered Truss

```
<S> ::= <program>{}<call>
     # initialises the grammar
<program> ::= def mutant():{<init>{}<constants>{}<get_node>{}<cross_brace>{}<get_genome>{}
                          <make_truss>{}<make_all>{}<return>{}}
     # defines the main function call of the phenotype
<init> ::= truss_graph = graph.graph(){}state = random.getstate(){}
     # initialises the graph class
<constants> ::= span = <span>{}depth = <depth>{}edge_list = []{}
               truss_graph.save_graph_info(["delaunay", span, depth]){}
     # sets the variables for span and depth
<get_node> ::= def get_node(node):{
                    x = round(float(span)/100 * (node[1] + (node[1]*(100-99.99)/99.99))){}
                    y = round(float(depth)/100 * (node[2] + (node[2]*(100-99.99)/99.99))){}
                    return [node[0], x, y, node[3]]}
     # function definition for percentage equaliser function, as defined in Eq. 6.2
<cross_brace> ::= def cross_brace():{essential_list = [[0, 0, depth, "fixed", [True, True,
     True]],[0, <span>, 0, "load",<load>],[0, <span>/2, 0, "load", <load>],[0, 0, 0, "fixed",
     [True, True, True]]]{}optional_list = [<node_iter>]{}all_nodes = []{}for node in
     essential_list:{if node not in all_nodes:{ all_nodes.append(node)}}for node in
     optional_list:{brogue = get_node(node){} if node not in all_nodes:
     {all_nodes.append(brogue)}}{}a_set = []{}b_set = []{}for point in all_nodes:{node =
     [point[1], point[2], point[0]]{}label = point[3]{}if label == "load":{load = point[4]{}item
     = [node, label, load, none]} elif label == "fixed":{fix = point[4]{} item = [node, label,
     None, fix]}else:{item = [node, label, None, None]}{}if a_set:{if item not in a_set:{if
     item[0] not in b_set:{a_set.append(item){}b_set.append(item[0])}else:{break}}}
     else:{a_set.append(item){}b_set.append(item[0])}}a = []{}b = []{}for node in a_set:{node_id
     = truss_graph.add_unique_node(node[0], str(node[1]), node[2],
     node[3]){}a.append(node[0][0]){}b.append(node[0][1])}cens,edg,tri,neig =
     triang.delaunay(a,b){}for edge in edg:{if edge not in edge_list:{edge_list.append(edge)}}}
     # function definition for truss generation, including Delaunay triangulation connections
<get_genome> ::= def get_genome(i):{return genome[i]}
     # function definition for returning material section size from Ch.B
<make_truss> ::= def make_truss():{for i, edge in enumerate(edge_list):{
     if truss_graph.node[edge[0]]['label'] == "fixed" and truss_graph.node[edge[1]]['label'] ==
     "fixed":{pass}else:{truss_graph.add_edge(edge[0], edge[1], material=get_genome(i),
     genome_id=i)}}}
     # function definition for graph generation
<load> ::= [0,-444800,0]
```

```
     # load
<node> ::= [0, <%>, <%>, "node"]
     # defines a node
<%> ::= float("<n><n>.<n><n>")
     # defines a nodal co-ordinate
<n> ::= 0|1|2|3|4|5|6|7|8|9
     # 10 terminal production choices for the non-terminal <n>
<span> ::= 18288
     # span
<depth> ::= 9144
     # depth
<node_iter> ::= <node> | <node>,<node_iter>
     # recursive node call, allowing the grammar to produce any number of node locations
<make_all> ::= cross_brace(){}make_truss(){}random.setstate(state){}
     # function call for all defined functions
<return> ::= return [truss_graph, len(edge_list), "cantilever"]{}
     # return for resultant solution
<call> ::= XXXeval_or_exec_outputXXX = mutant()
     # call to execute phenotype
```

2. SEOIGE Phenotype - Cantilevered Truss

```
def program():
    truss_graph = graph.graph()
     # initialises an instance of the graph class
    span = 18288
    depth = 9144
    edge_list = []
    truss_graph.save_graph_info(["delaunay", span, depth])
    def get_node(node):
    #     function definition for percentage equaliser function, as defined in Eq. 6.2
        x = round(span/100 * (node[1] + (node[1]*(100-99.99)/99.99)))
        y = round(depth/100 * (node[2] + (node[2]*(100-99.99)/99.99)))
        return [node[0], x, y, node[3]]
    def cross_brace():
    #     function definition for truss generation, including Delaunay triangulation connections
        essential_list = [[0,0 , depth, "fixed", [True, True, True]],[0, 18288, 0, "load", [0,-
444800,0]],[0, 18288/2, 0, "load", [0,-444800,0]],[0, 0, 0, "fixed", [True, True, True]]]
        # define the essential list
        optional_list = [[0, float("41.42"), float("83.51"), "node"],[0, float("54.99"),
float("38.78"), "node"]]
        # define the optional list
        all_nodes = []
        for node in essential_list:
            if node not in all_nodes:
                all_nodes.append(node)
        for node in optional_list:
            new_node = get_node(node)
            if node not in all_nodes:
                all_nodes.append(new_node)
        a_set = []
        b_set = []
        for point in all_nodes:
        # set the node definitions for the graph class
            node = [point[1], point[2], point[0]]
            label = point[3]
            if label == "load":
                load = point[4]
                item = [node, label, load, None]
            elif label == "fixed":
                fix = point[4]
                item = [node, label, None, fix]
            else:
                item = [node, label, None, None]
            if a_set:
```

```
        if item not in a_set:
        #   ensure we have no duplicate nodes
            if item[0] not in b_set:
                a_set.append(item)
                b_set.append(item[0])
            else:
                break
    else:
        a_set.append(item)
        b_set.append(item[0])
a = []
b = []
for node in a_set:
# add node information to the graph
    node_id = truss_graph.add_unique_node(node[0], str(node[1]), node[2], node[3])
    a.append(node[0][0])
    b.append(node[0][1])
cens,edg,tri,neig = triang.delaunay(a,b)
# generate the Delaunay mesh connections given the list of nodes
for edge in (edg):
    if edge not in edge_list:
        edge_list.append(edge)
```

```
def get_genome(i):
    return genome[i]
def make_truss():
#     create graph edges from the edge list
    for i, edge in enumerate(edge_list):
        if truss_graph.node[edge[0]]['label'] == "fixed"
        and truss_graph.node[edge[1]]['label'] == "fixed":
            pass
        else:
            truss_graph.add_edge(edge[0], edge[1], material=get_genome(i), genome_id=i)
cross_brace()
#          function call to generate nodes and edges
make_truss()
#          function call to generate graph
return [truss_graph, len(edge_list), "cant"]
```

Appendix B: File Formats

1. SLFFEA Input File Format (abridged)

```
numel numnp nmat nmode  (This is for a truss)
9    6    157  0
matl no., E modulus, density, and Area
0    210000    7.85e-06  153.0
1    210000    7.85e-06  2140.0
2    210000    7.85e-06  3310.0
3    210000    7.85e-06  3710.0
4    210000    7.85e-06  4210.0
5    210000    7.85e-06  4700.0
el no., connectivity, matl no
0    0    4    5
1    0    5    4
2    1    2    5
3    1    4    2
4    1    5    0
5    2    3    4
6    2    5    1
7    3    5    5
8    4    5    3
node no., coordinates
0    0    9144 0
1    18288     0    0
2    9144 0    0
3    0    0    0
4    10749     7600 0
5    8278 3529 0
prescribed displacement x: node  disp value
0    0.0
```

```
3    0.0
-10
prescribed displacement y: node  disp value
0    0.0
3    0.0
-10
prescribed displacement z: node  disp value
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
5    0.0
-10
node with point load and load vector in x, y, z
1    0    -444800   0
2    0    -444800   0
-10
element with stress and tensile stress vector
-10
```

2. SLFFEA Output File Format (abridged)

```
   numel numnp nmat nmode (This is for the truss mesh file:
slf/1)
       9    6  157    0
 matl no., E modulus, density, Area
     0    2.100000e+05 7.850000e-06 1.530000e+02
     1    2.100000e+05 7.850000e-06 2.140000e+03
     2    2.100000e+05 7.850000e-06 3.310000e+03
     3    2.100000e+05 7.850000e-06 3.710000e+03
     4    2.100000e+05 7.850000e-06 4.210000e+03
     5    2.100000e+05 7.850000e-06 4.700000e+03
el no., connectivity, matl no.
     0      0      4      5
     1      0      5      4
     2      1      2      5
     3      1      4      2
     4      1      5      0
     5      2      3      4
     6      2      5      1
     7      3      5      5
     8      4      5      3
node no., coordinates
   0       0.000000    9144.000000       0.000000
   1   18278.132161     -50.797659       0.000000
   2    9138.261321     -22.855161       0.000000
   3       0.000000       0.000000       0.000000
   4   10752.914765    7576.391149       0.000000
   5    8277.201854    3511.170309       0.000000
prescribed displacement x: node  disp value
   0   0.000000e+00
   1  -9.867839e+00
   2  -5.738679e+00
   3   0.000000e+00
   4   3.914765e+00
   5  -7.981465e-01
 -10
prescribed displacement y: node  disp value
   0   0.000000e+00
   1  -5.079766e+01
   2  -2.285516e+01
   3   0.000000e+00
   4  -2.360885e+01
   5  -1.782969e+01
 -10
prescribed displacement z: node  disp value
   0   0.000000e+00
   1   0.000000e+00
   2   0.000000e+00
   3   0.000000e+00
   4   0.000000e+00
   5   0.000000e+00
 -10
node with point load and load vector in x,y,z
   0  -1.334400e+06    5.572703e+05    0.000000e+00
   1  -4.110916e-10   -4.448000e+05    0.000000e+00
   2   2.910383e-10   -4.448000e+05    0.000000e+00
   3   1.334400e+06    3.323297e+05    0.000000e+00
   4  -1.164153e-10    5.820766e-10    0.000000e+00
   5   2.328306e-10    5.820766e-11    0.000000e+00
 -10
element no. with stress and tensile stress vector
   0    1.398491e+02
   1    1.962590e+02
   2   -9.482980e+01
   3    1.882505e+02
   4    4.763997e+01
   5   -1.317938e+02
   6    2.140172e+02
```

```
    7   -1.803043e+02
    8   -1.100180e+02
  -10
element no. with strain and tensile strain vector
    0    6.659480e-04
    1    9.345665e-04
    2   -4.515705e-04
    3    8.964311e-04
    4    2.268570e-04
    5   -6.275895e-04
    6    1.019130e-03
    7   -8.585921e-04
    8   -5.238952e-04
  -10
```

Bibliography

[1] J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu and G. Lanza, Genetic Programming IV: Routine Human-Competitive Machine Intelligence., Springer, 2005.

[2] K. Sörensen, "Metaheuristics - the metaphor exposed," *International Transactions in Operational Research*, pp. 1-16, 2013.

[3] K. A. DeJong, Evolutionary Computation: A unified approach, Cambridge, Massachusetts: The MIT Press, 2006.

[4] T. Jones, "Evolutionary Algorithms, Fitness Landscapes, and Search," Working Papers 95-05-048, Santa Fe Institute, 1995 a.

[5] A. G. M. Michell, "The limits of economy of material in frame structures.," *Philosophical Magazine,* vol. 8, no. 47, pp. 589-597, 1904.

[6] A. S. L. Chan, "The design of michell optimum structures," The College of Aeronautics, Cranfield, 1960.

[7] G. I. N. Rozvany, "A critical review of established methods of structural topology optimization," *Structural and Multidisciplinary Optimization,* vol. 37, pp. 217-237, 2009.

[8] M. P. Bendsøe and O. Sigmund, Topology Optimization: Theory, Methods and Applications, Berlin: Springer-Verlang, 2003.

[9] A. Ghali, A. M. Neville and T. G. Brown, Structural Analysis: A Unified Classical and Matrix Approach (6th Ed.), New York: Spon Press, 2009.

[10] W. Dorn, R. Gomory and H. Grenberg, "Automatic design of optimal structures," *Journal de Mechanique,* vol. 3, no. 1, pp. 25-52, 1964.

[11] X. Huang and Y. M. Xie, “A further review of ESO type methods for topology optimization,” *Structural and Multidisciplinary Optimization,* vol. 41, pp. 671-683, 2010 b.

[12] R. Kicinger, T. Arciszewski and K. A. DeJong, "Evolutionary Computation and Structural Design: A survey of the state of the art," *Computers and Structures,* vol. 83, no. 23-24, pp. 1-47, 2005.

[13] G. I. N. Rozvany, “Exact analytical solutions for some popular benchmark problems in topology optimization,” *Structural and Multidisciplinary Optimization,* vol. 15, pp. 42-48, 1998.

[14] K. Deb and S. Gulati, “Design of truss-structures for minimum weight using genetic algorithms.,” *Finite Elements in Analysis and Desing,* vol. 37, pp. 447-465, 2001.

[15] G. I. N. Rozvany, N. Olhoff, M. p. Bendsøe, T. G. Ong and W. T. Szeto, “Least-weight design of perforated elastic plates,” Lyngby, 1985.

[16] A. Turing, “Computing machinery and intelligence,” *Mind 59,* vol. 59, pp. 433-460, 1950.

[17] L. Samuel, “AI, where has it been and where is it going,” in *International Joint Conferences on Artificial Intelligence*, Karlsruhe, 1983.

[18] C. Darwin, On the origin of species by means of natural selection, London: John Murray, 1859.

[19] J. H. Holland, Adaptation in natural and artificial systems, 2nd ed., Ann Arbour, Michigan: University of Michigan Press, 1975.

[20] W. Banzhaf, J. R. Koza, C. Ryan, L. Spector and C. Jacob, “Genetic programming,” *Intellegent Systems and their Applications,* vol. 15, no. 3, pp. 74-84, 2000.

[21] M. A. Kaliakatsos-Papakostas, A. Floros and M. N. Vrahatis, “evoDrummer: Deriving rhythmic patterns through interactive genetic algorithms,” in *EvoMUSART*, Berlin Heidelberg, 2013.

[22] J. McDermott and U. M. O'Reilly, “An executable graph representation for evolutionary generative music,” in *GECCO '11*, Dublin, 2011.

[23] M. Nicolau and D. Costelloe, “Using grammatical evolution to parameterise interactive 3D image generation,” in *Applications of Evolutionary Computation*, vol. 6625, C. Chio, A. Brabazon, G. Caro, R. Drechsler, M. Farooq, J. Grahl, G. Greenfield, G. Prins, J. Romero, G. Squillero, E. Tarantino, A. G. B. Tettamanzi, N. Urquhart and A. S. Uyar, Eds., Springer Berlin Heidelberg, 2011, pp. 374-383.

[24] N. Shaker, M. Nicolau, G. N. Yannakakis, J. Togelius and M. O'Neill, “Evolving levels for super mario bros using grammatical evolution,” in *IEEE Conference on Computational Intelligence and Games*, Granada, 2012.

[25] S. M. Lucas and G. Kendall, “Evolutionary Computation and Games,” *IEEE Computational Intelligence Magazine,* vol. 1, no. 1, pp. 10-18, 2006.

[26] M. O'Neill and A. Brabazon, “Recent patents on genetic programming,” *Recent Patents on Computer Science,* vol. 2, no. 1, pp. 43-49, 2009.

[27] L. D. Lohn, D. S. Linden, G. S. Hornby, W. F. Kraus, A. Rodriguez and S. Seufert, “Evolutionary design of an X-band antenna for NASA's space technology 5 mission,” in *IEEE Antenna and Propagation Society Internation Symposiom and USNC/URSI National Radio Science Meeting,* 2004.

[28] A. Brabazon and M. O'Neill, “Anticipating bankrupcy reorganisation from raw financial data using grammatical evolution,” in *Applications of Evolutionary Computing*, Springer Berlin Heidelberg, 2003, pp. 368-377.

[29] J. McDermott, J. M. Swafford, M. Hemberg, C. McNally, E. Shotton, E. Hemberg, M. Fenton and M. O'Neill, “String-rewriting grammars for evolutionary architectural design,” *Environment and Planning B: Planning and Design,* vol. 39, no. 4, pp. 716-731, 2012.

[30] L. J. Fogel, A. J. Owens and M. J. Walsh, Artificial intelligence through simulated evolution, New York: Wiley, 1966.

[31] H. G. Beyer and H. P. Schwefel, “Evolutionary strategies: a compreshensive introduction,” *Natural Computing,* vol. 1, no. 1, pp. 3-52, 2002.

[32] I. Rechenberg, “Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien,” *Fromman-Holzboog,* vol. 104, 1973.

[33] J. R. Koza, Genetic Programming: On the Programming of Computers by Means, Cambridge, MA: MIT Press, 1992.

[34] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison Wesley, 1989.

[35] D. E. Goldberg and K. Deb, “A comparative analysis of selection schemes used in genetic algorithms,” *Urbana,* vol. 51, 1991.

[36] L. D. Whitley, “The GENITOR algorithm and selection pressure: why ranked-based allocation of reproductive trials is best,” in *Proceedings of the Third International Conference on Genetic Algorithms (ICGA'89),* Fairfax, VA, USA, 1989.

[37] K. A. DeJong, “Parameter setting in EAs: a 30 year perspective,” in *Studies in Computational Intelligence,* vol. 54, F. Lobo, C. Lima and Z. Michalewicz, Eds., Berlin / Heidelberg, Springer, 2007, pp. 1-18.

[38] M. Mitchell, An introduction to genetic algorithms, MIT Press, 1998.

[39] D. Gupta and S. Ghafir, “An overview of methods maintaining diversity in genetic algorithms,” *International Journal of Emerging Technology and Advanced Engineering,* vol. 2, no. 5, pp. 56-60, 2012.

[40] M. Li, C. O'Riordan and S. Hill, “An analysis of multi-chromosome GAs on deceptive problems,” in *Proceedings of the 13th annual conference on Genetic and Evolutionary Computation,* Dublin, 2011.

[41] R. Cavill, S. L. Smith and A. M. Tyrrell, “Variable length genetic algorithms with multiple chromosomes on a variant of the onemax problem,” in *Proceedings of the 8th annual conference on Genetic and Evolutionary Computation,* Seattle, WA, 2006.

[42] R. Poli, W. B. Langdon, N. F. McPhee and J. R. Koza, A field guide to genetic programming, 2008.

[43] R. Cavill, S. Smith and A. Tyrrell, “Multi-chromosomal genetic programming,” in *Proceedings of the 2005 conference on Genetic and evolutionary computation*, Washington, DC, 2005.

[44] C. Ryan, J. J. Collins and M. O'Neill, “Grammatical evolution: evolving programs for an arbitrary language,” *Genetic Programming,* pp. 83-96, 1998.

[45] J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, P. Naur, A. J. Perlis, H. Rutishauser, K. Samelson and B. Vauquois, “Revised report on the algorithmic language ALGOL 60,” *The Computer Journal,* vol. 5, no. 4, pp. 349-367, 1963.

[46] M. O'Neill and C. Ryan, Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language, Kluwer Academic Publishers, 2003.

[47] J. Byrne, Approaches to evolutionary architectural design exploration using grammatical evolution (Thesis), Dublin: UCD, 2013.

[48] M. O'Neill and C. Ryan, “Grammatical Evolution,” *IEEE Transactions on Evolutionary Computation,* vol. 5, no. 4, pp. 349-358, 2001.

[49] M. Kimura, The neutral theory of molecular evolution, Cambridge University Press, 1983.

[50] J. Byrne, M. Fenton, E. Hemberg, J. McDermott and M. O'Neill, “Optimising complex pylon structures with grammatical evolution,” *Information Sciences,* no. In Press, 2014.

[51] Y. Li and Y. Chen, “Beam Structure Optimization for Additive Manufacturing based on Principal Stress Lines,” in *Proceedings of Solid Freeform Fabrication Symposium*, Austin, Texas, 2010.

[52] H. L. Cox, “The theory of design,” Aeronautical Research Council, 1958.

[53] W. S. Hemp, “Theory of Structural Design,” College of Aeronautics, Cranfield, 1958.

[54] J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu and G. Lanza, Genetic Programming IV. Routine Human-Competitive Machine Intelligence, Boston, MA: Kluwer Academic Publishers, 2003.

[55] M. O'Neill, J. McDermott, J. M. Swafford, J. Byrne, E. Hemberg, E. Shotton, C. McNally, A. Brabazon and M. Hemberg, "Evolutionary Design using Grammatical Evolution and Shape Grammars: Designing a shelter," *International Journal of Design Engineering,* vol. 3, no. 1, pp. 4-24, 2010.

[56] J. Byrne, M. Fenton, E. Hemberg, J. McDermott, M. O'Neill, B. Shotton and C. McNally, "Combining structural analysis and multi-objective criteria for evolutionary architectural design," in *C. Di Chio et al. (Eds.): Applications of Evolutionary Computation, Part II, LNCS 6625,* Berlin Heidelberg, Springer-Verlang, 2011, pp. 204-213.

[57] F. Cobb, Structural Engineer's Pocketbook, Second Edition, Oxford: Butterworth-Heinemann, 2009.

[58] E. Glaylord and C. Glaylord, Structural engineering handbook, McGraw-Hill, 1979.

[59] British Standards Institution, "BS 449-2: 1969, The structural use of steel in building," BSI, London, 1969.

[60] British Standards Institution, *BS 5950-1: 2000, Structural Use of Steelwork in Building,* London: BSI, 2000.

[61] M. Fenton, "Analysis of timber structures created using a GE-based architectural design tool," University College Dublin, Dublin, 2010.

[62] N. Pholdee and S. Bureerat, "Performance Enhancement of Multi-objective Evolutionary Optimisers for Truss Design using an Approximate Gradient," *Computers and Structures,* no. 106/107, pp. 115-124, 2012.

[63] M. Ohsaki, "Genetic Algorithm for Topology Optimization of Trusses," *Computers and Structures,* vol. 57, no. 2, pp. 219-225, 1995.

[64] L. J. Li, Z. B. Huang, F. Liu and Q. H. Wu, “A heuristic particle swarm optimizer for optimization of pin connected structures,” *Computers and Structures,* vol. 85, pp. 340-349, 2007.

[65] A. Kaveh and S. Talatahari, “Particle swarm optimizer, ant colony strategy and harmony search scheme hybridized for optimization of truss structures,” *Computers and Structures,* vol. 87, no. 5-6, pp. 267-283, 2009.

[66] K. Murawski, T. Arciszewski and K. De Jong, “Evolutionary computation in structural design,” *Engineering with computers,* vol. 16, pp. 275-286, 2000.

[67] D. E. Grierson and G. E. Cameron, “Microcomputer-based optimisation of steel structures in professional practice,” *Microcomputers in Civil Engineering,* vol. 4, no. 2, pp. 289-296, 1989.

[68] R. Kicinger, T. Arciszewski and K. A. DeJong, "Evolutionary design of steel structures in tall buildings," *Journal of Computing in Civil Engineering,* vol. 19, no. 3, pp. 223-237, 2005.

[69] P. Hajela and E. Lee, “Genetic Algorithms in Truss Topological Optimization,” *International Journal of Solids and Structures,* vol. 32, no. 22, pp. 3341-3357, 1995.

[70] S. Ruiyi, G. Liangjin and F. Zijie, “Truss topology optimization using genetic algorithm with individual identification technique,” in *Proceedings of the World Congress on Engineering 2009*, London, 2009.

[71] F. Glover and M. (. T. s. (. 2.-2. S. U. Laguna, "Tabu search," in *Handbook of combinatorial optimization*, vol. 3, D. Du and P. M. Pardalos, Eds., Springer US, 1999, pp. 2093-2229.

[72] O. Bohnenberger, J. Hesser and R. Männer, “Automatic Design of Truss Structures Using Evolutionary Algorithms,” in *IEEE Conference on Evolutionary Computation*, Perth, 1995.

[73] H. Kawamura, H. Ohmori and N. Kito, “Truss topology optimization by a modified genetic algorithm,” *Structural and Multidisciplinary Optimization,* vol. 23, pp. 467-472, 2002.

[74] R. C. Eberhart and J. Kennedy, “A new optimizer using particle swarm theory,” in *Proceedings of the sixth international symposium on micro machine and human science,* Nagoya, Japan, 1995.

[75] Z. W. Geem, J.-H. Kim and g. v. Longanathan, “A new heuristic optimization algorithm: harmony search,” *Simulation,* vol. 76, pp. 60-68, 2001.

[76] K. S. Lee and Z. W. Geem, “A new structural optimization method based on the harmony search algorithm,” *Computers and Structures,* vol. 82, pp. 781-798, 2004.

[77] G. C. Luh and C. Y. Lin, “Optimal design of truss-structures using particle swarm optimization,” *Computers and Structures,* vol. 89, pp. 2221-2232, 2011.

[78] R. T. Marler and J. S. Arora, “Survey of multi-objective optimization methods for engineering,” *Structural and Multidisciplinary Optimization,* vol. 26, pp. 369-395, 2004.

[79] T. Jones, “One Operator, One Landscape,” Working Papers 95-02-025, Santa Fe Institute, 1995 b.

[80] Z. Michalewicz, "A Survey of Constraint Handling Techniques in Evolutionary Computation Methods," in *In J.R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, Proceedings of the 4th Annual Conference on Evolutionary Programmin*, Cambridge, MA, 1995.

[81] T. Aittokoski and K. Miettinen, " Efficient evolutionary approach to approximate the Pareto-optimal set in multiobjective optimization: UPS-EMOA," *Optimiz Methods Software,* vol. 25, pp. 841-858, 2010.

[82] E. Zitzler, M. Laumanns and L. Thiele, "SPEA2: improving the strength Pareto evolutionary algorithm for multiobjective optimization.," in *Evolutionary methods for design, optimization and control*, Barcelona, Spain, 2002.

[83] M. R. Sierra and C. A. Coello Coello, "Multi-objective particle swarm optimizers: a survey of the state-of-the-art," *Int J Comput Intell Res,* vol. 2, p. 287–308, 2006.

[84] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, “A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II,” *IEEE Transactions on Evolutionary Computation,* vol. 6, no. 2, pp. 182-197, April 2002.

[85] K. Shea and I. F. C. Smith, “Improving full-scale transmission tower design through topology and shape optimization,” *Journal of Structural Engineering,* vol. 132, no. 5, pp. 781-790, 2006.

[86] L. L. Stromberg, A. Beghini, W. F. Baker and G. H. Paulino, “Topology Optimization for Braced Frames: Combining continuum and beam/column elements,” *Engineering Structures,* vol. 37, pp. 106-124, 2012.

[87] M. Gilbert and A. Tyas, "Layout optimisation of large pin-jointed frames," *Engineering Computations,* vol. 20, no. 8, pp. 1044-1064, 2003.

[88] T. Zegard and G. H. Paulino, "Truss layout optimisation within a continuum," *Struct Multidisc Optim,* vol. 48, pp. 1-16, 2013.

[89] P. Wei, H. Ma and M. Y. Wang, “The stiffness spreading method for layout optimization of truss structures,” *Structural and Multidisciplinary Optimization,* vol. 49, pp. 667-682, 2014.

[90] T. Sokol, "A 99 line code for discretized Michell truss optimization written in Mathematica," *Struct Multidisc Optim,* vol. 43, no. 2, pp. 181-190, 2011.

[91] X. Huang and Y. M. Xie, “A new look at ESO and BESO optimization methods,” *Structural and Multidisciplinary Optimization,* vol. 25, pp. 89-92, 2008 b.

[92] M. Zhou and G. I. N. Rozvany, “On the validity of ESO type methods in topology optimization,” *Structural and Multidisciplinary Optimization,* vol. 21, pp. 80-83, 2001.

[93] M. P. Bendsøe, “Optimal shape design as a material distribution problem,” *Structural Optimization,* vol. 1, no. 4, pp. 193-202, 1989.

[94] O. Sigmund, “A 99 line topology optimization code written in Matlab,” *Structural and Multidisciplinary Optimization,* vol. 21, pp. 120-127, 2001.

[95] H. A. Eschenauer and N. Olhoff, “Topology optimization of continuum structures: a review,” *Applied Mechanics Reviews,* vol. 54, no. 4, pp. 331-390, 2001.

[96] M. P. Bendsøe and N. Kikuchi, “Generating optimal topologies in structural design using a homogenization method,” *Computer Methods in Applied Mechanics and Engineering,* vol. 71, pp. 197-224, 1988.

[97] G. I. N. Rozvany, M. Zhou and T. Birker, “Generalized shape optimization without homogenization,” *Structural Optimization,* vol. 4, no. 3-4, pp. 250-252, 1992.

[98] M. Bruggi and P. Duysinx, “Topology optimization for minimum weight with compliance and stress constraints,” *Structural and Multidisciplinary Optimization,* vol. 46, pp. 369-384, 2012.

[99] D. Drucker and W. Prager, “Soil mechanics and plastic analysis or limit design,” *The Quarterly of Applied Mathematics,* vol. 10, pp. 157-165, 1952.

[100] K. Zhou, “Topology optimization of truss-like continuum structures for natural frequencies,” *Structural and Multidisciplinary Optimization,* vol. 47, pp. 613-619, 2013.

[101] Y. M. Xie and G. P. Steven, “A simple evolutionary procedure for structural optimization,” *Computers and Structures,* vol. 49, no. 5, pp. 885-896, 1993.

[102] O. Querin, G. Steven and Y. Xie, “Evolutionary structural optimization (ESO) using a bidirectional algorithm,” *Engineering Computations,* vol. 15, no. 8, pp. 1031-1048, 1998.

[103] O. M. Querin, G. P. Steven and Y. M. Xie, “Evolutionary structural optimization using an addative algorithm,” *Finite Elements in Analysis and Design,* vol. 34, no. 3-4, pp. 291-308, 2000.

[104] X. Huang and Y. M. Xie, "Bi-directional evolutionary topology optimization of continuum structures with one or multiple materials," *Computational Mechanics,* vol. 43, pp. 393-401, 2009.

[105] X. Huang and Y. M. Xie, "Optimal design of periodic structures using evolutionary topology optimization," *Structural and Multidisciplinary Optimization,* vol. 36, pp. 597-606, 2008 a.

[106] M. J. Jakiela, C. Chapman, J. Duda, A. Adewuya and K. Saitou, "Continuum structural topology design with genetic algorithms," *Computer Methods in Applied Mechanics and Engineering,* vol. 186, pp. 339-356, 2000.

[107] Z. H. Zuo, Y. M. Xie and X. Huang, "Combining genetic algorithms with BESO for topology optimization," *Structural and Multidisciplinary Optimization,* vol. 38, pp. 511-523, 2009.

[108] X. Huang, Y. M. Xie and K. Ghabraie, "Computational algorithms for generating efficient and innovative load-carrying structures," in *Proceedings of the 3rd WSEAS International Conference on Computer Engineering and Applications (CEA '09),* Univ Ningbo, China, 2009.

[109] C. A. Coello Coello, "A survey of constraint handling tehniques used with evolutionary algorithms," Lania-RI-99-04, Laboratorio de Informatica Avanzada, Veracruz, Mexico, 1999.

[110] C. A. Coello Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms," *Computer Methods in Applied Mechanics and Engineering,* vol. 191, pp. 1245-1287, 2002.

[111] J. T. Richardson, M. R. Palmer, G. E. Liepins and M. R. Hilliard, "Some guidelines for genetic algorithms with penalty functions," in *Proceedings of the Third International Conference on Genetic Algorithms (ICGA'89),* Fairfax, VA, 1989.

[112] K. Deb, "An efficient constraint handling method for genetic algorithms," *Computer Methods in Applied Mechanics and Engineering,* vol. 186, pp. 311-338, 2000.

[113] R. I. McKay, N. Xuan Hoai, P. A. Whigham, Y. Shan and M. O'Neill, "Grammar-based genetic programming: a survey," *Genetic Programming and Evolvable Machines,* vol. 11, no. 3/4, pp. 365-396, 2010.

[114] J. Hugosson, E. Hemberg, A. Brabazon and M. O'Neill, "Genotype Representations in Grammatical Evolution," *Applied Soft Computing,* vol. 10, pp. 36-43, 2010.

[115] I. Dempsey, M. O'Neill and A. Brabazon, Foundations in Grammatical Evolution for Dynamic Environments, Springer, 2009.

[116] S. Le, "San Le's Free Finite Element Analysis," 2010. [Online]. Available: http://slffea.sourceforge.net/. [Accessed 2010].

[117] M. Fenton, C. McNally, J. Byrne, E. Hemberg, J. McDermott and M. O'Neill, "Automatic innovative truss design using grammatical evolution," *Automation in Construction,* vol. 39, pp. 59-69, 2014.

[118] TATA Steel, "Tata Steel Sections Interactive "Blue Book"," 2014. [Online]. Available: http://www.tatasteelconstruction.com/en/design_guidance/the_blue_book/. [Accessed 2013].

[119] British Standards Institution, *BS EN 10210-2: 2006, Hot finished structural hollow sections of non-alloy and fine grain steels. Tolerances, dimensions and sectional properties,* London: BSI, 2006.

[120] M. Nicolau, M. O'Neill and A. Brabazon, "Termination in grammatical evolution: grammar design, wrapping, and tails," in *ECCI 20120 IEEE World Congress on Computational Intelligence*, Brisbane, Australia, 2012.

[121] D. E. Goldberg, "Optimum initial population size for binary-coded genetic algorithms," TCGA Report no. 85001, 1985.

[122] M. O'Neill, C. Ryan, M. Keijzer and M. Cattolico, "Crossover in Grammatical Evolution," *Genetic Programming and Evolvable Machines,* vol. 4, pp. 67-93, 2003.

[123] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on*

Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation, vol. 8, no. 1, pp. 3-30, 1998.

[124] M. O'Neill, C. Ryan and M. Nicolau, “Grammar defined introns: an investigation into grammars, introns, and bias in grammatical evolution,” in *Genetic and Evolutionary Computation Conference (GECCO 2001),* San Francisco, 2001.

[125] T. Zegard, W. F. Baker, A. Mazurek and G. H. Paulino, "Geometrical aspects of lateral bracing systems: where should teh optimal bracing point be?," *Journal of Structural Engineering,* vol. 140, no. 9, 2014.

[126] B. L. Miller and D. E. Goldberg, “Genetic algorithms, tournament selection, and the effects of noise,” *Complex Systems,* vol. 9, pp. 193-212, 1995.

[127] P. Hajela, E. Lee and C. Y. Lin, “Genetic algorithms in structural topology optimization,” in *Topology Design of Structures,* vol. 227, M. Bendsoe and C. Soares, Eds., Kulwer Academic Publishers, 1993, pp. 117-133.

[128] N. S. Khot and L. Berke, “Structural optimization using optimality criteria methods,” in *New Directions in Optimum Structural Design*, E. Atrek, R. H. Gallagher, K. M. Ragsdell and O. C. Zienkiewicz, Eds., New York, John Wiley, 1984.

[129] J. Adeli and S. Kumar, “Distributed genetic algorithm for structural optimization,” *Journal of Aerospace Engineering,* vol. 8, no. 3, pp. 156-163, 1995.

[130] Z. Michalewicz, "Genetic algorithms, numerical optimization, and constraints," in *L.J. Eshelman (Ed.), Proceedings of the Sixth International Conference on Genetic Algotithms*, San Mateo, Morgan Kaufmann, 1995, pp. 151-158.

[131] D. Fagan, E. Hemberg, M. O'Neill and S. McGarraghy, “Fitness reactive mutation in grammatical evolution,” in *18th International Conference on Soft Computing,* Brno University of Technology, 2012.

[132] J. D. Deaton and R. V. Grandhi, "A survey of structural and multidisciplinary continuum topology optimization: post 2000," *Structural and Multidisciplinary Optimisation,* vol. 49, pp. 1-38, 2014.

[133] X. Huang and Y. M. Xie, Evolutionary Topology Optimization of Continuum Structures: Methods and Applications, John Wiley & Sons, 2010 a.

[134] B. Delaunay, “Sur la sphère vide,” *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk,* vol. 7, pp. 793-800, 1934.

[135] M. Abdoli, J. R. De Jong, J. Pruim, R. A. J. O. Dierckx and H. Zaidi, “Reduction of artefacts caused by hip implants in CT-based attenuation-corrected PET images using 2-D interpolation of a virtual sinogram on an irregular grid,” *European Journal of Nuclear Medicine and Molecular Imaging,* vol. 38, no. 12, pp. 2257-2268, 2011.

[136] E. Murphy, M. O'Neill and A. Brabazon, “A comparison of GE and TAGE in dynamic environments,” in *GECCO '11: Proceedings of the 13th annual conference on genetic and evolutinoary computation,* Dublin, 2011.

[137] E. A. P. Hemberg, An Exploration of Grammars in Grammatical Evolution, Ireland: University College Dublin, 2010.