Information Sciences 316 (2015) 582–597

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Optimising complex pylon structures with grammatical evolution

Jonathan Byrne *, Michael Fenton, Erik Hemberg, James McDermott, Michael O'Neill

University College Dublin, Ireland

ARTICLE INFO

Article history:
Received 16 July 2013
Received in revised form 19 February 2014
Accepted 4 March 2014
Available online 19 March 2014

Keywords:
Structural optimisation
Architecture
Grammar
Genetic programming

ABSTRACT

Evolutionary algorithms have proven their ability to optimise architectural designs but are limited by their representation, i.e., the structures that the algorithm is capable of generating. The representation is normally constrained to small structures, or parts of a larger structure, to prevent a preponderance of invalid designs. This work uses a grammar based representation to generate large scale pylon designs. It removes invalid designs from the search space, but still allows complex and large scale constructions. In order to show the suitability of this method to real world design problems, we apply it to the Royal Institute of British Architects pylon design competition. This work shows that a combination of a grammar representation with real world constraints is capable of exploring different design configurations while evolving viable and optimised designs.

1. Introduction

Evolution has shown that it is capable of generating elegant and parsimonious solutions to a myriad of design problems. It is also capable of generating multiple viable solutions to the same problem. Accordingly, variants of evolutionary algorithms (EAs) have been applied to design problems since their inception [29]. One of the biggest issues when evolving architectural designs is how to specify and limit the design space. The search space of all possible designs is infinite [21] and so the search space must be bounded by limiting the representation of possible designs. Grammars limit and formalise the expression of a language while still allowing the components of a language to be freely and recursively combined. This work demonstrates how grammar representations naturally lend themselves to describing large scale structural design while still allowing for exploration of different design possibilities.

Grammars offer significant advantages when applied to architectural optimisation. A design must first be analysed before it can be optimised and structural analysis can only be applied to a well formed design. Grammatical rules control what can be "said" in that language. Using grammars allows for the search space to be limited exclusively to valid (i.e. analysable) designs. Structural analysis requires information about where forces are applied and what points are fixed. Grammars provide a technique for component labelling that, in turn, provide a methodology for force application. Grammatical labelling means that structural analysis can be performed automatically, without having to manually specify the fixed points and forces for each design. An additional advantage of a grammar is that it is a concise representation.

Evolutionary algorithms need a method to judge the quality of individuals in the population. While design constraints are normally seen as a limitation by designers, they can provide a methodology for evaluating designs. This work incorporates

* Corresponding author. Tel.: +353 863257989.
E-mail address: jonathanbyrn@gmail.com (J. Byrne).

http://dx.doi.org/10.1016/j.ins.2014.03.010
0020-0255/© 2014 Elsevier Inc. All rights reserved.

constraints from a design specification document into a fitness function and uses it to drive evolutionary search. The structures generated by the grammar are evaluated depending on how they perform under real world loading conditions that were taken from the RIBA pylon design competition [30]. We investigate whether combining the goals described in design specification document directly with an evolutionary algorithm can generate suitable designs.

This work describes how a grammar representation, an evolutionary algorithm and real world constraints can be combined to generate architectural structures. This paper is organised as follows. Section 2 discusses related work in shape generating grammars and previous approaches to pylon optimisation. Section 3 gives an overview of how a genetic algorithm is used to select rules from a grammar to generate designs. Sections 4 and 5 explain how grammars can be used to decompose a design problem and how complexity can be obtained through grammatical techniques. Section 6 explains the methods used for computer based structural analysis and how it can be used with evolutionary algorithms. Section 7 describes the real world design challenge, the RIBA pylon design competition. Section 8 examines the optimisation of grammar representations based on real world constraints. Finally conclusions and future work are discussed in Sections 9 and 10.

2. Related research

There are several examples where evolutionary algorithms have been used to optimise electricity pylons. Bohnenberger et al. [3] optimised the leg of an electricity pylon using a genetic algorithm and an indirect coding, i.e., an indirect mapping procedure. The mapping procedure was required as they were confronted by the same issue of a large amount of invalid designs in the search space. The mapping rules enabled the edges of their graph structure to be subdivided and nodes added to increase the overall strength of the pylon leg. Although the optimisation was limited to a small component of the pylon and the force and fixed points were specified in advance, the approach showed that an indirect genome mapping could be used for truss optimisation.

Deb and Gulati [8] optimised 2D and 3D truss structures similar to a pylon arm using a GA and a fixed length representation scheme. The vector of design variables represented member areas and positioning of nodal coordinates. Although the representation had a fixed length, it used member exclusion to create topological variations. This approach places a maximum bound on the search space but can still explore the possibilities within that search space. Despite limiting the representation there were issues with redundant nodes. The nodes in the structure were classified into basic nodes, which support the truss or apply a load, and non-basic nodes, which neither support nor bear a load. Structures containing non-basic nodes were not evaluated to speed up computation. Their fitness function used constraint handling for several design parameters such as self weight, stress, deflection and kinematic stability.

The largest scale optimisation of electricity pylons was the work done by Shea et al. [31]. They simultaneously optimised topology, sizing and shape of the electricity pylon (the distinction between topology, sizing and shape is discussed in greater detail in Section 6.1). Their approach used structural topology and shape annealing (STSA) which combined a stochastic optimisation technique with a relational grammar. They optimised both the number and connections of nodes (topology) and the placement (shaping) of nodes in a transmission tower. The structural grammar constrained the design modification by defining relational rules that linked the section variables, the sizing variable and the shape variables. Making a change at one point in the structure limited the possible changes for the following points.

Another novelty was seeding the algorithm with an existing design tower layout rather than generating conceptual designs from scratch. Starting with a viable design meant that any improvement to the initial design could provide an applicable benefit. Although this work showed the efficacy of grammars and stochastic optimisation it was limited by making small changes to an existing structure instead of exploring a range of different design configurations.

2.1. Grammar representations for design generation

To evolve architecture, a technique is required to automatically generate shapes. EAs traditionally encode individuals in the population using a fixed length string of digits called a genome. The genome by itself is meaningless and so it is mapped using a function to generate useful output. It is the mapping function that limits the representation. EAs have shown their ability to optimise constrained design problems but have scaling issues when applied to more open ended design problems. Normally the function maps directly to output. An example of such a direct mapping is a voxel (volumetric pixel) representation. A fixed length binary string is used to represent points in either a two or three dimensional space. An example of such an approach is Zuo et al. [37], where volumetric pixels (voxels) were added and removed from a truss structure to maximise the stiffness of a Michell truss.

A direct mapping means that the complexity of the design is proportional to the length of the genome. Anything more complicated requires the genome to be extended and so the search space increases. A direct mapping is not sufficient to generate complex designs. Indirect mapping procedures, referred to as embryogenies by Bentley [2], treat the genome as a set of "growing instructions" that define components and how they change during the mapping process. Bentley [2] showed that indirect mappings reduced the search space, provided better enumeration of the search space and produced more complex solutions. Grammars provide a method for explicitly defining an indirect mapping. The next section discusses previous approaches to pylon optimisation and how indirect mappings were used to generate complex designs.

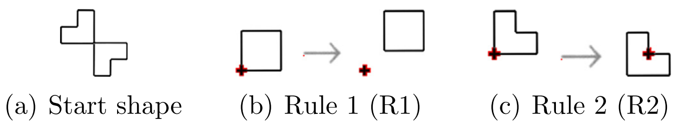


Fig. 1. The start shape and transition grammar defined for a shape grammar.

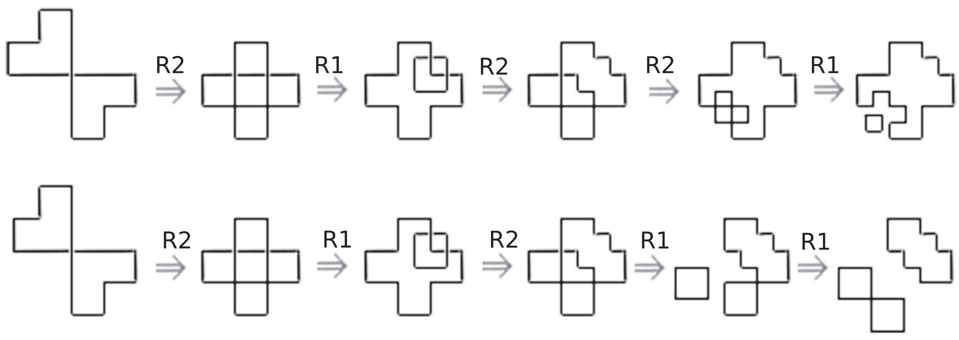


Fig. 2. Two sample derivations using the rules defined in Fig. 1.

A grammar explicitly defines an indirect mapping procedure in a form that can be easily adapted. A grammar is a set of structural rules that define the composition of sentences for a given language. Grammatical rules provide a concise way of restricting a language while allowing endless variation through recursive application of the rules. Normally grammars are used to ensure the validity of a sentence but grammars can also be used to generate sentences. The grammatical rules are sequentially applied to transform an initial start symbol until it creates a sentence. This generative process can be combined with an EA, where the genome is used to select rules from the grammar to apply to the start symbol. This allows the genome to generate an output sentence from the grammar.

The ability of a grammar to constrain the search space while allowing for variation has led to its application in shape generation. The original form of shape grammars was developed by Stiny [32]. A starting shape and grammar of transitions are defined, as shown in Fig. 1. The transformations are iteratively applied to the start shape to create sub-shapes. The sub-shapes are then mapped for an arbitrary number of iterations to generate a design. Two example mappings are shown in Fig. 2. Sub-shapes must be extracted and identified for the mapping to work and accordingly it is normally implemented by hand. Although humans can easily complete this task, sub-shape recognition for an arbitrary number of open items is an NP-complete problem [36]. Despite this issue, several successful automatic approaches have been implemented for design generation. Gero and Louis [12] evolved pareto optimal beam designs using a shape grammar and genetic algorithm. Interestingly they then evolved the grammatical rules themselves and showed that it enabled greater pareto optimality to be reached. A detailed list of computer implementations of shape grammars is shown in Gips [13].

An alternative approach that does not require sub-shape recognition is the context free shape grammar (CFSG) as described in McDermott et al. [23]. Context free derivations are based on iterative string matching and rewriting which, while difficult for a human, are easily accomplished by a computer. The ease and speed at which context free grammars are used by computers means that they have been used to describe the syntax of nearly all programming languages developed from the Algol project [1]. CFGs are a naturally amenable form for a computer to process. Combining the abstraction and automation of context free grammars with shape functions provides a powerful methodology for generating designs. The implementation used in this work generate architectural designs is based on CFGs. How this is accomplished is shown in the next section (see Fig. 3).

3. Grammatical evolution

Grammatical evolution [28,10] is a grammar based form of genetic programming [25]. GE accomplishes the mapping process by combining a context free grammar (CFG) with a rule selection mechanism implemented using a genetic algorithm. The integer codons decide which production is chosen from the CFG by computing the total number of rule productions and using that number to calculate the modulus of the codon value. This can be represented with the following formula:

$$Rule\ (idx) = Codon\ Value\ \%\ Num.\ Productions \tag{1}$$

The grammar rules are applied to the left most non-terminal with the respective codon choosing the production from the grammar. The mapping process is repeated until all non-terminal rules have been converted into terminal rules in the grammar. By separating the grammar from the GA it allows traditional GA operators to be applied to any grammar based

```
<expr> ::= <op><expr><expr>     (0)
         | <var>                (1)

<op>   ::= +                    (0)
         | -                    (1)
         | *                    (2)
         | /                    (3)

<var>  ::= x                    (0)
         | y                    (1)
         | z                    (2)
```

Fig. 3. A simple context free grammar.

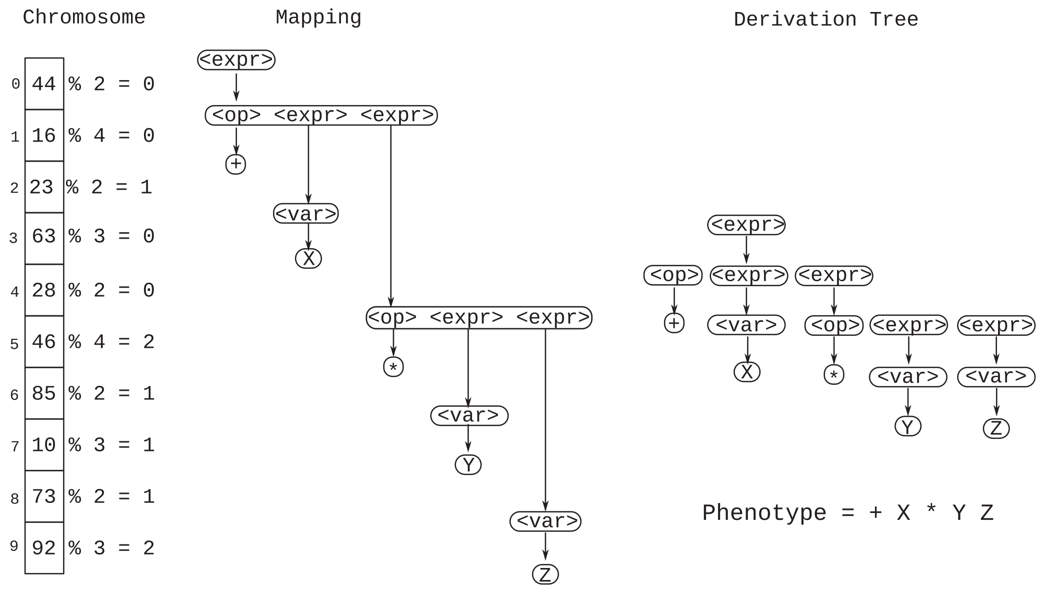


Fig. 4. A sample derivation of a context free grammar using the mod rule.

representations. The separation also mimics the genotype to phenotype mapping as seen in nature, where the integer string is equivalent to the DNA and the output program is the phenotype. A sample derivation is shown in Fig. 4. The terminals at the leaf nodes of the derivation tree create a string from the grammar which in turn represents the program.

The example given in this section gives an overview of how a simple mathematical grammar can be used to generate equations using an integer string. The next section shows how shape generating functions can be used by a grammar to generate designs.

4. Grammatical decomposition

Context free shape grammars enable design decomposition from both the top down and bottom up. Abstracting the design into different grammatical modules allows the grammar to be limited to valid configurations. Although a grammar ensures syntactic correctness it does not prevent the generation of meaningless sentences, e.g., “colourless green ideas sleep furiously”. In this work, grammars ensure that every node in a design is connected but it does not imply that the structure is necessarily suitable for supporting loads. It must also be stated that the grammar only generates a subset of the set of valid designs, which is infinite.

A context free grammar is composed of non-terminal symbols, denoted by angle brackets, and terminal symbols. Non-terminal symbols are abstractions and must be converted to terminal symbols using the rules of the language. Terminals are literal characters in the language that cannot be broken down into smaller parts by the rules of a grammar.

```
<bridge> ::= <walkway><handrail><support>
<walkway> ::= <straight> | <stepped> | <arch>
```

Fig. 5. Using non-terminal abstraction to decompose a bridge.

```
<bezierHandrail> ::= x = draw_points(0,10)
                     y = bezier_curve(5,5,5)
                     map(x,y)
```

Fig. 6. A handrail defined by combining two shape generating functions. The non-terminal rules specify combinations of terminal rules to generate the final design.

```
<bezierHandrail> ::= x = draw_points(0,10)
                     y = bezier_curve(<var>,<var>,<var>)
                     map(x,y)
<var> ::= 1|2|3|4|5|6|7|8|9|10
```

Fig. 7. Variations in the shape rule are introduced by removing hard coded values and replacing them with a var non-terminal.

```
<handrail> ::=<bezier>|<sin>|<random>|<triangle>|<polygon>
```

Fig. 8. Once a shape generating rule is correctly defined it can used as a production choice for a top-down rule.

The ability to abstract through non-terminals allows us to outline the design without specifying the details. Conversely terminal rules allow for standalone components to be constructed before the overall design has been completed. An example of such a decomposition for a bridge design is given below. Initially a bridge is broken down into its salient features. Features can be added or removed from the grammar at any point. The features can then be broken down into their constituent parts. An example is given in Fig. 5.

The terminal rules are developed separately from the non-terminal rules. The terminal rules are shape generating functions, which are then combined using non-terminal rules to generate a final design. An example of a terminal rule is given in Fig. 6, where a Bezier curve with *fixed values* is mapped to points on a line. Any geometric function such as the equation for a line, a polynomial or a plane can be used as a shape generating function.

The fixed values may now be abstracted to allow variations in the design as shown in Fig. 7. If the variations generate invalid configurations the design space may also be restricted at this point. Once the terminal variations have been successfully defined it can be combined with other shape terminals that then map to a non-terminal rule, as shown in Fig. 8.

The ability of shape grammars to allow for quick and parsimonious design definition has led to its application in different design areas such as smoking shelters [27], bridges [5,23], trusses [11], and interactive art generators [7]. The next section describes techniques that have been learned from previous applications to increase the complexity of the generated designs.

5. Grammatical techniques for design complexity

If a representation is to be capable of generating complex designs it must exhibit modularity, regularity and hierarchy [17]. Although context free grammars have an inherent modularity (terminal rules) and hierarchy (non-terminal to terminal mapping), additional grammatical techniques have shown themselves useful when applied to design generation.

5.1. Anonymous functions

Modules enable a design to generate complexity through reuse. Anonymous functions, also known as lambda expressions, give the grammar the ability to define new shape functions “on the fly”. Anonymous functions can be used to wrap combinations of existing shape functions to create new functional output. Anonymous functions also allow for currying, where a pre-existing function is called except some of the arguments to that function are fixed. Wrapping an existing function in an anonymous function allows for a level of hierarchy in the structure of the program.

5.2. Higher order functions

In an effort to create more coherent designs and also to avoid problems of infinite recursion in a grammar, higher order functions (HOF) were introduced. In this work, passing a set of points through different higher order functions is the key to

```
#path generating function
circle_path(scalar, radius, midpoint, axis):
    When given a scalar value it will return a 3D coordinate on a circular path.
    The axis is the plane that remains constant for the circle coordinates

#connecting function
connect_3(starting_point, axis):
    Generates two additional points equidistant from original and then connects them.
    The axis is the plane that remains constant for the circle coordinates

#connecting function
drop_perpendicular(point):
    When given a point it will add a point where the z-plane is zero and connect the points

#higher order function
map(scalar_point_function, scalar_list):
    Uses a list of scalar values and a function to create a list of points on a path

#higher order function
function_map(function_list, point_list):
    Higher order function that applies each function in the function list to the point list
```

Fig. 9. A simplified function set from the higher order function grammar. It consists of a path generating function, two connecting functions and two higher-order functions.

```
function_map([lambda x: connect3(x ,'y'),
               lambda x: drop_perpendicular(x)],
               map(lambda t : circle_path(t, 50, (0,0,0), 'z'),
                   [0,1,2,3,4,5])
```

Fig. 10. An example program generated by the grammar. `function_map()` calls `drop_perpendicular()` and `connect3()` on the point list generated by `map()` and `circle_path()`.

generating fully connected designs. Recursion generates complexity and similarity by repeatedly applying a function on a particular shape. McDermott et al. [24] and Yu [35] showed that grammars were capable of encoding such qualities through the use of higher-order functions (HOF).

Higher-order programming requires that functions can be passed to other functions as arguments (first-class functions) and that anonymous functions can be created. Limited recursion is accomplished by using a variable to specify the maximum recursion depth. As the function is recursively called, the value of the depth variable is decreased. Recursion stops when the depth variable reaches zero.

5.3. Example grammar implementation

A simplified example of a grammar using anonymous functions, higher order functions and function currying is given in Fig. 9. The geometric objects are created by combining path functions and connecting functions. Path functions are given scalar values and return points on a path. Connecting functions are given points and then generate additional points and connect them together. An example of a program is shown in Fig. 10. The program uses the map function in conjunction with a scalar list to generate a list of points that follow a circular path Fig. 11(a). Two of the arguments to `circle_path()` have been fixed or "curried", the axis is set to z-axis and the circle radius is set to 50. Function map the applies the functions `Connect3()` and `drop_perpendicular()` to the set of points generated by `circle_path()`. `Connect3()` adds triangles to the points, as shown in Fig. 11(b), and has the axis argument curried so that it is set to the y-axis for all of the triangles. Finally `drop_perpendicular()` add a vertical line as shown in Fig. 11(c). Now that the grammar for generating shapes has been introduced, a description of the evolutionary algorithm used to optimise the grammars is given.

5.4. Simplified pylon grammar

A simplified version of the grammar used in this work is shown in Fig. 12. The maintenance requirement for the pylon means that only symmetrical designs need to be considered (see Section 7). The symmetrical constraint means that a full tower can be generated by creating one quarter of it and mirroring the result. The constants (maximum height/width) and line points (where the lines meet the tower) are initially defined. Once the constraints are in place, the base and the

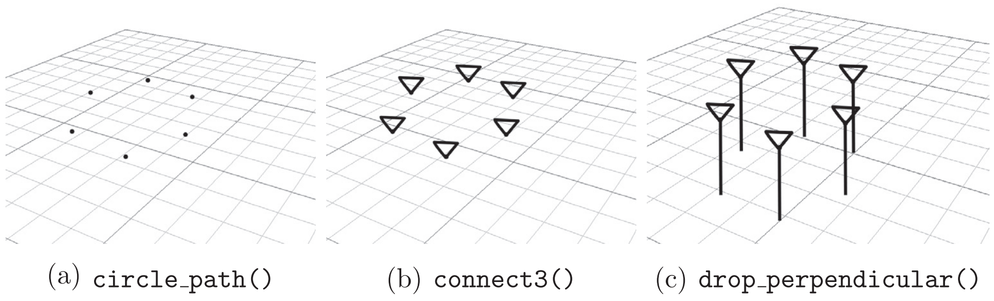


Fig. 11. The output design generated by the program example.

```
#non-terminal rules
program ::= <constants><pylon><rotate>
<pylon> ::= <lines><base><section1><section2><section3>
<base> :: label="base",<create_legs>
<create_legs> ::= <sierpinksi> | <crosshatch> | <anglehatch>
<section1> ::= label = "section1", <section>
<section2> ::= label = "section2", <section>
<section3> ::= label = "section3", <section> <create_arms>
<create_arms> ::= <sierpinksi> | <crosshatch> | <anglehatch>
#terminal rules
<constants> ::= "Max Height, Max Width, etc"
<lines> ::= "defines the positions of the lines and insulators"
<section> ::= "generates a lattice structure"
<rotate> ::= "mirror through y-axis and then rotate 180 degrees"
<crosshatch> ::= "generate frame using square lattice"
<anglehatch> ::= "generate frame using angled lattice"
<sierpinski> ::= "lattice based on sierpinski gasket fractals"
```

Fig. 12. A simplified version of the pylon grammar. The textual descriptions for the terminal rules give a brief overview of what each terminal rule generates. The full grammar is available online at [4].

three sections that make up the body of the structure are defined. The line points are then connected to their respective sections and finally the structure is mirrored to create the tower.

Structural analysis requires a mechanism for fixing points and applying loads before the stresses can be calculated. This process can be automated by adding labels to an existing grammar. Different components are labelled depending on the non-terminal rule that created them. As the non-terminal rules are expanded sequentially, a label variable is set during expansion. Any nodes created during a non-terminal expansion are labelled with this value. An example of the label variable being set is shown in Fig. 12. Once the structure is complete the nodes are iterated over and the loading value is set by the node label. An example of different structural loading can be seen in Fig. 13. This section explained how grammars can be used to generate shapes and gave implementation details of how such a grammar can be used for generating pylons. The next section describes structural analysis techniques and shows how they can be used to evolve designs.

6. Structural analysis

Structural engineering is the field of analysing and constructing designs that are capable of handling loads [14]. The goal of structural engineering is to maximise resistance to load while reducing material usage and cost. Computer modelling provides a methodology that allows designs to be easily and quickly analysed in a virtual environment. Simulated forces are applied to a structure and the resultant loads, stresses and deformations can be computed. An example of this is shown in Fig. 14. The forces on a structure are traditionally calculated using partial differential equations (PDE). While PDEs can be calculated using a computer, a method based on discretisation for modelling stresses is the preferred approach. Finite Element Analysis (FEA) [14] simplifies this process into a more computationally friendly form by converting PDEs into a set of ordinary differential equations (ODE). This work uses SLFFEA [22], an open source FEA library to perform structural analysis.

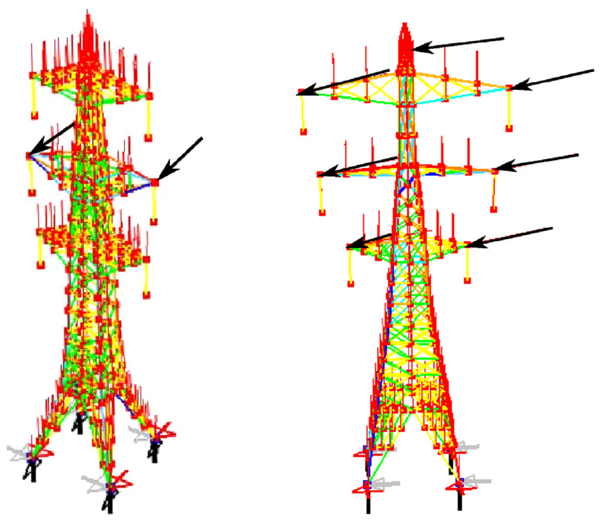

(a) Cable break loading. (b) Extreme wind loading.

Fig. 13. The result of semantic labelling. The bases of the design are fixed and the self weights shown in red. The loading constraints for specific conditions are highlighted by the black arrows. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

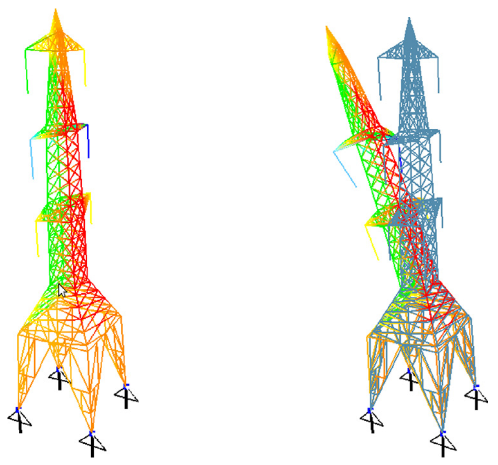

(a) High load areas shown in red. (b) Scale of displacement increased to highlight deformation.

Fig. 14. Analysing the structural loading of an electricity pylon when a cable break condition is simulated.

6.1. Types of structural analysis

Structural optimisation is classified into three categories [18].

- Topology: The overall layout of the system.
- Shape: The optimal contour for a fixed topology.
- Sizing: The size and dimensions of the components.

The categories relate to the three major stages of the engineering design process, i.e., the overall layout is chosen at the conceptual stage, the shape of the structure is optimised during the embodiment stage and the final sizing is optimised during the detail design stage [20].

The categories form a hierarchy for the structural engineer. A topology must be fixed before shape optimisation can be carried out. Accordingly, the components must be chosen before their sizing can be optimised. This work simultaneously optimises the topology and shape of the structure. Although sizing optimisation is possible using a grammatical approach, this aspect is ignored as our intention is to build the structure with equal angle beams traditionally used in pylon construction.

6.2. Structural analysis and evolutionary algorithms

The extensive use of computer-based FEA led to its application as an objective and automatic fitness function for evolutionary algorithms. Accordingly, there has been a large amount of work in this area [20]. The computational cost of structural analysis meant that early papers focused on greatly simplified structures, such as small scale two dimensional trusses [15]. As computational power increased, so did the scope of the applications. Structures such as bridges [33], electricity pylons [31], and even entire buildings [19] have been optimised using EAs.

6.3. Incorporating structural analysis into a fitness function

Structural analysis provides several pieces of information that can be used as fitness values. When the tension or compression on a beam exceeds a certain value the beam breaks. While this is a useful method for eliminating invalid structures, the "all or nothing" fitness value discards much useful information [6]. Summing the stress on the structure provides a fitness value for the structure but ignores localised areas of high stress. Our initial approach recorded the maximum stress point in a structure as a fitness function but this produced poor individuals that did not meet the intended design goals. Instead a maximum stress level was specified and any individual that did not meet this criteria was given a default fitness that removed it from the population.

The approach used in this work records the displacement of a structure to evaluate fitness. This is a measure of how far the structure moves when a load is placed on it. While this approach generates strong structures, a single fitness value alone does not evolve optimal structures. An EA can generate a strong but non-optimal structure by simply adding more beams to increase load handling abilities. To counteract this tendency, a second conflicting objective is introduced, weight.

When there is a conflict between design objectives, it means that a trade off is required between them. An example of such a conflict during the design process would be minimising the amount of material used to build a structure while trying to reduce the load on the structure's components. When optimising designs with multiple constraints two approaches can be taken. The constraints can be weighted and summed, thus reducing a multi objective problem back to a single objective problem. This simplifies the problem but is still inferior to a truly multi-objective approach. The correct weighting must be manually chosen and even if the correct weighting is found, only individuals that maximise this weighting will be produced.

The alternative approach is that there is no globally optimal solution. Instead the solutions can be placed on a "front" composed of the non-dominated solutions in the population [34]. The set of non-dominated (pareto optimal) solutions are solutions that are better than the rest of the population for at least a single constraint and at least equivalent for all other constraints. The set of non-dominated solutions form what is called the Pareto front. Substituting one solution for another on the Pareto front will always sacrifice quality for at least one constraint, while improving at least one other.

The multi-objective selection operator from the Non-Sorting Genetic Algorithm (NSGA2) [9] was used as the fitness function in this work. The NSGA2 ranks each individual based on the number of solutions in the population that dominate them. The non-dominated solutions are the highest rank and worst solutions are the lowest rank. The non-dominated solutions are solutions that are better than the rest of the population for at least a single constraint and at least equivalent for all other constraints. This can be stated mathematically as: $\mathbf{f}$ is the set of fitness functions: $\mathbf{f} = [f_o, \ldots, f_n]$ such that $\forall f \in \mathbf{f}$ where $f^{non-dom} \leqslant f^{dom}$ and $\exists f \in \mathbf{f}$ where $f^{non-dom} < f^{dom}$.

The new population is created by successively adding the individuals in each rank, starting with the highest rank, until the population reaches the correct size. Displacement and weight were chosen as the two objective constraints to be minimised. The NSGA2 algorithm generates a population of pareto equivalent individuals so there is no optimal individual. An average of the fitness value is used instead to observe overall progress of the algorithm. The formula used for calculating displacement is given in Eq. (2) where n is each node in the structure, (x_1, y_1, z_1) is the original position and (x_2, y_2, z_2) is the position after loading.

$$d = \sum_{n \in N} \sqrt{(nx_1 - nx_2)^2 + (ny_1 - ny_2)^2 + (nz_1 - nz_2)^2} \qquad (2)$$

The formula for computing the weight is given in Eq. (3), where $B = \{Beams\}$, $|b| =$ is the Length of beam and κ is the area of the cross-section.

$$w = \sum_{b \in B} |b| \cdot \kappa \qquad (3)$$

There is an additional check to ensure that no structures that cannot support the loading are in the population. Each beam is checked to ensure that it is not stressed beyond a predefined value. If it exceeds this value a default fitness is assigned that removes it from the population.

7. Pylon design competition

The RIBA pylon design competition [30] was chosen as a suitable challenge for a grammar based approach. The competition invited engineers, designers and architects to rethink the traditional design of the electricity pylon. The UK are targeting an 80% reduction in carbon emissions by 2050 and substantial change to the electricity infrastructure is part of that plan. Although the focus was primarily to develop new conceptual designs, there were a number of real world constraints specified.

The existing standard pylon height requirement is based on a standard span of 360 m in flat terrain, having a minimum conductor to ground clearance of 8.1 m at maximum operating temperature. Each pylon leg is rooted in concrete foundations, so the base points are assumed to be fixed. Most importantly, maintenance must be possible on one side of the pylon at a time in other words, with one single circuit outage while the other circuit is live. The result of this constraint is that the structure must be symmetrical.

Our pylon design is based on the traditional steel lattice structure composed of angled steel beams. The lattice structure is strong and efficient and reduces the surface area available for the wind to act upon. Angled steel is a cheap, plentiful and well understood material. Lattice structures are effectively transparent at distances of 500 m or more [16]. Novel materials such as carbon fibre were considered as they remove the need for insulators but long term resilience is not yet known. There is also a risk to the structure from lightning strikes [26].

7.1. Loading constraints

The intention of this work is to examine if load cases taken directly from a design specification are capable of generating usable designs. The pylon structure must not only support the weight and tension of conducting cable and insulator strings but must also be capable of withstanding gale force winds, ice, snow, floods and lightning strikes. The designs were tested against three extreme load cases, extreme wind, extreme ice, and a combination of wind and ice. Two loading constraints were chosen to examine if structures can be evolved for environmental extremes. A combination of those loading constraints is then used to examine if a structure can be optimised to handle multiple environmental conditions simultaneously to an adequate degree.

The load cases are illustrated in Fig. 15. All loads are conductor loads. They exclude structure weight and wind loading, which are added separately. All loads are in kilonewtons (kN). The extreme wind loading states that the wind is applied to the body of the structure in a transverse direction. The mean hourly wind speed is 35 m/s at 10 m above ground level. Allowances are made for the variation of wind speed with structure height and the total wind load on the structure will allow for gust effects in addition to mean wind. A loading tree for the extreme wind case is shown in Fig. 15(a).

The extreme ice loading assumes an ice buildup on the power lines. Although ice buildup is not initially problematic, the increase in surface area on the cable means that the ice buildup gets progressively worse. The load case states that the ice buildup has a radial ice thickness of 65 mm. A loading tree for the extreme ice case is shown in Fig. 15(b).

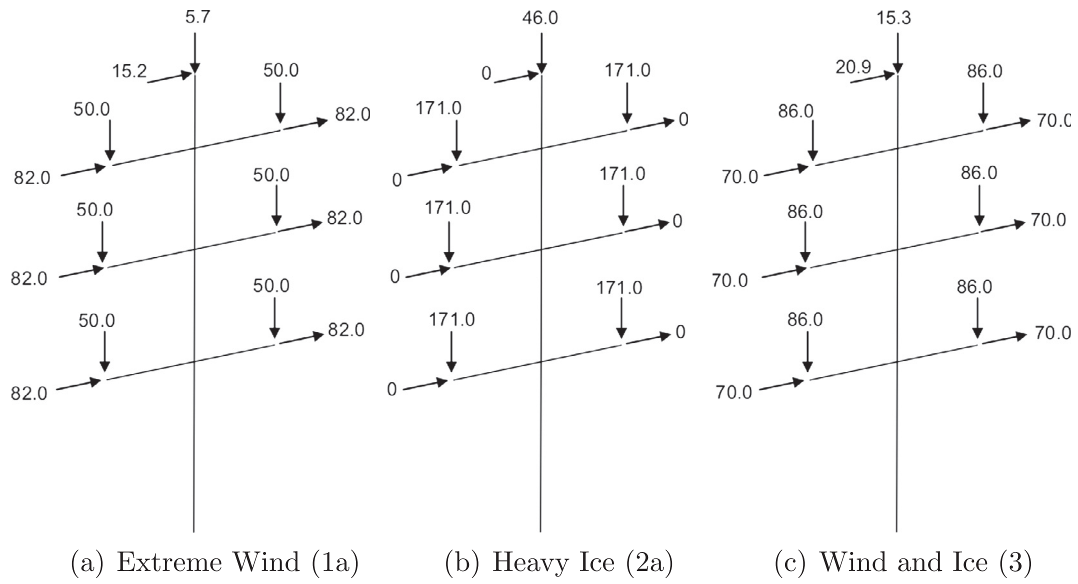


Fig. 15. The loading trees for each test case.

The extreme wind and ice loading combines lesser amounts of the previous load cases. The wind speed is assumed to be 30 m/s and the radial ice thickness is 15 mm. A loading tree for the extreme wind and ice case is shown in Fig. 15(c). Although the separate ice and wind loading are less the combination amounts to a difficult test case for the structure. The design must be capable of handling both forces adequately. This test case examines the possibility of optimising a structure for combinations of weather conditions that the structure might face in a particular location.

8. Optimisation based on real world constraints

Grammars are capable of limiting the design space while still allowing design variations to be explored. Loading constraints from a design specification document can be used to create a fitness function for an evolutionary algorithm. We now examine if the grammar representation was capable of being optimised through evolution, i.e., that genetic information is being transferred in a way that leads to better solutions. To investigate if there was significant improvement when evolving the grammatical representation, the algorithm was run thirty times (population size 100 for 50 generations). The results were then compared with the best of 150,000 randomly generated individuals. This is equivalent to the number of evaluated individuals in the evolutionary run except that no genetic information was transferred between individuals. This was repeated for each loading case.

The experiment was carried out using the pylon grammar described in Section 5. The experimental settings are shown in Table 1. It should be noted that mutation in GE is applied on a per codon basis, i.e., The mutation operator is applied to every integer and mutation occurs with a given probability. Single point crossover was used for this experiment. The same index point is chosen on both parents' genome and then everything after that point is swapped to create two new individuals.

8.1. Results

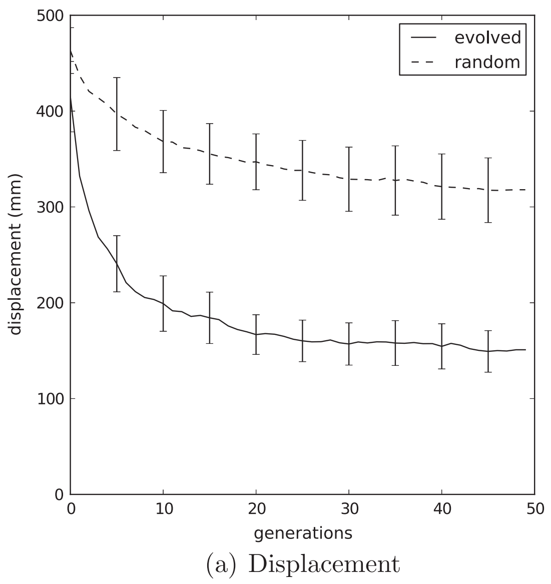
The population fitness results for the three test cases are shown in Figs. 16, 18 and 20. There is no "best" individual when using a Pareto optimisation and the size of respective fronts can vary when using the NSGA2 algorithm. Accordingly, the results show the average population fitness and standard deviation of the thirty runs for each generation. All results are graphed using the same scale on the y-axis. The random results are shown as a dotted line. Although no genetic information was transferred between the random individuals there was still a selection pressure produced by the NSGA2 fitness function. The parent and child population are compared at each generation and the best of both is kept. The result of keeping the best 100 individuals is that the average population fitness slowly improves over time. Although this continuous improvement looks like hill-climbing, it is a result of random sampling combined with elitism.

Fig. 16 shows the result for an extreme wind loading. There is significantly more displacement for the wind loading cases as the force is applied laterally rather than vertically and the fixed points are at the base of the structure, resulting in a cantilever style action. The results show that evolving the representation generates significantly more improvement for displacement on all of the load cases even though the initial population has poor fitness. There is a 55% reduction in the average displacement for an evolved population compared to the 22.3% reduction in average fitness for the randomly generated population. There is an increase in overall weight for the designs compared to randomly generated solutions this was to be expected, stronger designs generally require more material. The lack of improvement on both the random and evolved results indicates that the grammar representation has constrained the weight search space to a degree where only limited change to the weight of the structure is possible. Although this is an unexpected restriction of the grammar, there is the benefit that the weight does not exponentially increase for stronger individuals during the course of the run.

Example individuals from the Pareto front are shown in Fig. 17. The sample individuals have developed similar features to account for the extreme wind loading. They have increased their base and core structure width, this allows it to distribute the loading over a greater cross-sectional area. The structure has increased its moment of inertia, thus giving it a greater resistance to lateral loading.

Table 1
Experimental settings.

Parameter	Value
Population size	100
Generations	50
No. of runs	30
Mutation rate	1.5%
Mutation operator	Integer
Crossover rate	70%
Crossover operator	Single point
Selection scheme	NSGA2
Replacement scheme	NSGA2
Initialiser	Random
Random number generator	Mersenne twister



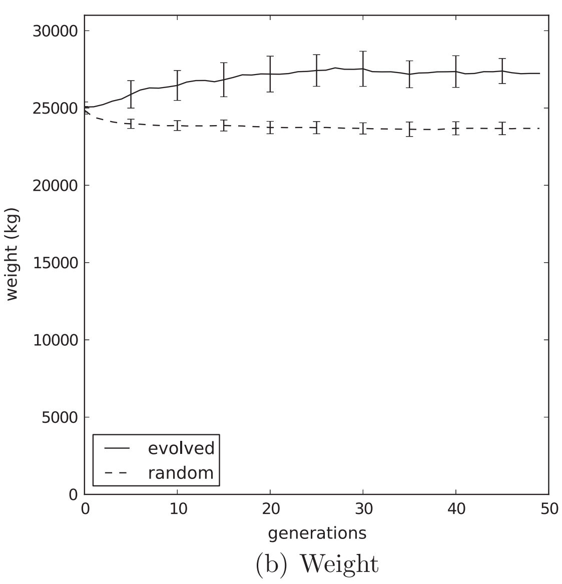


Fig. 16. Results from extreme wind optimisation.

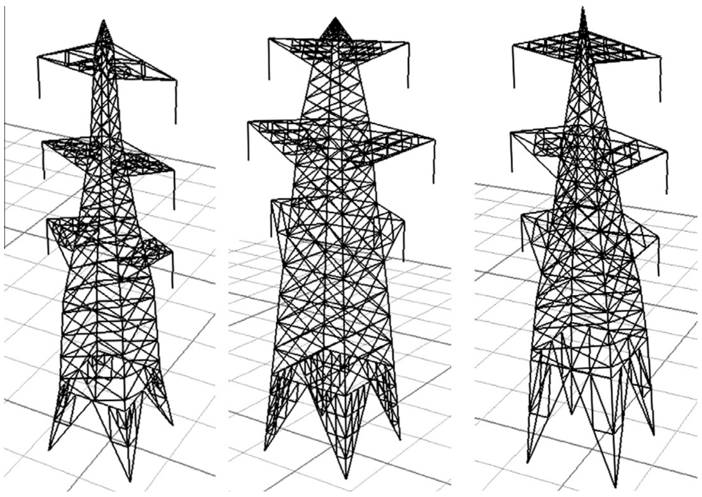
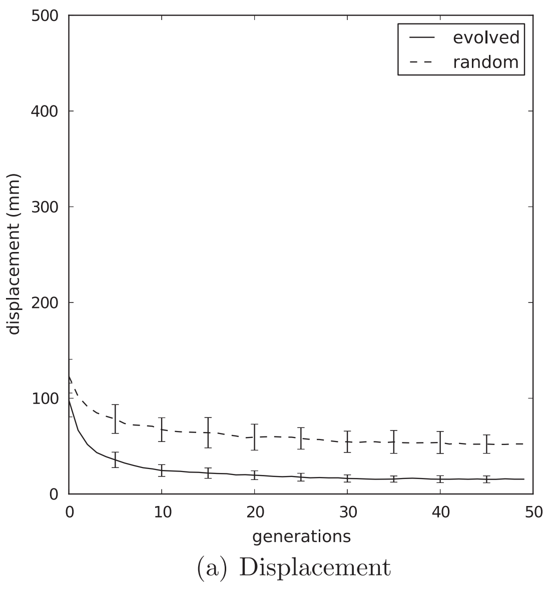


Fig. 17. Sample individuals for extreme wind optimisation.

Fig. 18 shows the results for the heavy ice loading. The amount of displacement experienced by these structures is lower as the loading is transmitted directly down the core of the structure. There is still a significant reduction in displacement when compared against randomly generated designs. The evolved population reduced the average displacement by 85% compared to the 33% displacement reduction in the randomly generated population. The reduction in displacement once again results in an increase in weight.

Sample individuals for the heavy ice loading are shown in Fig. 19. The ice loading builds up predominantly on the cables rather than the main structure so most of the additional force is transmitted through the arms where the cables are suspended from. As the force is transmitted directly downwards through the arms, the evolutionary algorithm narrowed the internal core structure, as there were no lateral loads present. The sample individuals also show shorter arms that have been reinforced, this reduces the moment applied by the arms to the main structure. As the fixed points cannot be changed the structures broaden at the base.

The results for the wind and ice loading are shown in Fig. 20. Even though there is a lesser combination of the extreme ice and wind loading conditions, the initial population starts with a poor fitness that is comparable to the extreme wind loading. The displacement improves significantly during the course of the run and evolution outperforms randomly generated solutions but this is once again offset by an increase in the overall weight. The average displacement for the evolved population is reduced by 59% compared with an 18.7% reduction for the randomly generated population.



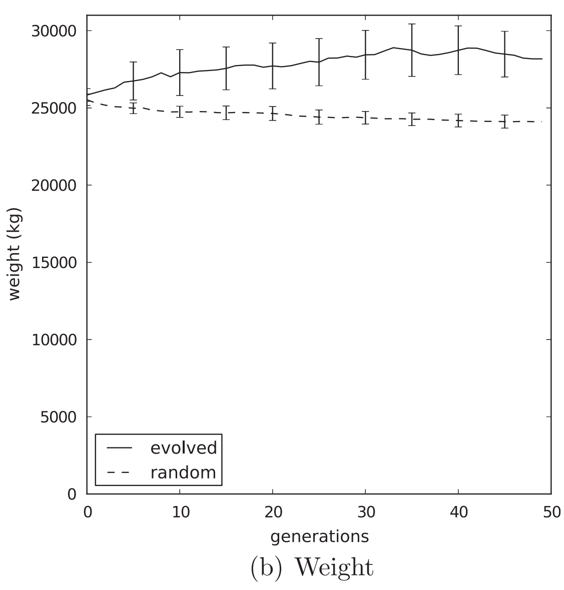


Fig. 18. Results from ice loading optimisation.

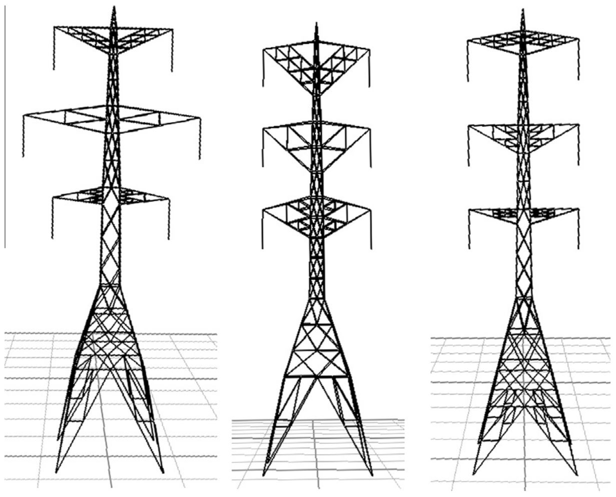
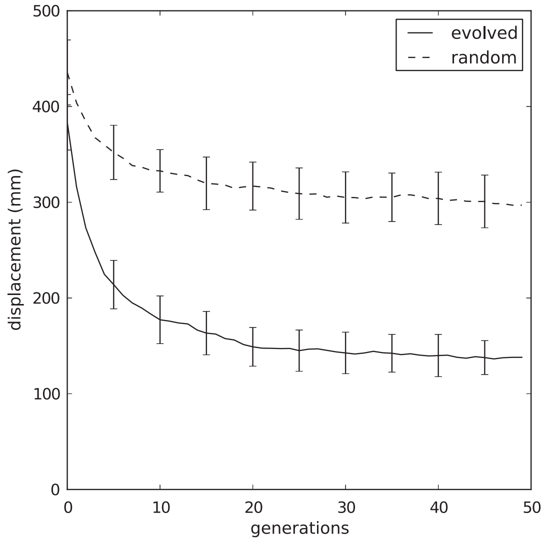


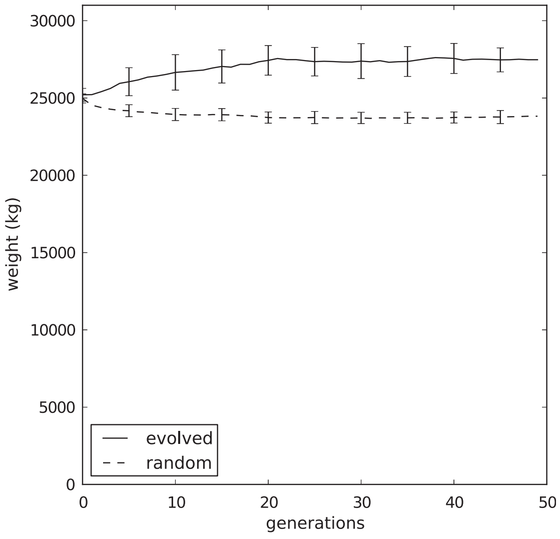
Fig. 19. Sample individuals for ice loading optimisation.

Sample individuals for the combined wind and ice loading are shown in Fig. 21. These designs show a compromise keeping the narrow core structure for the upper part but widening the base to reduce wind deflection. The shorter arms have been chosen to reduce the moment applied due to the ice loading on the main structure. This amalgamation of design features shows that different design constraints can be combined to generate results that satisfy both conditions.

Overall the results show that pylon designs generated by a context free shape grammar are capable of being significantly optimised and that the loading constraints described in Section 7 are capable of driving evolutionary search. The displacement was reduced by at least 50% in all of the loading cases and in one case by 85%. The evolved solutions outperformed the randomly generated solutions, meaning that useful genetic information was being passed between individuals despite the complex mapping procedure used by context free shape grammars. One of the unforeseen results of constraining the search space to valid pylon designs was that there was only a small amount of variation in the weight of the structures. Although this limited the ability of the NSGA2 algorithm to optimise weight, lighter individuals were still preferentially selected due to their position on the pareto front. Weight did not improve but neither did it increase exponentially. A trade off between weight and strength is to be expected when optimising for loading conditions. The multi-objective fitness function meant that it only added weight if the solution was non-dominated rather than adding weight even if it only had minimal loading benefit. The designs selected from the pareto front showed that it evolved specific features for the extreme loading test cases.



(a) Displacement



(b) Weight

Fig. 20. Results from wind and ice optimisation.

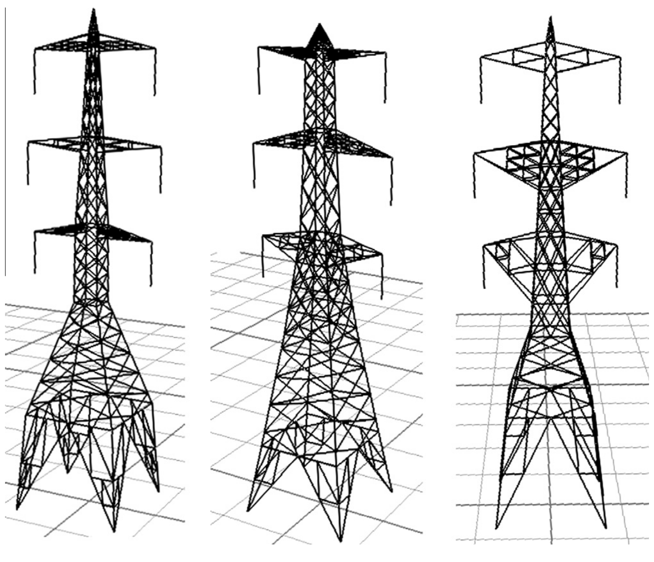


Fig. 21. Sample individuals for wind and ice loading optimisation.

It also showed that different loading constraints could be combined and optimised simultaneously. The resulting designs had a combination of the features seen in the extreme load cases.

9. Conclusion

This work set out to explore the use of grammar representations for large scale structural optimisation. The approaches for decomposing an architectural design using a grammar were described and grammatical techniques for generating large scale structures and design complexity were introduced. A grammar was then devised to generate electricity pylon structures. Unlike previous approaches which optimised parts of the pylon structure, the pylon grammar was used explore different configurations and optimise several components of a structure simultaneously. The structure was evaluated based on displacement and overall weight using constraints taken directly from an electricity pylon design competition.

The experiments clearly show that shape generating grammar representations are capable of being optimised and that genetic information is being retained though the grammar mapping. The population fitness showed a significant reduction in displacement for all of the loading cases. Although there was an increase in structural weight, this was to be expected as a trade off for greater strength. the experiments also showed that real world structural constraints can be used to drive evolution and that designs could be evolved for specific extreme load cases. More importantly, combining the constraints resulted in structures that had features of the designs optimised for individual cases, thus creating structures capable of handling multiple loading conditions. As grammars allow for automatic force application and the loading constraints can be taken directly from a design specification document, this approach can be easily adapted to optimise bespoke structures for loading conditions specific to certain locations.

10. Future work

The finite element analysis described in this work was used to analyse loading conditions but it can also be applied to computational fluid dynamics. It is our intention to further develop grammatical approaches to design decomposition and apply it to more complex structures such as wind turbine blades and aircraft wing design.

Acknowledgments

We would like to thank Science Foundation Ireland, The Financial Mathematics Computation Cluster and Andrea McMahon for her help during this project. This work was funded by the SFI Grants 08/RFP/CMS1115, 08/IN.1/I1868 and 08/SRC/FM1389.

References

[1] J.W. Backus, F.L. Bauer, J. Green, C. Katz, J. McCarthy, P. Naur, A.J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, et al, Revised report on the algorithmic language ALGOL 60, Comput. J. 5 (4) (1963) 349–367.
[2] Peter J. Bentley, Exploring component-based representations-the secret of creativity by evolution?, in: Ian C. Parmee (Ed.), Proceedings of the Fourth International Conference on Adaptive Computing in Design and Manufacture (ACDM 2000), University of Plymouth, 2000, pp. 161–172.
[3] O. Bohnenberger, J. Hesser, R. Manner, Automatic design of truss structures using evolutionary algorithms, IEEE International Conference on Evolutionary Computation, 1995, vol. 1, IEEE, 1995, p. 143.
[4] Jonathan Byrne, Architype source code, 2013. <https://github.com/squeakus/architype/>.
[5] Jonathan Byrne, Michael Fenton, Erik Hemberg, James McDermott, Michael O'Neill, Elizabeth Shotton, Ciaran Nally, Combining structural analysis and multi-objective criteria for evolutionary architectural design, in: Applications of Evolutionary Computing, EvoApplications 2011: EvoCOMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, EvoTRANSLOG, LNCS, vol. 6625, Springer Verlag, Turin, Italy, 2011, pp. 200–209.
[6] Carlos A. Coello Coello, A survey of constraint handling techniques used with evolutionary algorithms, Lania-RI-99-04, Laboratorio Nacional de Informática Avanzada, 1999.
[7] Chris Coyne, Context free art, 2010. <http://www.contextfreeart.org/>.
[8] K. Deb, S. Gulati, Design of truss-structures for minimum weight using genetic algorithms, Finite Elem. Anal. Des. 37 (5) (2001) 447–465.
[9] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evolution. Comput. 6 (2) (2002) 182–197. ISSN: 1089-778X.
[10] Ian Dempsey, Michael O'Neill, Anthony Brabazon, Foundations in Grammatical Evolution for Dynamic Environments, Springer, 2009.
[11] Michael Fenton, Analysis of timber structures created using a g.e-based architectural design tool, Master's thesis, University College Dublin, Ireland, 2010.
[12] John S Gero, Sushil J Louis, Improving pareto optimal designs using genetic algorithms, Comput.-Aid. Civil Infrastruct. Eng. 10 (4) (1995) 239–247.
[13] J. Gips, Computer implementation of shape grammars, in: NSF/MIT Workshop on Shape Computation, Citeseer, 1999.
[14] E.H. Glaylord, C.N. Glaylord, Structural Engineering Handbook, McGraw-Hill, 1979.
[15] J. Weidermann, A. Hoeffler, U. Leysner, Optimization of the layout of trusses combining strategies based on Mitchel's theorem and on biological principles of evolution, in: Proceeding of the 2nd Symposium on Structural Optimisation, Milan, Italy, 1973.
[16] William Holford, The holford rules, The Royal Society of Arts (1959).
[17] G.S. Hornby, Measuring, enabling and comparing modularity, regularity and hierarchy in evolutionary design,;;, in: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, ACM, 2005, pp. 1729–1736.
[18] M.J. Jakiela, C. Chapman, J. Duda, A. Adewuya, K. Saitou, Continuum structural topology design with genetic algorithms, Comput. Methods Appl. Mech. Eng. 186 (2–4) (2000) 339–356.
[19] Rafal Kicinger, Tomasz Arciszewski, Kenneth DeJong, Evolutionary design of steel structures in tall buildings, J. Comput. Civil Eng. 19 (3) (2005) 223–238. <http://link.aip.org/link/?QCP/19/223/1>.
[20] Rafal Kicinger, Tomasz Arciszewski, Kenneth De Jong, Evolutionary computation and structural design: a survey of the state-of-the-art, Comput. Struct. 83 (23–24) (2005) 1943–1978. ISSN: 0045-7949.
[21] B. Lawson, How designers think: the design process demystified, Elsevier/Architectural (2006). ISBN: 9780750660778. <http://books.google.ie/books?id=lPvqZJNAdG8C>.
[22] San Le, San le's free finite element analysis, 2010. <http://slffea.sourceforge.net/>.
[23] James McDermott, Jonathan Byrne, John Mark Swafford, Martin Hemberg, Ciaran McNally, Elizabeth Shotton, Erik Hemberg, Michael Fenton, Michael O'Neill, String-rewriting grammars for evolutionary architectural design, Environ. Plan. B: Plan. Des. 39 (4) (2012) 713–731. <http://www.envplan.com/abstract.cgi?id=b38037>.
[24] James McDermott, Jonathan Byrne, John Mark Swafford, Michael O'Neill, Anthony Brabazon, Higher-order functions in aesthetic EC encodings, in: 2010 IEEE World Congress on Computational Intelligence, IEEE Press, Barcelona, Spain, 2010, pp. 2816–2823.
[25] R.I. McKay, N.X. Hoai, P.A. Whigham, Y. Shan, M. O'Neill, Grammar-based genetic programming: a survey, Genet. Program. Evolvable Machines 11 (3) (2010) 365–396.
[26] Toshio Ogasawara, Yoshiyasu Hirano, Akinori Yoshimura, Coupled thermoelectrical analysis for carbon fiber/epoxy composites exposed to simulated lightning current, Compos. Part A: Appl. Sci. Manuf. 41 (8) (2010) 973–981. ISSN: 1359-835X. <http://www.sciencedirect.com/science/article/pii/S1359835X10001016>.
[27] Michael O'Neill, James McDermott, John Mark Swafford, Jonathan Byrne, Erik Hemberg, Elizabeth Shotton, Ciaran McNally, Anthony Brabazon, Martin Hemberg, Evolutionary design using grammatical evolution and shape grammars: designing a shelter, Int. J. Des. Eng. (2011).
[28] Michael O'Neill, Conor Ryan, Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language, Kluwer Academic Publishers, 2003. ISBN: 1402074441.

[29] I. Rechenberg, Evolution Strategy: Optimization of Technical Systems According to the Principles of Biological Evolution, Frommann-Holzboog, Stuttgart, 1973.
[30] Royal Institute of British Architects, Riba pylon design competition, 2011. <http://www.ribapylondesign.com/>.
[31] K. Shea, I.F.C. Smith, et al, Improving full-scale transmission tower design through topology and shape optimization, J. Struct. Eng. 132 (2006) 781.
[32] G. Stiny, Introduction to shape and shape grammars, Environ. Plan. B 7 (3) (1980) 343–351.
[33] B.H.V. Topping, J.P.B. Leite, Parallel genetic models for structural optimization, Eng. Optim. 31 (1) (1988) 65–99. ISSN: 0305-215X.
[34] David A. van Veldhuizen, Gary B. Lamont, Multiobjective evolutionary algorithms: analyzing the state-of-the-art, Evolution. Comput. 8 (2) (2000) 125–147. <http://www.citeseer.ist.psu.edu/vanveldhuizen00multiobjective.html>.
[35] Tina Yu, Hierarchical processing for evolving recursive and modular programs using higher-order functions and lambda abstraction, Genet. Program. Evolvable Machines 2 (4) (2001) 345–380.
[36] K. Yue, R. Krishnamurti, F. Gobler, Computation-friendly shape grammars, School Architect. (2009).
[37] Z.H. Zuo, Y.M. Xie, X. Huang, Combining genetic algorithms with beso for topology optimization, Struct. Multidisc. Optim. 38 (5) (2009) 511–523.