

This is a repository copy of *EvoDevo: Bioinspired Generative Design via Evolutionary Graph-based Development*.

White Rose Research Online URL for this paper:

<https://eprints.whiterose.ac.uk/id/eprint/229440/>

Version: Published Version

Article:

Tahernezhadjavazm, Fred, Colligan, Andrew, Friel, Imelda et al. (6 more authors) (2025) *EvoDevo: Bioinspired Generative Design via Evolutionary Graph-based Development*. Algorithms. 467. ISSN: 1999-4893

<https://doi.org/10.3390/a18080467>

Reuse

This article is distributed under the terms of the Creative Commons Attribution (CC BY) licence. This licence allows you to distribute, remix, tweak, and build upon the work, even commercially, as long as you credit the authors for the original work. More information and the full terms of the licence here:









<https://creativecommons.org/licenses/>

Takedown

If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

Article

EvoDevo: Bioinspired Generative Design via Evolutionary Graph-Based Development

Farajollah Tahernezhad-Javazm ^{1,†}, Andrew Colligan ^{2,†}, Imelda Friel ², Simon J. Hickinbotham ¹, Paul Goodall ³, Edgar Buchanan ¹, Mark Price ², Trevor Robinson ² and Andy M. Tyrrell ^{1,*}

¹ Intelligent Systems & Robotics Research Group, School of Physics, Engineering and Technology, University of York, York YO10 5DD, UK; f.taher@york.ac.uk (F.T.-J.); simon.hickinbotham@york.ac.uk (S.J.H.); edgar.buchanan@york.ac.uk (E.B.)

² School of Mechanical and Aerospace Engineering, Queen's University Belfast, Belfast BT7 1NN, UK; a.colligan@qub.ac.uk (A.C.); i.friel@qub.ac.uk (I.F.); m.price@qub.ac.uk (M.P.); t.robinson@qub.ac.uk (T.R.)

³ School of Mechanical, Electrical and Manufacturing Engineering, Loughborough University, Loughborough LE11 3TT, UK; p.a.goodall@lboro.ac.uk

* Correspondence: andy.tyrrell@york.ac.uk

† These authors contributed equally to this work.

Abstract

Automated generative design is increasingly used across engineering disciplines to accelerate innovation and reduce costs. Generative design offers the prospect of simplifying manual design tasks by exploring the efficacy of solutions automatically. However, existing generative design frameworks rely heavily on expensive optimisation procedures and often produce customised solutions, lacking reusable generative rules that transfer across different problems. This work presents a bioinspired generative design algorithm utilising the concept of evolutionary development (EvoDevo). This evolves a set of developmental rules that can be applied to different engineering problems to rapidly develop designs without the need to run full optimisation procedures. In this approach, an initial design is decomposed into simple entities called *cells*, which independently control their local growth over a development cycle. In biology, the growth of cells is governed by a gene regulatory network (GRN), but there is no single widely accepted model for this in artificial systems. The GRN responds to the state of the cell induced by external stimuli in its environment, which, in this application, is the loading regime on a bridge truss structure (but can be generalised to any engineering structure). Two GRN models are investigated: graph neural network (GNN) and graph-based Cartesian genetic programming (CGP) models. Both GRN models are evolved using a novel genetic search algorithm for parameter search, which can be re-used for other design problems. It is revealed that the CGP-based method produces results similar to those obtained using the GNN-based methods while offering more interpretability. In this work, it is shown that this EvoDevo approach is able to produce near-optimal truss structures via growth mechanisms such as moving vertices or changing edge features. The technique can be set up to provide design automation for a range of engineering design tasks.

Keywords: generative design; evolutionary development; evolutionary algorithm; gene regulatory network; neuroevolution; graph neural network; graph convolutional network; cartesian genetic programming; genetic algorithm; truss optimization



Academic Editor: Sheng Du

Received: 12 June 2025

Revised: 19 July 2025

Accepted: 19 July 2025

Published: 26 July 2025

Citation: Tahernezhad-Javazm, F.; Colligan, A.; Friel, I.; Hickinbotham, S.J.; Goodall, P.; Buchanan, E.; Price, M.; Robinson, T.; Tyrrell, A.M.

EvoDevo: Bioinspired Generative Design via Evolutionary Graph-Based Development. *Algorithms* **2025**, *18*, 467. <https://doi.org/10.3390/a18080467>

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

In traditional design processes, a designer will take a project brief or specification and use their expert knowledge to construct a solution and then test it to see whether it meets the set of requirements. There is normally an iterative process between the design and analysis stages before the part is sent to manufacturing. As a result of time, cost and resource constraints, this approach can lead to design fixation or non-value-adding tasks [1,2]. Generative design (GD) looks to address these issues via automation. Generally, it can be summed up as a series of tools that use advanced algorithms and artificial intelligence to create designs for a given set of criteria. This involves the exploration of a vast number of design options, presenting a range of design solutions to the user for selection [3]. The design solutions that emerge may be different from the general best practice or from what may be possible to discover using manual design processes.

Some GD systems consist of both an expert system and an optimisation system [4]. The expert system emulates the expert's knowledge and ability to generate and critique solutions to a specific design problem. An optimisation system, however, improves designs with respect to some objective criteria. Unlike traditional optimisation, where objectives relate to weight or cost, for example, GD systems also optimise other criteria such as aesthetics or manufacturability. The optimisation task may not be concerned with discovering global optima; instead, a multi-modal optimisation task can be performed to find multiple solutions to a problem.

Commonly, topology optimisation acts as an engine to improve the design of a pre-defined shape. GD not only deals with topology optimisation but also integrates other techniques such as shape and size optimisation. This integration of different optimisation techniques can often be a limitation to interoperability between the geometric representations required for topology and shape/size optimisation. Generally, topology optimisation utilizes the finite element method (FEM), where the design space is discretised into simple primitives called *elements*. This discretised geometry is discontinuous and can be difficult to map back to the parametric computer-aided design (CAD) model (i.e., boundary representation (B-Rep)) used for shape optimisation [4]. There can also be a high computational cost associated with generating and analysing the large number of designs required for a GD system. For small-to-medium-sized businesses with no access to high-performance computing, this could represent a barrier to adoption.

Bioinspired GD systems apply features of biological systems to the design creation process [5]. In this paper, inspiration is taken from the biological concept of evolutionary development (EvoDevo) to evolve development rules that can be used to generate design solutions for different engineering problems using evolutionary algorithms [6]. This can be thought of as “evolving the designer, not the design”. By encoding the development rules in the genome of the evolutionary algorithm instead of in the design itself, a richer exploration of the design space is possible. In this paper, this is applied to the automated design of truss bridge structures, a popular benchmark in structural optimisation [7,8]. In EvoDevo, an initial design is divided into basic units called *cells*, each containing an identical copy of an artificial gene regulatory network (GRN) model. This GRN model governs the growth of the structure through a set of developmental rules applied to each cell. The GRN model itself is encoded in a genome, which is optimised using an evolutionary algorithm.

Learning from graph-structured data using graph neural networks (GNNs) offers a promising approach, as GRN models reason over graphs, with trusses being naturally representable in this format. However, the “black-box” nature of neural network outputs can make interpretation difficult. To address this, a novel configuration of a Cartesian genetic programming (CGP)-based GRN model has been developed within the GNN paradigm, offering more interpretable “white box” outputs. This paper explores both approaches. These

GRNs are developed utilising an evolutionary search algorithm to regulate local development mechanisms, including the movement of vertices or the alteration of link features. The objective is to demonstrate that this EvoDevo framework can generate near-optimal truss structures. Once the development rules are established, they may be repurposed to address analogous engineering challenges, hence decreasing the computational expense of the GD system for the end user and facilitating design automation.

The remainder of this paper proceeds as follows: Section 2 reviews prior work in GD, focusing particularly on optimisation paradigms and geometric representations. The proposed EvoDevo framework is proposed in Section 3 in detail. Section 4 describes the experimental setup parameter settings and evaluation metrics. Section 5 demonstrates the experimental results and compares the two proposed GRN controllers. Finally, Section 6 summarises the main findings, outlines the practical results and highlights directions for future work.

2. Related Work

There are two areas of research to consider when applying GD to real-world engineering problems. The first is the *learning algorithm*, which is the means by which appropriate GD is discovered. The second is the *design representation*, which is the means by which the specific design problem is represented to the learning algorithm.

2.1. Gradient- vs. Non-Gradient-Based Methods

Gradient-based methods utilise sensitivities (i.e., gradient information) to explore the design space for optimal solutions. Many such methods can efficiently solve high-resolution problems requiring millions of design parameters using a small number of function evaluations [9]. Common gradient-based methods include density-based (e.g., Solid Isotropic Material with Penalisation (SIMP)) [10] and level-set [11] methods.

For applications in GD, there are several limitations that make gradient-based approaches less attractive. GD systems require optimisation over a wide range of criteria such as manufacturability or multi-physics performance criteria. Therefore, the calculation of gradients may be infeasible due to numerical noise or a large number of variables [12], or, simply, the required objective function may not be differentiable. Gradient-based approaches are also not robust to the noisy or discontinuous search spaces that can result from the non-linearities of GD problems, and they are sensitive to the initial solution [12].

Non-gradient-based or black-box methods often utilise evolutionary algorithms to obtain near-optimal solutions. Compared to gradient-based methods, non-gradient-based methods tend to have higher computational costs due to the need to analyse large populations of designs, therefore needing many simulations. Non-gradient-based approaches tend to be more robust to the limitations mentioned previously for gradient-based approaches. The use of non-gradient-based approaches allows for greater flexibility in terms of application for GD [12,13]. The use of evolutionary algorithms for GD is desirable for the multi-modal optimisation requirements of a GD system. Due to the population-based approach, these methods are able to maintain populations of possible solutions that contain diversity and meet the set objectives.

2.2. Representation of the Design

Within topology optimisation and GD, there have been many attempts to represent the design domain (the structure of the design). Due to the computational cost of non-gradient-based algorithms, the representation chosen for this is paramount, as it defines the search space dimensionality. The chosen structural representation also constrains the possible solutions and the ability to find optimal solutions [12]. For evolutionary algorithms,

this representation is synonymous with the genotype. In the literature, the different representations have been grouped into grid, geometric and indirect representations [13].

2.2.1. Grid Representation

In this representation, the design space is discretised by a grid. For evolutionary algorithms, the genotype encodes properties for different locations on the grid. This representation is commonly used in density-based topology optimisation [10]. The resolution of the grid is often high; therefore, the number of grid elements can be controlled within the millions to represent a specific geometry. For evolutionary methods, increasing this grid increases not only the cost of computing a fitness value but also the size of the chromosomes. This large number of grid elements generates an intractable search space, which consequentially requires large computational resources to converge to the global optima [14].

2.2.2. Geometric Representation

In geometric representations, the structure is represented by a set of shape primitives/components, which can either be fixed or free to move. It is the parameters of these primitives that are encoded in the genotype for evolutionary algorithms. The properties of these shape primitives can be the position, thickness or shape of a structure or sub-structure. The complexity of the solution depends on the number of primitives; hence, the dimensionality of the search space can be greatly reduced compared to that of grid representations [12].

The genotype of Voronoi representations encodes the coordinates and material properties of Voronoi cells. These representations allow for the subdivision of the design domain into arbitrarily shaped polyhedra [15,16]. For level-set methods, the shape boundary is defined by a level-set function. This representation has become widespread for gradient-based optimisation, where it has the advantage of producing smooth, watertight geometry compared to density-based approaches. There have been several approaches that have utilised evolutionary algorithms with a level-set representation. Some of these approaches have used beam-shaped components to construct the level-set boundary [17].

Other approaches have represented the structure as a graph. Encoded in the genotype are the coordinates of the nodes and the properties of the edges (e.g., thickness). The edges can be characterised as Bézier curves, allowing for a larger exploration of shape [18–20] or beam elements, parameterising their shape and thickness [21,22]. This maintains a direct link with the analysis model, similar to grid representations, thereby reducing interoperability issues.

Constructive solid geometry (CSG) involves the use of simple primitives and Boolean operations to create a more complex geometry. Several authors have proposed defining nodes via boundary conditions and using a Delaunay triangulation to produce the edges of the graphs [23,24]. Optimisation involves the parameterisation of these entities. Recently, Wang et al. [14] showed how a CSG representation can be combined with additive manufacturing constraints in a GD system. Watson et al. [4] split the GD process into a topology optimisation step on a grid representation, followed by a shape optimisation step on a graph representation, where the graph was extracted from the grid representation using image processing algorithms.

2.2.3. Indirect Representation

Compared to grid and geometric representations, where the structure is directly parameterised by the optimisation model, indirect representations instead look to implicitly optimise these parameters via a generative model or development rules [12]. Their benefit over grid and geometric representations is that they allow for the formation and

re-deployment of problem-specific design patterns [12]. Although this could mean a high computational cost initially in the formation of the generative model, once formed, the ability to redeploy could alleviate the computational cost for the end-user.

Deep learning methods have recently been used in GD and topology optimisation. Within the field of deep learning, there are several different types of generative models, many of which have been used in the application of GD. Previous works have applied variational autoencoders (VAEs) within topology optimisation to allow the network to reconstruct optimal designs for given boundary conditions [25]. Others have used reinforcement learning via a proximal policy optimisation (PPO) model for wheel design [26]. The issue with deep learning methods is that they are not sample-efficient and require large amounts of data that do not exist in the field of engineering. Many of these approaches have utilised grid-based representations, which becomes computationally prohibitive when moving to 3D voxels due to the cubic scaling factor [27]. Conversely, bioinspired algorithms do not require large amounts of initial training data, and they are less dependent on the extensive predefined structure or specific parametric assumptions found in many deep learning algorithms.

The following approaches all take inspiration from different concepts and mechanisms in biology to produce and control the developmental rules used to generate a design. Price et al. [28] created heuristic growth rules for development in a bioinspired GD system. Through cellular division and other plant growth analogies, a design with additive manufacturing constraints was grown from a single cell “seed” to a mature organism by reacting to external stimuli.

EvoDevo has been applied via Lindenmayer system (L-system) representations. Tiago et al. [29] utilised L-systems to model the cellular division of truss structures for topology optimisation. Artificial GRNs have also been used for EvoDevo in structural optimisation. GRNs are the controllers in cells that switch genes on and off due to the state induced by stimuli in the environment, allowing the cells to grow in a certain manner. This allows for the development of an organism from a single cell to a fully developed organism. The use of GRNs for the design of complex structures was first proposed by Steiner et al. [30]. The proposed GRN model included cellular division, cell specialisation into material and void cells and the physical interactions of cells. Later, Steiner et al. [31] added chemotaxis (the movement of cells relative to the concentration gradient of chemicals), cell adherence and repulsion.

Neuroevolution is the field of using evolutionary algorithms to optimise the parameters of neural network models [32]. Aulig et al. [33] used a state-based representation, where a neuroevolutionary approach evolved the weights of feedforward neural networks that mapped local sensory information to an update signal for the topology optimisation of a grid-based structure. Others have used forms of the neuroevolution algorithm NeuroEvolution of Augmenting Topologies (NEAT) [34] that can also evolve the topologies of neural networks, as well as the weights and biases. Cheney et al. [35] used HyperNEAT [36] for vibration design using a grid representation. Prior work by Hickinbotham et al. [37] showed the ability to use a GRN evolved using NEAT to move node positions in a Warren truss to produce solutions that are more optimal in minimising objective functions such as the total strain energy and total volume. This combined the GRN approach with state-based approaches and operated on a geometric representation of the design.

Furthermore, Cartesian genetic programming (CGP) [38,39] is a graph-based evolutionary algorithm that allows for the efficient evolution of GRN structures. In general, CGP employs fixed-size grids of nodes (including columns or rows), in which node inputs are mostly from nodes in earlier columns. Also, the genomes in CGP comprise connection genes and primitive functions that are optimised through an evolution process with a

mutation operator [40]. CGP offers two practical advantages over other GRN models: computational efficiency and interpretability. During optimisation, only the active sub-graph, often a small fraction of the full genotype, is executed, so evaluation times decrease significantly. In addition, CGP exposes clear routes between inputs and outputs, making the resulting network easier to interpret than GRNs based on GNNs or other neural network models [40,41].

This section highlights how gradient-based, non-gradient and indirect representations each address different aspects of GD. In the next section, we translate these insights into a new framework, detailing our cellular representation, growth mechanisms and the dual GNN/CGP gene regulatory networks that drive development.

3. Proposed Method

3.1. Overview

Akin to Price et al. [28], the approach adopted here is to develop generative models that encompass the development of an organism from a single cell to a fully developed organism (Figure 1). The process starts with a minimal seed that meets the baseline requirements. In each development step, each cell senses its local state via message passing and aggregating information from neighbouring cells. This information is passed through GRNs, which generate growth commands. These commands are necessary for changing the global geometry and the cell states for the next steps. Through a loop of sensing, decision and execution, the cells grow and end up with the optimised structure based on the predefined fitness criteria.

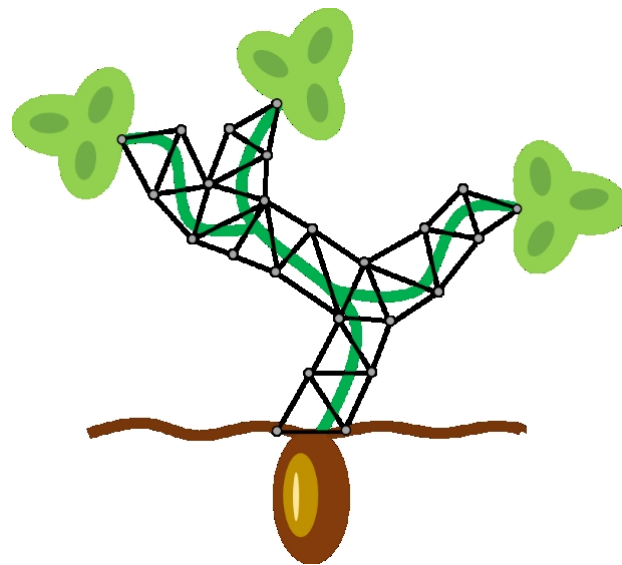


Figure 1. Concept of a design growing from a seed. Growth is regulated through the continuous interaction between external stimuli and the outputs of the GRN.

3.2. Cellular Representation

As discussed in Section 2.2, there are many different ways to represent the geometry for GD. When the geometry is encoded directly into the genome of an evolutionary algorithm, there are not many constraints on what geometries can be represented; however, scaling to complex problems becomes a real and often insurmountable issue. Here, an indirect representation is used, where the genome encodes a GRN represented by a neural network. A key consideration for such a neural network-based GRN is its interaction with geometric data. Traditional neural networks, due to their fixed input vector size, typically require shapes to be described in regular, grid-like formats (e.g., pixels or voxels). Most 3D

shape representations such as meshes or B-Reps do not have consistent or predefined arrangements. To enable our GRN to operate directly on such irregular, graph-structured cellular designs, this work leverages GNNs, which have been shown to be able to deal with these kinds of irregular shape representations [42–46]. A geometric representation is therefore constructed with these key limitations and advantages in mind.

As with [28,37], the design is divided into simple primitive entities called *cells*. As in biology, these are the basic building blocks that together can be used to construct more complex systems. Each of these cells is an independent entity that contains a local controller via a GRN for growth but interacts with the global structure through its shared topology. In this paper, cells are considered to only have structural properties and function. A graph is constructed from these cellular entities and their shared topology. In this paper, these cells also represent the finite elements used for the design's analysis, which helps to reduce interoperability issues, in a similar manner to some previous beam element approaches [21,22].

For this initial study, truss structures are explored as the benchmark problems. To represent a truss structure, a heterogeneous graph of multiple node and edge types can be used. Each node represents a distinct cell type, and the edge types represent different adjacency information between the same and different cell types. This graph can have a hierarchy, capturing features from local to global scales, as illustrated in Figure 2. In this paper, two cell types are used, representing the vertices/nodes and edges/members of the truss structure. These edge cells can be characterised as truss elements in a finite element analysis (FEA) when solving statically indeterminate structures. All graph edges are assumed to be undirected, although direction can also be encoded.

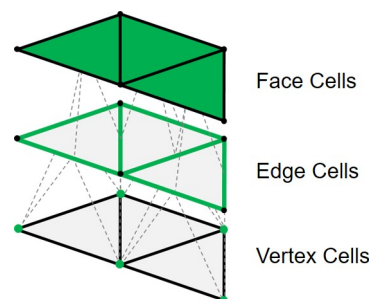


Figure 2. A heterogeneous graph representing a shape, unfurled into a hierarchy of cell types and topologies. Cells can be represented as faces, edges or vertices; however, this paper focuses on the edge and vertex representations.

3.3. Evolutionary Development

The EvoDevo algorithm consists of an evolution (evo) process and a development (devo) process. The evo process optimises the GRN model in the shape of genomes. The devo process can be thought of as the life-cycle of the design/organism. It is where the cells of the design will change in terms of shape, size and topology as a sequence of rules in order to adapt to the environment in which they are placed. Once the devo process is complete, the organism, or, more specifically, the shared GRN of the cells, will be given a reward based on its performance. The rewards of the organisms are then used in the selection process in the evo process, which decides which organisms and their genetic material are allowed to pass to the next generation.

3.4. Development

The development of a design/organism can be cast as a reinforcement learning problem, where the agent/GRN in each cell interacts with an environment and must produce a sequence of actions (here, devo steps) that yields a trajectory in the objective space to

maximise some cumulative reward. The problem is underpinned by assuming the Markov property that the history of all previous states is encapsulated in the current state (e.g., the x and y coordinates of joints and the strain energy and material volume of members). This means that, at any given devo step, a cell's growth decision is only based on its current and potential future states. It is also assumed that the environment is deterministic and that no uncertainty exists between transitions from one state to another. The action space is continuous, where the GRN needs to determine the appropriate values for a set of continuous control parameters corresponding to a given growth mechanism.

3.4.1. Environment

In the proposed framework, the *environment* is the collection of problem-specific properties and constraints that influence the development of the designs. The actions update the cell states at every development step and therefore change and define the search space. The organism and its cells interact with stimuli in the environment, updating the cell states in the process. The environment sets up the problem search space in which solutions can be found. Therefore, knowledge of the problem is required to effectively define the environment to ensure that the desired solutions are output from the system. The search space is defined through the construction of different environmental elements. These relate to the engineering discipline of interest, e.g., structural, thermal or manufacturing. Within an environmental element, different variables can be added, such as stimuli, objectives and constraints. In this paper, the focus is on the structural environmental element that can be used to generate feedback for different structural analysis problems, with other environmental elements being explored in future work.

In this paper, potential designs for statically determinate and indeterminate pin-jointed truss structures are considered. The tension coefficient method and FEA with truss elements are used to calculate the forces on these statically determinate and indeterminate truss structures, respectively. An analytical method via the tension coefficient method is used to speed up analysis where possible. The main objectives explored in this paper are to minimise the total strain energy and the total volume of the designs, with the aim of generating organisms with optimal stiffness-to-weight properties. A lower strain energy value for a given load indicates a higher overall structural stiffness. The total strain energy is the energy stored in a link as a result of its elastic deformation.

3.4.2. Growth Mechanisms

Each cell uses its GRN to make an independent decision on its local growth. Evolutionary selection is used to maximise the design's performance globally (by minimising an objective function). The action space of this growth is parameterised by the growth mechanisms available to the cell: different growth mechanisms allow for different explorations of the design space in terms of shape, topology and size. These growth mechanisms can also be linked to certain engineering disciplines and engineering knowledge. For performance, it is important that there is a direct mapping between the cell and the growth mechanism that acts upon it. This decreases the noise in the search space and allows for more generalizable GRNs. In this contribution, the vertex/node cell type has the ability to change its Cartesian coordinate position, and the edge cell type can alter its cross-sectional area. These parameters can be altered in a continuous range of $[-1, 1]$ at each devo step.

3.5. Gene Regulatory Network

In a single biological organism, the cells all share the same genome but have different gene expression levels as a reaction to the local state of the cell. The GRN is the controller in the cell that switches genes on and off due to stimuli in the environment, allowing for the cell to grow in a certain manner in response. The development of an organism from a

single cell to a fully developed organism occurs due to local responses on a cellular level and independently of a global controller. In this work, inspiration is taken from gene regulatory networks and their ability to dictate development without the influence of a global controller using state-based information.

There are many different proxies for gene regulatory networks in the literature. These can be classified into logical, continuous and single-molecule-level models. The different classifications can be placed on a spectrum of ranging model complexities or speed of analysis. Logical models tend to have a lower model complexity but suffer in terms of the speed of analysis. Single-molecule-level models instead have a higher model complexity leading to a lower speed of analysis. Continuous models, which include ordinary differential equations (ODEs), genetic programming and neural networks, lie somewhere in the middle between the other two classifications. As with Hickinbotham et al. [37], the GRN in the system is represented by a neural network. This supports a certain level of model complexity, but, as multiple simulations would also need to be run, faster analysis speeds would also be required.

One major difference from [37] is that a GNN is used instead of a simple feedforward neural network for the GRN. There are several advantages of using a GNN for learning from these kinds of data representations and shapes in general. As discussed previously, shapes can be represented in graphical representations with topological relationships between entities. GNNs are designed to process and learn from this graph-structured data, making them a natural fit to the problem at hand. GNNs are also able to capture both the local and global contexts within a graph. In the context of shape representations, this means that GNNs can effectively capture the local geometric relationships between neighbouring entities while also incorporating global shape information. GNNs can also learn from hierarchical representations by propagating information across the graph, capturing and aggregating features from local to global scales. This propagation of information or message passing is very important, as it enables cell-to-cell communication. This communication is not constrained to cells of the same type, meaning that cells are able to have different types in the representation used here, e.g., both vertex and edge types, as shown in Figure 2, enabling more flexibility and generalisation to the approach.

In the proposed neural architecture, graph convolutional layers are used to filter and aggregate state information from cells and their neighbouring cells. The graph neural network operator [47] used is shown in Equation (1):

$$\mathbf{x}'_i = W_{\text{self}}\mathbf{x}_i + W_{\text{nbr}} \sum_{j \in \mathcal{N}(i)} e_{j,i} \mathbf{x}_j. \quad (1)$$

where \mathbf{x}_i denotes the current feature vector of node i , and \mathbf{x}'_i is its updated feature vector after one graph convolution operation. The set $\mathcal{N}(i)$ contains all nodes that share an edge with node i in the graph. The learnable weight matrix W_{self} filters the target node's own features, while W_{nbr} filters the aggregated features received from the neighbouring nodes. The scalar $e_{j,i}$ is the edge weight from node j to node i ; it is set to 1 in the present study.

As previously stated, information can be propagated between different cell types. Thus, vertex cells exchange messages not only with other vertices but also with edge cells. Therefore, we decompose the neighbourhood into *intra-type* and *inter-type* sets:

$$\mathbf{x}'_i = W_{\text{self}}\mathbf{x}_i + W_{\text{intra}} \sum_{j \in \mathcal{N}_{\text{intra}}(i)} e_{j,i} \mathbf{x}_j + W_{\text{inter}} \sum_{k \in \mathcal{N}_{\text{inter}}(i)} e_{k,i} \mathbf{x}_k. \quad (2)$$

Here, $\mathcal{N}_{\text{intra}}(i)$ contains neighbours of the same cell type as node i , whereas $\mathcal{N}_{\text{inter}}(i)$ contains neighbours of a different type. The weight matrices W_{intra} and W_{inter} can therefore

learn distinct linear transformations between possibly different feature dimensions, so cell-type X_k is not required to share the same feature size as X_i ($W_{nbr} = W_{intra} + W_{inter}$).

For both types of neural network operators, a leaky Relu activation function [48] and bias operator are used. The GRN also has a fully connected layer after the graph convolutional layer. This uses a hyperbolic tangent or tanh function to give a continuous output for the growth mechanism in the range $[-1, 1]$. The tanh function serves as a stabilising mechanism that maintains equilibrium during development while permitting both positive and negative modifications. The absence of the tanh function results in significant output oscillations, causing simulation failures. For instance, Figure 3 shows the architecture used to optimize the structure by changing the vertex positions and edge cross-sectional areas simultaneously. This is constructed by two GRNs, one for the vertex cells and one for the edge cells. The edge cells hold state information in terms of their strain energy and volume. The vertex cells hold state information in terms of their Cartesian coordinates. The edge cells are able to communicate their state information to adjacent vertex cells. Therefore, the vertex cells make growth decisions based on their own state and the state of adjacent vertex and edge cells. It is decided that the vertex cells should not communicate their state to the edge cells, as it is assumed that the vertex position is not necessary for the mapping of the edge state and its growth mechanism.

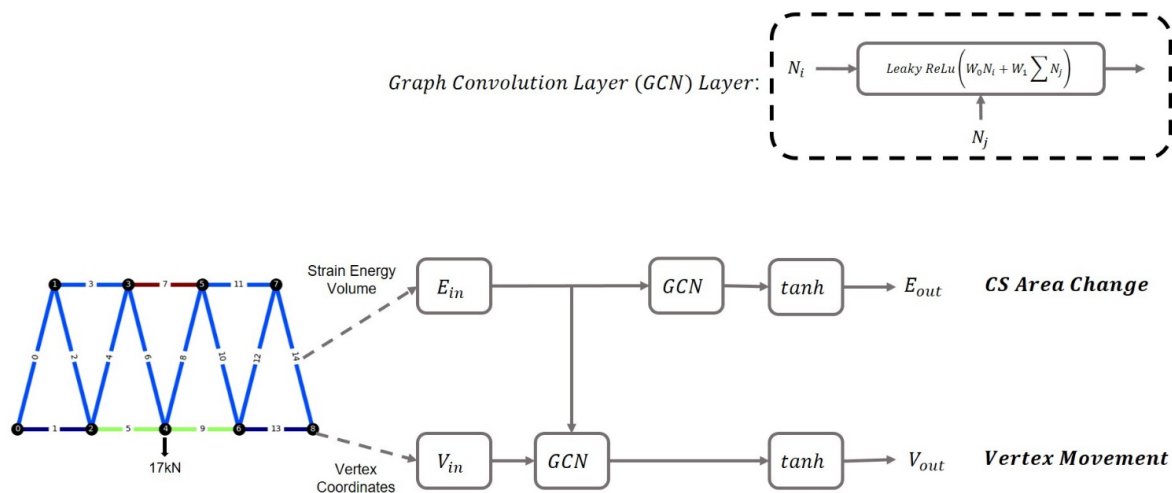


Figure 3. Hybrid control structure: two gene regulatory networks for vertex and edge cells with neighbour message exchange. A vertex GRN updates node positions $(\Delta x, \Delta y)$, while an edge GRN adjusts member areas ΔA .

3.6. GRN Implemented as Cartesian Genetic Programming (CGP)

Alongside the GNN, an alternative implementation of a GRN based on CGP is designed. Compared to the GNN, CGP offers a human-readable way to encode a GRN with a grid of nodes and logical operators in a linear chromosome. Each chromosome is simply a vector of numbers that is interpreted to build a directed grid-based graph. The two-dimensional grid consists of R rows and C columns, and each grid intersection hosts a single computational node. The type of computation is determined by an operator chosen from the primitive set such as addition, subtraction, multiplication and division. Every node has exactly k inputs (generally $k = 2$) and one output. In CGP, there is a level-back parameter L , which guarantees acyclicity. Node (r, c) can only read from primary inputs or from nodes situated in columns $c + 1, \dots, c + L$. In CGP, genomes evolve under a $(\mu + \lambda)$ mutation operator. Indeed, exploration and exploitation rely on point mutation; only with probability ε is a field in the chromosome replaced (primitive or connection) or perturbed, and crossover is omitted.

To implement the CGP-based GRN, several CGP structures are instantiated that differ in the number of inputs, the number of outputs and the way that neighbourhood statistics are aggregated (similar to a GNN). Table 1 lists the exact settings used in these experiments; all genomes share the same grid size, level-back and primitive set.

Table 1. Summary of CGP methods.

CGP Structure	Inputs	Outputs	Purpose
edge	2	1	strain energy & volume $\rightarrow \Delta A$
node	2	2	$(x, y) \rightarrow (\Delta x, \Delta y)$
node–edge (hybrid)	2/2	2/1	two cgp controllers update hierarchically

The edge controller (row 1 of Table 1) is the minimal CGP structure. Each truss cell receives only its own strain energy (SE) and volume (V) and returns a single increment ΔA for the cross-sectional area. Also, the node controller maps the current vertices/node coordinates (x, y) to a displacement vector $(\Delta x, \Delta y)$. Ultimately, two hybrid node–edge controllers co-evolve a pair of CGP controllers. node-cgp controls the node positions, and edge-cgp adjusts the members' cross-sectional areas.

3.7. Evolution and Selection

After each development step, the total normalised strain energy and volume (each is normalised by its initial value) is recorded and summed to give the fitness $F = SE_n + V_n$. In addition, the incremental reward $\Delta F = F_t - F_{t-1}$ is computed through all development steps, resulting in overall progress rather than single-step jumps. This reward, which is the summation of all ΔF over 10 development steps, is considered the criterion for ranking the performance of each CGP controller and selecting the promising ones for the next evolutionary generation. The selection of 10 steps is determined empirically as a balance between performance enhancement and computational expense [49,50].

3.8. GRN Implemented as Neuroevolution (GNN)

To learn the growth rules that underpin development in different environments, an evolutionary parameter search of the graph neural network GRNs is performed. Neuroevolution has been shown to be a powerful and flexible technique that can be used in a wide range of reinforcement learning problems, especially those with complex and continuous state and action spaces. Therefore, a novel neuroevolutionary genetic search algorithm is proposed to implement this parameter search of the graph neural networks. The genome of each organism in the population encodes the weights and biases of each layer of its graph neural network (GRN). This genetic search algorithm evolves the GRN through the process of mutation and selection. This genetic search algorithm can be seen in Figure 4. The initial populations of the GRNs are parameterised by random weights and biases. Each of these GRNs is placed in their own development loop, where they grow each cell in an organism until a final design is produced. This final design has an associated cumulative reward based on its performance in the environment relative to the predefined objectives. A selection process is then undertaken, where the best performing GRN in terms of this reward metric is passed to the next generation, and offspring are produced via the mutation of the weights and bias parameters of the two parent GRNs.

The mutation rate ϵ dictates the probability of changing the weights and biases in the GRN. It is randomly sampled from a Gaussian distribution with a mean of zero and a standard deviation equal to the mutation rate. There is a dilemma between the exploration and exploitation of the solution space. The process should not get stuck in local optima or commit to certain GRN parameterisation too early in the generations. An adaptive

mutation rate strategy is employed to allow for a wider exploration in the initial generations with a larger probability of mutation occurring, with the mutation rate decreasing in the later generations allowing for more fine-tuning of the better performing GRN. The mutation rate ϵ can be computed as

$$\epsilon = \frac{\epsilon_0}{1 + \epsilon_{\text{taper}} \text{generation}_{\text{id}}} \quad (3)$$

where ϵ_0 is a hyperparameter denoting the initial mutation rate, ϵ_{taper} controls the amount in which the mutation rate is decreased each generation, and $\text{generation}_{\text{id}}$ is the identity of the current generation.

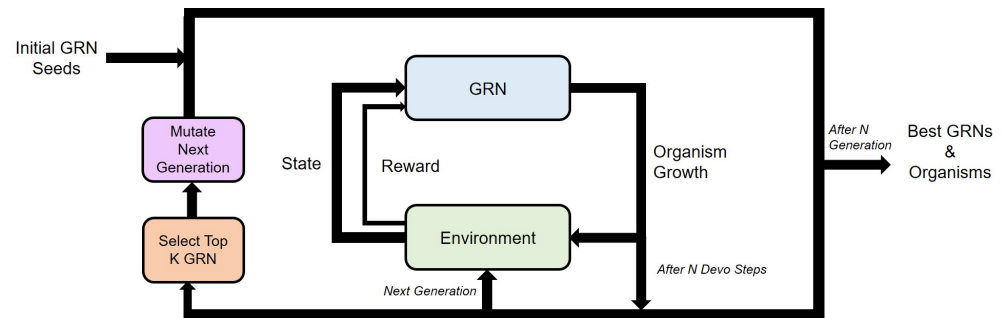


Figure 4. Genetic search algorithm for evolving GRN.

In evolutionary algorithms, letting every GRN reproduce in every generation would be wasteful. Instead, we adopt the elitist selection scheme used in Augmented Random Search [51]. After each generation, individuals are ranked by their reward, and the top k elites (a user-defined hyperparameter) are copied unchanged into the next generation. The remaining population slots are filled by offspring generated through the mutation of these k elites, ensuring an efficient exploitation of high-performing regions while still permitting directed exploration.

3.9. Normalising Inputs

To be able to create a GRN that is able to generalise to new environments, the inputs to the GRN (i.e., cell states) must be normalised. This also helps to ensure that one feature type does not have dominance over another. Normalising the data is difficult, as it is hard to define the minimum and maximum that will be achieved by each cell state feature. A Z-normalisation scheme is applied across the features of each cell's state to set the mean μ to zero and the standard deviation σ to one for the organism at each development step.

$$X' = \frac{X - \mu}{\sigma} \quad (4)$$

The moving average of μ and σ is computed at each development step and is used to help normalise the cost features over the course of development. At each devo step $n \in \{1, 2, \dots, N\}$, each objective $f_i(x)$ where $i \in \{1, 2, \dots, k\}$ is normalised by the objective at the initial devo step $f_i^0(x)$.

$$f_i^n(x) = \frac{f_i^n(x)}{f_i^0(x)} \quad (5)$$

The normalised objectives at each devo step $f_i^n(x)$ are then combined into a single objective $F_i^n(x)$.

$$F^n(x) = \sum_{i=1}^k f_i^n(x) \quad (6)$$

The reward metric is used to evaluate the organisms in the population. The reward is defined as

$$R = - \sum_{n=1}^N (F^n(x) - F^{n-1}(x)) \quad (7)$$

This essentially defines a comparison of the single objective at devo step n against the initial objective at devo step 0 for N devo steps. The minus sign inverts the sign of the metric for maximisation. Algorithms 1 and 2 represent the neuroevolution procedure described above. The algorithm consists of a two-level nested process: the inner process is the devo stage, where the GRN (either the GNN or CGP) applies the developmental rules, while the outer process (evo) involves the application of the mutation operator.

Algorithm 1 EVODEVOALGORITHM—outer loop

Require: $GRNType \in \{CGP, GNN\}$ ▷ choose the GRN model

- 1: *EliteSize* (number of top individuals to keep)
- 2: *OffspringCount* (new individuals per generation)
- 3: *NumGenerations* G
- 4: *MaxDevSteps* T
- 5: **initialise** *Population* \leftarrow CREATEINITIALPOPULATION(*EliteSize*, *GRNType*)
- 6: **for** $g = 1$ to G **do**
- 7: **for** *organism* in *Population* **do**
- 8: *organism.Fitness* \leftarrow DEVELOPORGANISM(*organism*, T , *GRNType*)
- 9: **end for**
- 10: *EliteGroup* \leftarrow SELECTTOP(*Population*, *EliteSize*)
- 11: *OffspringGroup* $\leftarrow \emptyset$
- 12: **while** size of *OffspringGroup* $<$ *OffspringCount* **do**
- 13: *Parent* \leftarrow RANDOMCHOICE(*EliteGroup*)
- 14: *Child* \leftarrow MUTATEGENOME(*Parent*, *GRNType*)
- 15: add *Child* to *OffspringGroup*
- 16: **end while**
- 17: *Population* \leftarrow *EliteGroup* \cup *OffspringGroup*
- 18: **end for**

Algorithm 2 DEVELOPMENT LOOP—inner loop

Require: *Genome*, *MaxDevSteps* T , *GRNType*

- 1: build initial *SeedlingStructure*
- 2: *CumulativeFitness* $\leftarrow 0$
- 3: **for** $t = 1$ to T **do**
- 4: gather and normalise current cell states \rightarrow *InputStates*
- 5: **if** *GRNType* is CGP **then**
- 6: *Actions* \leftarrow CGP(*Genome*, *InputStates*)
- 7: **else**
- 8: *Actions* \leftarrow GNN(*Genome*, *InputStates*)
- 9: **end if**
- 10: apply *Actions* to update geometry and
- 11: recompute physics; update *CumulativeFitness*
- 12: **end for**
- 13: **return** *CumulativeFitness*

3.10. Seedling

The seedling describes the starting point from which the development of the organism begins. This seedling can be thought of as the minimal viable structure required to start interacting with the environment and defining load paths. The seedling is identical across all organisms in every generation. This allows for a fair, objective comparison in the evolutionary selection process. The initial structure of the seedling can be given to the GD

system if further optimisation on a sub-optimal design is required. The geometries of this seedling are not constrained, but, in this paper, as in [37], a Warren truss bridge is selected as the seedling.

The proposed framework combines a heterogeneous graph representation with GNN and CGP-based GRNs, with the parameters adjusted using neuroevolutionary search. By combining cellular-level sensing, decision making and actuation within the EvoDevo loop, the system can generate optimum truss architectures from a little seedling. The following section exhibits the use of these approaches to benchmark problems. Table 2 encapsulates the fundamental components of the EvoDevo algorithm and connects each with its biological equivalent. This provides additional insights into the translation of concepts between different domains for the remainder of this paper.

Table 2. Key EvoDevo concepts and their biological counterparts.

This Work	Biological Analogue	Purpose
Seedling truss	Embryo	Minimal viable structure that initiates growth.
Node/edge cells	Somatic cells	Local agents that sense state and apply growth rules.
GRN	GRN	Maps sensed state to growth commands for each cell.
GNN/CGP weights	Genome	Heritable code evolved by the genetic algorithm.
Development step	Cell cycle	One round of sensing, decision and growth.
Growth mechanism	Cell growth	(i) Move a node, (ii) resize an edge.
Fitness	Fitness	Global signal guiding evolution towards better performance.
Genetic algorithm	Darwinian evolution	Selects and mutates GRNs each generation.

4. Experiments

Two GRN varieties (a GNN and CGP) are implemented for the three experimental treatments described here. Each experiment investigates the development of truss bridges, although the approach is not limited to these types of problems. Every evolutionary experiment, unless stated otherwise, uses the parameters in Table 3.

Table 3. Default evolutionary experiment parameters.

Development steps = 10	Generations = 150	Population size = 512
Top $k = 32$	Obj ^a 1: strain energy (se)	Initial mutation rate (ϵ_0) = 1.0
Mutation rate taper (ϵ_{tap}) = 0.4	Obj 2: volume (v)	

^a Objectives.

The number of best individuals that move forward each generation is fixed at $k = 32$ (6.25% of the population). Based on these work experiments, it is proven that smaller values ($k < 16$) lead to premature convergence, while larger values ($k > 64$) slow progress [52,53]. In addition, Figure 5 illustrates how the mutation rate balances exploration and exploitation over the evolutionary generation.

As stated in Section 3.2, two different cell types are used in the experiments: vertex cells (nodes) and edge cells. Also, we apply a combination of node and edge cells as the ultimate design scenario. The node cells are equivalent to the nodes in the truss, and the edge cells are equivalent to the members. Each cell type has different growth mechanisms available to them to globally change the structure. To test whether the GRN of each cell type can learn growth mechanisms to minimise the objectives, the total strain energy and total volume, a Warren truss with seven triangles and edges with equal cross-sectional areas is used as the seedling. The environment consists of a 17 kN load placed on centre vertex 4, with a pinned support at vertex 0 and a roller support at vertex 8. The bottom line

of the vertices (0, 2, 4, 6 and 8) are constrained to prevent any movement. This seedling is shown in Figure 6, where the colour of the edges represents the strain energy induced.

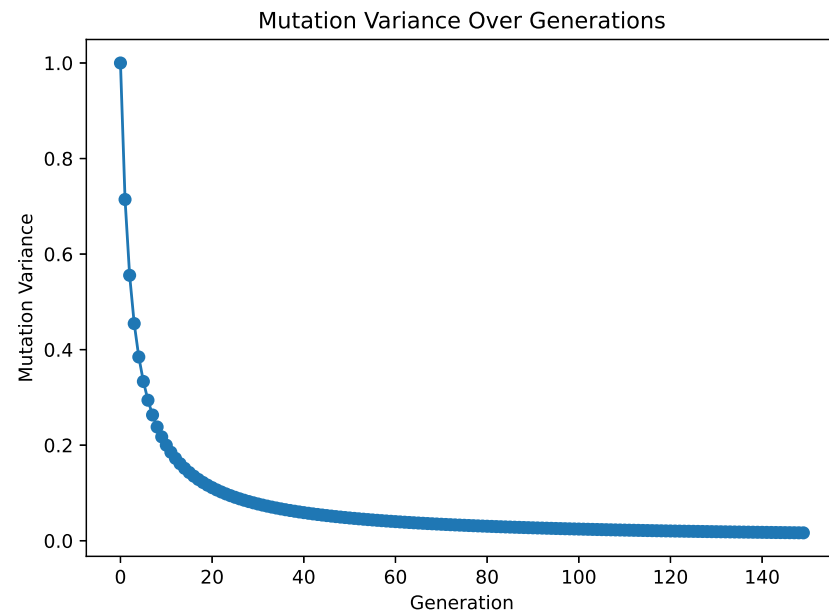


Figure 5. Influence of initial mutation rate $\varepsilon_0 = 0$ and taper coefficient $\varepsilon_{\text{tap}} = 0.4$ on overall mutation rate over evolutionary generations.

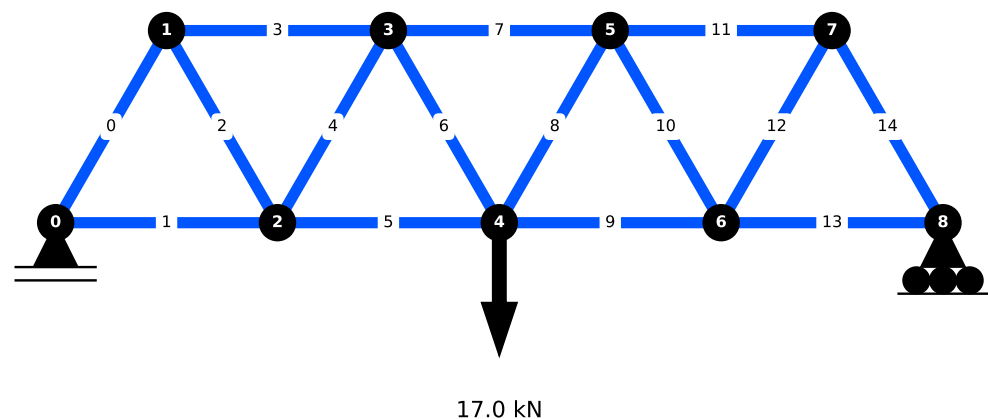


Figure 6. Seedling in single-load (17.0 kN) case environment with 9 nodes and 15 edges.

In the implementation of the GNN structure, the truss is represented as a graph including n vertices and m edges, utilising three binary adjacency matrices in each method (Table 4). All three matrices are static during the development steps and provide the neighbourhood information needed for aggregation.

Table 4. Adjacency and incidence matrices used by the GNN controllers.

$A_N \in \{0,1\}^{n \times n}$	node–node adjacency: $A_N[i, j] = 1$ if vertices i and j share a member
$A_E \in \{0,1\}^{m \times m}$	edge–edge adjacency: $A_E[p, q] = 1$ if members p and q share a vertex
$A_{NE} \in \{0,1\}^{n \times m}$	node–edge incidence: $A_{NE}[i, p] = 1$ if vertex i is an endpoint of member p

As shown in Figure 7, the node-only GNN method converts each vertex's position (x_i, y_i) and the state of its neighbourhood into the final vertex translation $(\Delta x_i, \Delta y_i)$. Each vertex begins with the vector of $\mathbf{n}_i = [x_i, y_i]^T$. Neighbour information comes from two

sums: (i) the *Edge-neigh sum* ($A_{NE}E$), which adds the strain energy and volume of all edges connected to the vertex, and (ii) the *Node-neigh sum* A_NN , which adds the x, y coordinates of the neighbouring vertices. Then, these two vectors are concatenated. The vertex and the neighbour features are linearly projected by $W_0 \in \mathbb{R}^{2 \times 2}$ and $W_1 \in \mathbb{R}^{4 \times 2}$, respectively. Then, these two results are added with bias b_0 .

$$\mathbf{h}_i = \text{LReLU}(W_0 \mathbf{n}_i + W_1 [A_{NE}E \parallel A_NN]_i + b_0) \quad (8)$$

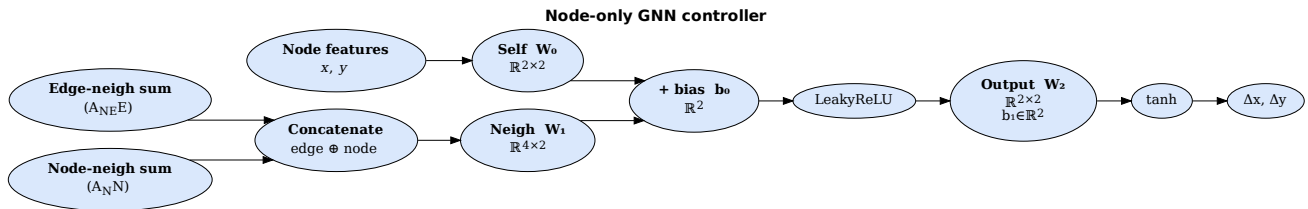


Figure 7. The node-only GNN: This combines coordinates and aggregated neighbour/edge features and then uses linear projections, a leaky ReLU and a final linear tanh function to output $(\Delta x_i, \Delta y_i)$. The weight matrices form the evolvable genome.

The matrices W_0 , W_1 and W_2 with the biases b_0 and b_1 are considered the learnable parameters, evolved by the GA during the optimisation process. The output with weights $W_2 \in \mathbb{R}^{2 \times 2}$ and bias b_1 , followed by a tanh function, produces the vertex movements $(\Delta x_i, \Delta y_i)$.

$$(\Delta x_i, \Delta y_i) = \tanh(W_2 \mathbf{h}_i + b_1) \quad (9)$$

The *edge-only* GNN (Figure 8) calculates the cross-sectional update (ΔA_j) for each edge of the truss. Each member starts with the feature vector $\mathbf{e}_j = [SE_j, V_j]^T$, including the strain energy and volume. The raw edge features are mapped by $W_0 \in \mathbb{R}^{2 \times 2}$, and the summed features of adjacent members, $(A_{EE})_j$, are transformed by $W_1 \in \mathbb{R}^{2 \times 2}$. These two results are added together; a bias b_0 is also included, and the whole output goes through a leaky ReLU function:

$$\mathbf{h}_j = \text{LReLU}(W_0 \mathbf{e}_j + W_1 (A_{EE})_j + b_0). \quad (10)$$

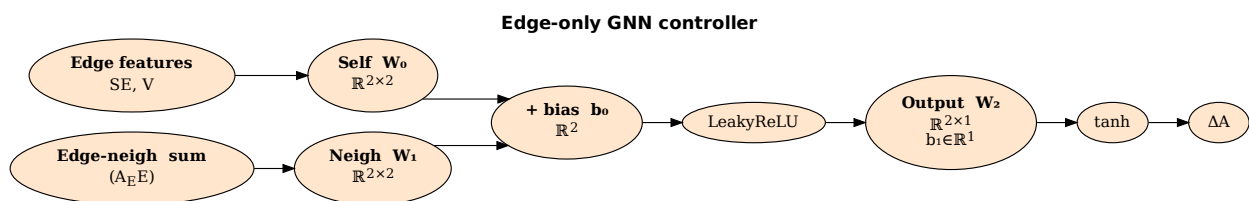


Figure 8. The edge-only GNN: This combines strain energy, volume and neighbour features and then uses linear projections, a leaky ReLU and a final tanh function to output the cross-sectional area update (ΔA_i) . The weight matrices form the evolvable genome.

The final linear layer with weights $W_2 \in \mathbb{R}^{2 \times 1}$ and bias $b_1 \in \mathbb{R}$, followed by a tanh activation function, calculates the cross-sectional area for each edge member (j):

$$\Delta A_j = \tanh(W_2 \mathbf{h}_j + b_1). \quad (11)$$

Finally, Figure 9 shows two parallel GNN controllers for the node–edge method. It consists of two node-only and edge-only methods that update the edge cross-sectional area ΔA_j and joints $(\Delta x_i, \Delta y_i)$ in parallel.

$$\begin{aligned} \mathbf{h}_j^E &= \text{LReLU}(W_0 \mathbf{e}_j + W_1 (A_E E)_j + b_0), \\ \Delta A_j &= 10 \tanh(W_4 \mathbf{h}_j^E + b_4) \\ \mathbf{h}_i^N &= \text{LReLU}(W_2 \mathbf{n}_i + W_3 [A_{NE} E \parallel A_{NN}]_i + b_2), \\ (\Delta x_i, \Delta y_i) &= 10 \tanh(W_5 \mathbf{h}_i^N + b_5) \end{aligned} \quad (12)$$

where $\mathbf{e}_j = [SE_j, V_j]^\top$ and $\mathbf{n}_i = [x_i, y_i]^\top$. In addition, the dashed grey arrow in Figure 9 defines the one-way path $(A_{NE} E)$ from edges to nodes; concluding the edge branch itself does not depend on node data during the process. All six weight matrices and four biases are learnable parameters that the GA tunes during the optimisation process.

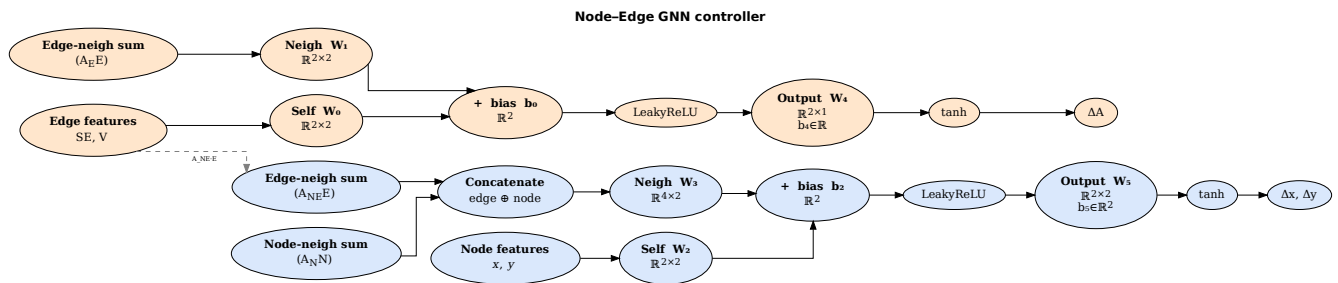


Figure 9. Hybrid node–edge GNN: The blue path maps edge features and their neighbours to the cross-sectional update (ΔA_i) , and the red path maps vertex coordinates plus node/edge neighbours to $(\Delta x_i, \Delta y_i)$. The weight matrices form the evolvable genome.

In the CGP implementation, three methods are employed: node, edge, and node–edge methods. The initial two methods employ a single CGP controller, whereas the final method (node–edge) utilizes two interconnected CGP controllers, with each responsible for node position and cross-sectional changes. The structures of the CGP controllers are outlined in Table 5.

Table 5. Cartesian genetic programming parameters used for each method.

Methods	n_{in}	n_{out}	n_{col}	n_{row}	L_{back}	Primitive Set [†]
CGP_edge	2	1	8	2	3	{Add, Sub, Mul, Const}
CGP_node	2	2	8	2	3	{Add, Sub, Mul, Const}
CGP_node–edge (edge)	2	1	8	2	3	{Add, Sub, Mul, Const}
CGP_node–edge (node)	2	2	8	2	3	

[†] Primitive set abbreviations: $Add(x, y) = x + y$, $Sub(x, y) = x - y$, $Mul(x, y) = x \cdot y$, $Const = \text{constant float}$.

n_{in} and n_{out} denote the numbers of inputs and outputs, respectively. In addition, n_{row} and n_{col} specify the CGP grid dimensions (rows and columns). Finally, L_{back} (levels-back) defines the maximum number of previous columns that a node can take as input.

4.1. Edge-Only EvoDevo

The first experiment investigated how the edge cells (GNN and CGP) could grow to minimise the objectives. The reward metric combined the total strain energy and volume objectives with equal weighting. The strain energy and volume of the edge cell were given as the input to the GRN, and the output was a delta change in the edge's cross-sectional area, where the cross-sectional area of an edge could not decrease by more than 1% of its original value. For each GNN- and CGP-based method, 15 independent runs were applied.

4.2. Node-Only EvoDevo

The second experiment investigated the node (vertex) cell interactions. Although the control variables were changed to the coordinates, the same reward metric was applied. Each cell's GRN took its current x and y positions (as input) and generated Δx and Δy , which moved the nodes. Similar to the edge experiments, 15 independent runs were carried out for both the GNN- and CGP-based controllers.

4.3. Node-Edge EvoDevo

The third and final experiment employed the node–edge method, in which the node positions and member areas were updated simultaneously at every development step. In the CGP version, two coupled controllers were used: one adjusted the cross-sectional areas, and the other moved the vertices. Therefore, the configuration evolved in a single cycle (single development step). The GNN-based method performed both tasks within a single network.

5. Results and Discussion

5.1. Edge-Only Results

Figure 10 shows the final structure that achieved the highest reward in the edge method among the 15 runs. According to the findings in Figure 10, despite the similar structure of these two optimal controllers, the GNN yields slightly lighter components, resulting in a lower volume. Nonetheless, the developmental steps exhibit significant variation. CGP shows a fast reduction in energy and volume, while the GNN represents a gradual decline in all criteria. The CGP controller update rule for each edge is

$$\Delta A = se^2 (5 se - v) (5 se^2 - se v - v)^2 \quad (13)$$

where se is the strain energy in the structure, and v is the volume. The linear term $5 se - v$ shows the direction of changes (negative or positive). For instance, if a member stores a lot of strain energy ($5 se > v$), the term is positive, and the cross-section grows, whereas if the member carries little energy relative to its volume ($5 se < v$), it becomes negative, and the member is thinned. The factors se^2 and $(5 se^2 - se v - v)^2$ set how big the area change should be. They make the change large when a member is highly stressed and thin and small when the member already has a good balance between strain energy and volume.

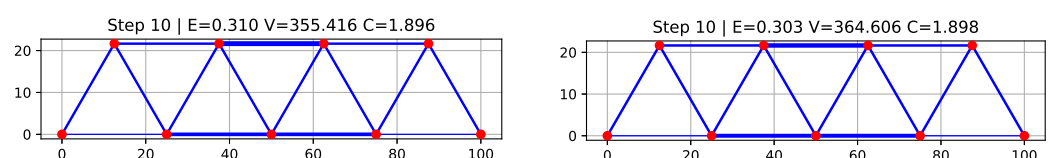


Figure 10. Cont.

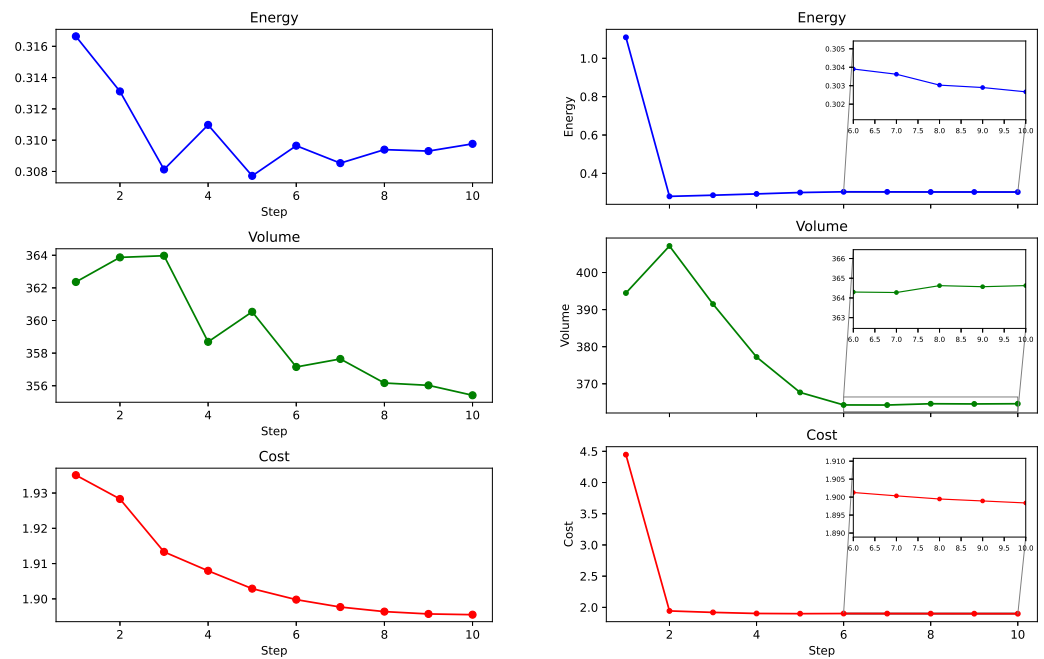


Figure 10. Metrics related to the highest reward structures obtained by the edge-only method after 10 development steps. (Left): GNN controller. (Right): CGP controller. (Top row): best performing structure. (Second row): change in strain energy E through devo. (Third row): change in volume V through devo. (Fourth row): change in cost $C = E + V$ through devo.

5.2. Node-Only Results

Figure 11 shows the structure that achieved the highest reward across the runs for both the GNN and CGP controllers. As shown, both the GNN and CGP controllers followed almost the same trajectories. The total volume increased as a result of the node's relocation, which resulted in the extension of some edges. Conversely, strain energy exhibited a distinct trend and decreased throughout the developmental steps. The results indicate that the controller incorporated a minimal amount of material while achieving greater stiffness, demonstrating an effective trade-off. The CGP controller update rule for each node (x and y position) is

$$\begin{aligned}\Delta x &= x(2x - y(2x + y) + y)(3x - y(2x + y) + y) \\ \Delta y &= (y^2 - x^2)(y + 1)\end{aligned}\quad (14)$$

The controller preserves truss symmetry, as shown by the variable x in the Δx equation, resulting in zero horizontal displacement for any node on the vertical axis ($x = 0$). However, on the vertical axis, because $y + 1 > 0$ is positive for all vertices, the sign of the update is controlled by $y^2 - x^2$. Nodes with $y^2 > x^2$ (located near the centre, where x is small) move upward, whereas nodes with $y^2 < x^2$ (near the supports, where x is large) move downward. Consequently, the initially flat top chord arches after 10 development steps (Figure 11), with the centre nodes elevated and the outer nodes dropped.

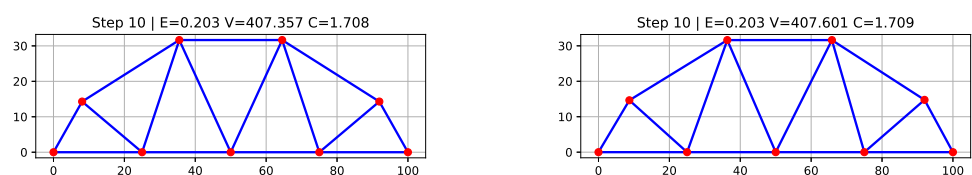


Figure 11. Cont.

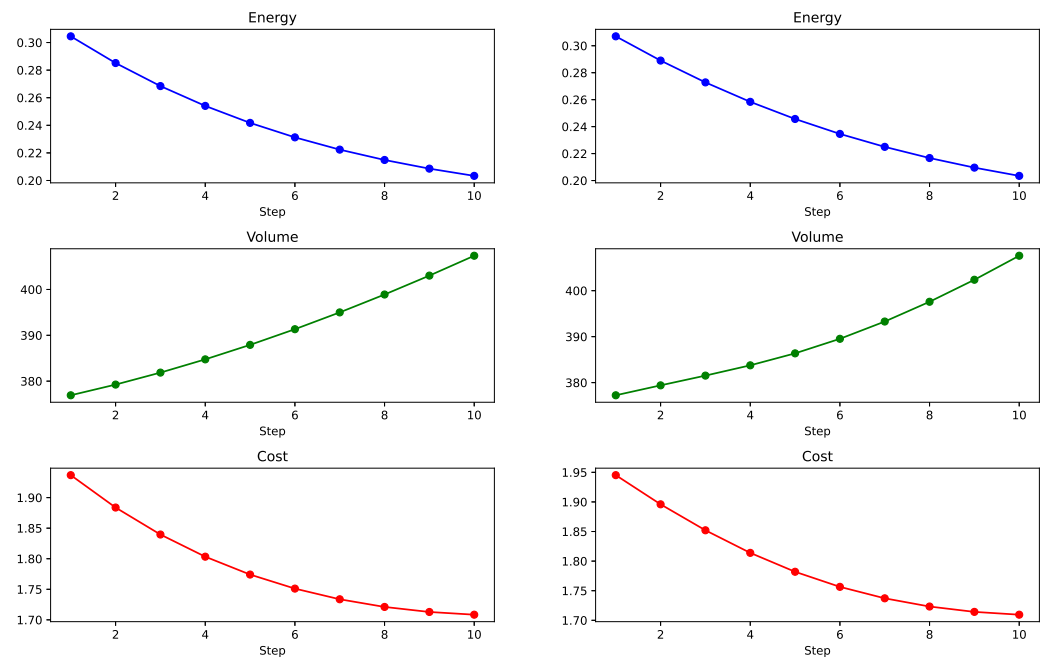


Figure 11. Metrics related to the highest reward structures obtained by the node-only method after 10 development steps. **(Left):** GNN controller. **(Right):** CGP controller. Rows as described in Figure 10.

5.3. Node–Edge Results

Figure 12 illustrates the way in which the node–edge controllers modify the truss structure. The CGP implements a considerable first adjustment, reducing both the strain energy and volume; however, it subsequently shows minimal improvements. The GNN controller, by comparison, decreases the energy and volume more gradually while maintaining this trend across all development steps, ultimately resulting in a lower overall cost. Therefore, Figure 12 demonstrates a potential trade-off: CGP is beneficial when only a limited number of development steps is feasible, whereas the GNN provides superior performance with long runs (development steps).

In addition, based on Equation (15), the coupled CGP controllers applied one rule to each edge (regarding one output) and two rules to each node in each development step, as in the following equation:

$$\begin{aligned}\Delta A &= 2se^2(3se - v) \\ \Delta x &= xy \quad \Delta y = y^2 - x^2\end{aligned}\tag{15}$$

Regarding the edge update ΔA , the term $2se^2$ enhances the updates for highly stressed members, while the linear term $3se - v$ determines the sign (positive or negative). The edges whose strain energy is higher than one-third of their volume become thicker, while the rest become thinner. This is the same “energy-to-volume balance” logic that was seen in the edge-only rule (Equation (13)), where $5se - v$ played the same role with a different coefficient. Regarding the vertex rules, they were the same as in the node-only experiment, with $\Delta y = y^2 - x^2$ raising the centre nodes and lowering the outer ones, making an arch shape.

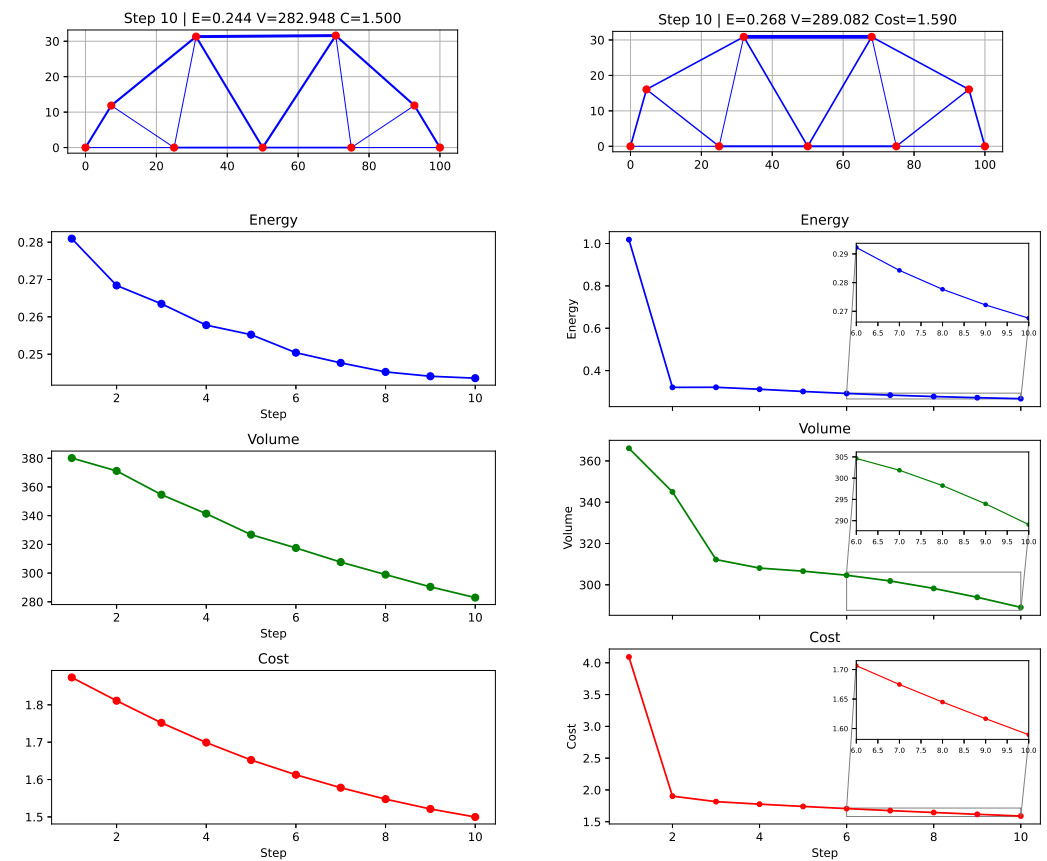


Figure 12. Metrics related to the highest reward structures obtained by the node-edge method after 10 development steps. (Left): GNN controller. (Right): CGP controller. Rows as described in Figure 10.

5.4. Comparison Between GNN and CGP

Table 6 and Figure 13 depict the best reward obtained from 15 independent runs for each CGP and GNN controller. A clear trend is evident: expanding the control space from the edge method to the node method and ultimately to the combined node-edge method consistently increases the mean reward for both the CGP (0.10 \rightarrow 0.39) and GNN (0.10 \rightarrow 0.43) controllers. The GNN-based node-edge method has the highest reward score (0.50), but it also has the widest variance, revealing irregular stagnation in the local minimum. In contrast, the GNN-based node-edge variations and all three CGP methods have less variance, producing highly reproducible results. Overall, it is shown that a trade-off exists between performance (reward) and consistency in the GNN and CGP controllers. Furthermore, the CGP-based approaches are integrated as a white box, allowing the expert to interpret modifications based on the acquired equations. The full development trajectories from the 15 runs of the best CGP and GNN controllers (for each method) are presented in Appendix A and Appendix B, respectively. These figures illustrate the evolution of the truss during all 10 developmental stages for the edge-only, node-only and combined node-edge methodologies, demonstrating how each strategy incrementally transforms the truss structure.

Table 6. Summary statistics of best rewards achieved over 15 runs per method.

Method	Num_Runs	Mean	Std	Min	Max
cgp_edge	15	0.098	0.003	0.094	0.102
cgp_node	15	0.282	0.005	0.277	0.291
cgp_node_edge	15	0.389	0.021	0.364	0.447

Table 6. Cont.

Method	Num_Runs	Mean	Std	Min	Max
gnn_edge	15	0.103	0.002	0.095	0.104
gnn_node	15	0.291	0.001	0.289	0.292
gnn_node_edge	15	0.425	0.075	0.291	0.500

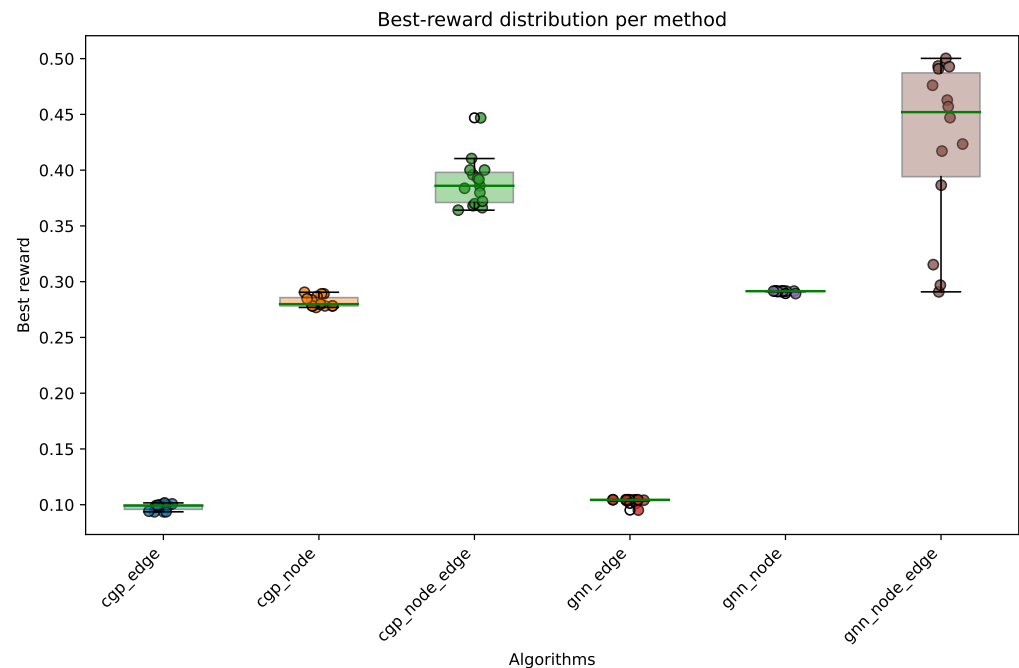


Figure 13. Distribution of the best reward obtained by each algorithm over 15 independent runs. The box shows the inter-quartile range, the horizontal line is the median, and the individual circles represent single runs. The circle's colour varies for each method.

5.5. Generalisation to a New Environment

One of the most important aspects of an indirect representation for generative design systems is the ability of the development rules to generalize to new problems that differ from the one on which they were evolved. To investigate this capability, two additional “seedling” truss benchmarks are created (Figure 14). All the load cases are applied simultaneously to the structure. The first environment is a multi-load, 9×15 truss (Figure 14a) in which three point loads of 10 kN, 17 kN and 17 kN are applied to the original 9-vertex, 15-edge Warren seedling introduced in Section 4. The second environment is a single-load 17×30 truss (Figure 14b) in which a single 17 kN point load is applied in the middle of the 17-vertex, 30-edge Warren seedling. For each of the two new seedling environments, both GRN controllers (GNN and CGP) are trained and compared with the best node–edge controllers previously evolved on the original single-load 9×15 truss. The controller structures and all Evo/Devo parameters are set as described in Section 4 wherever possible.

Table 7 shows the differences in the performance of the trained and frozen (untrained) GNN/CGP controllers on the two new seedling benchmarks. In Environment #1, the 9×15 truss with three point loads, both the trained GNN and CGP controllers perform better than the untrained controllers. The trained GNN lowers the cost mainly by decreasing the volume, while the trained CGP achieves the best cost by largely reducing the strain energy. In addition, the gap between the trained and untrained GNNs is smaller than that between the CGPs, showing that the untrained GNN works better. In Environment #2, the larger 17×30 truss with a single mid-span load, the outputs of the trained and untrained controllers are more similar. Although the untrained GNN outperforms the untrained CGP,

after training, the CGP again delivers the lowest cost, just as in Environment #1. These results illustrate that the generalisation characteristics of both the CGP and GNN controllers vary when changing the structure and parameters of the environments. Figures 15–18 show the final truss geometries and the evolution of energy, volume and cost over the ten developmental steps.

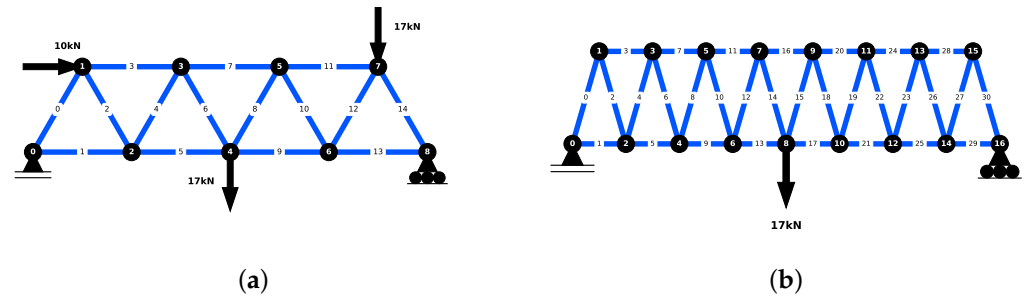


Figure 14. Seedling truss used for generalisation experiments: (a) three point loads with 9 vertices and 15 edges; (b) single central load with 17 vertices and 30 edges. Numbers inside the circles denote node indices, while numbers along the edge members denote edge indices.

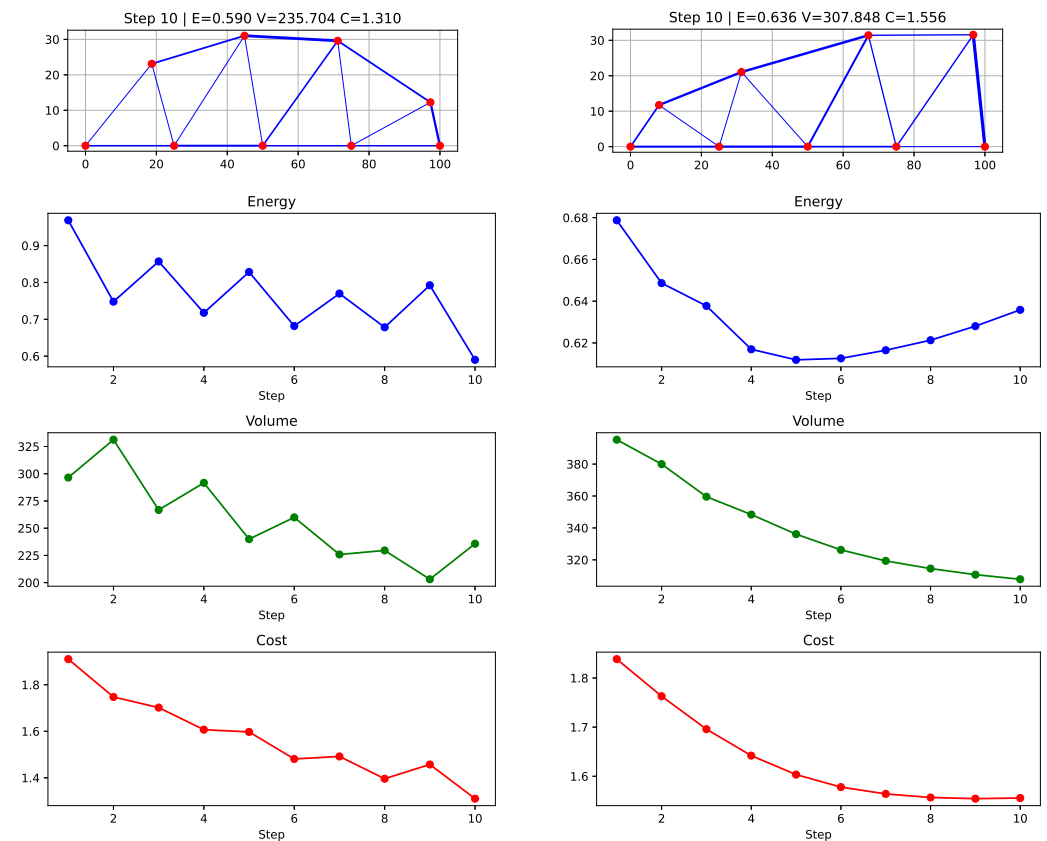


Figure 15. GNN behaviour in Environment #1 (three loads and 9×15 truss). (Left): trained controller; (right): untrained controller. (Top plots) show the final geometry after ten devo steps with strain energy E , volume V and cost. (Lower plots) trace E , V and cost over the ten devo steps.

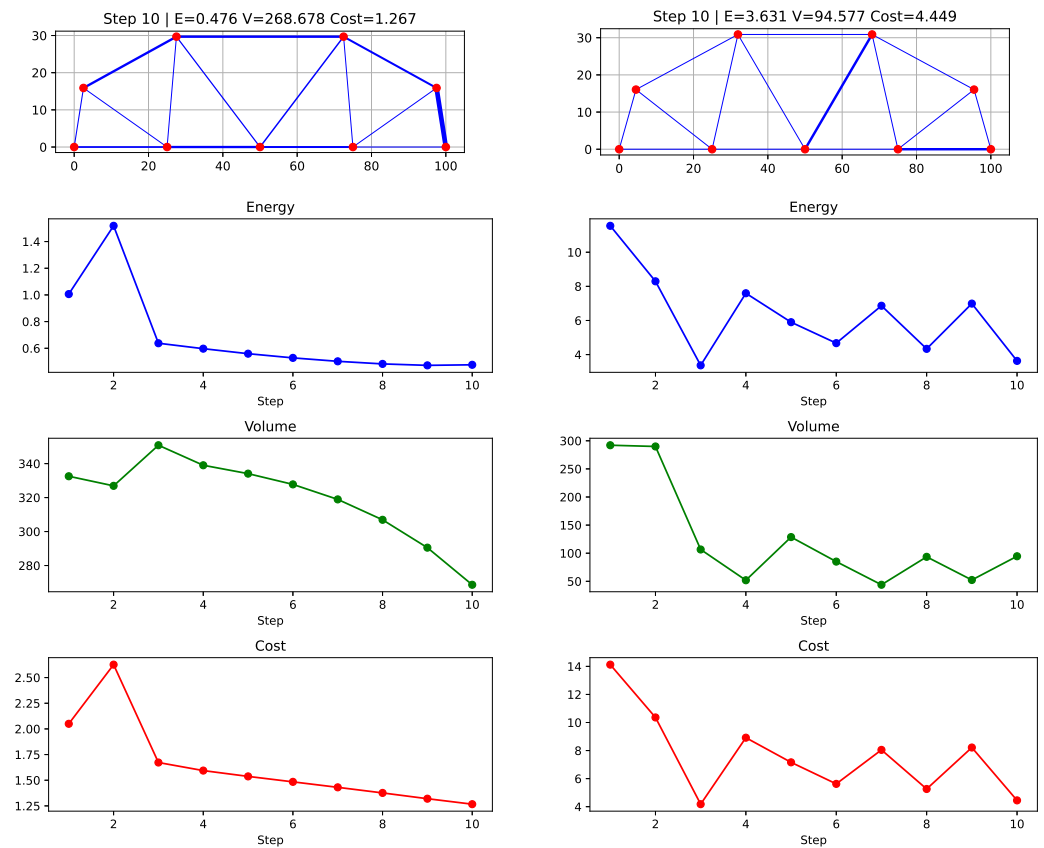


Figure 16. CGP behaviour in Environment #1 (three loads and 9×15 truss). (Left): trained controller; (right): untrained controller. (Top plots) show the final geometry after ten devo steps with strain energy E , volume V and cost. (Lower plots) trace E , V and cost over the ten devo steps.

Table 7. Performance of GNN and CGP controllers on the two new seedling benchmarks for the generalisation test.

Seedling	GRN	Criteria	Trained Controller	Untrained Controller
Environment #1	GNN	Energy	0.590	0.6358
		Volume	235.704	307.848
		Cost	1.310	1.556
	CGP	Energy	0.476	3.631
		Volume	268.678	94.577
		Cost	1.267	4.449
Environment #2	GNN	Energy	1.381	1.732
		Volume	529.189	642.085
		Cost	1.469	1.816
	CGP	Energy	1.201	1.459
		Volume	585.465	518.154
		Cost	1.440	1.500

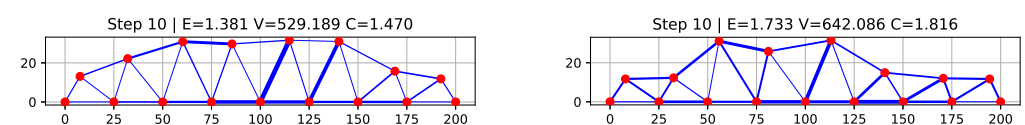


Figure 17. Cont.

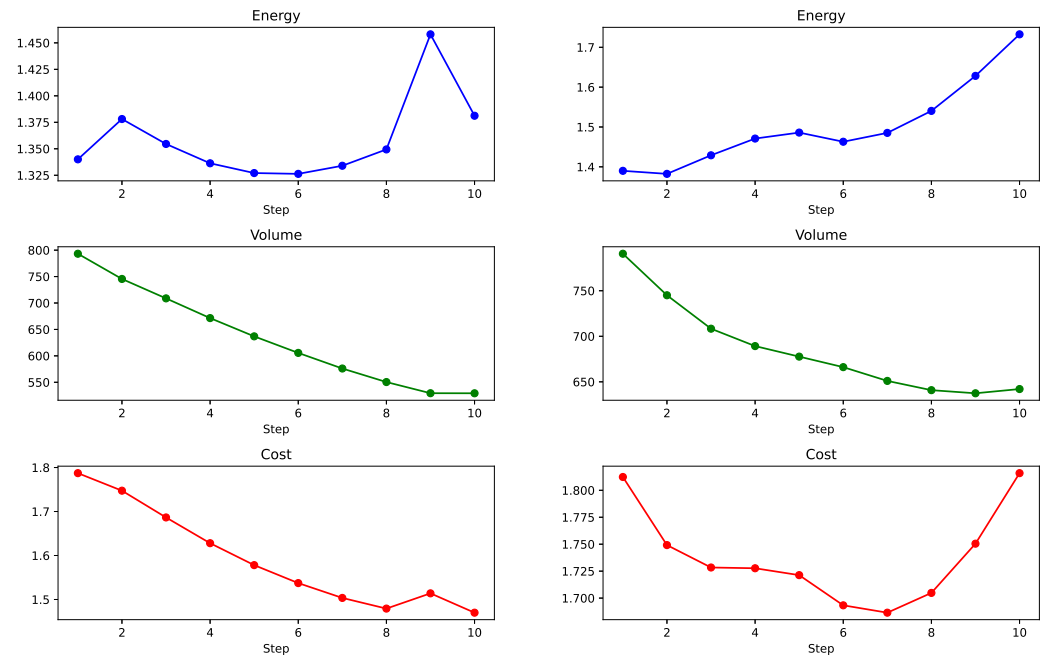


Figure 17. GNN behaviour in Environment #2 (one load and 17×30 truss). (Left): trained controller; (right): untrained controller. (Top plots) show the final geometry after ten devo steps with strain energy E , volume V and cost. (Lower plots) trace E , V and cost over the ten devo steps.

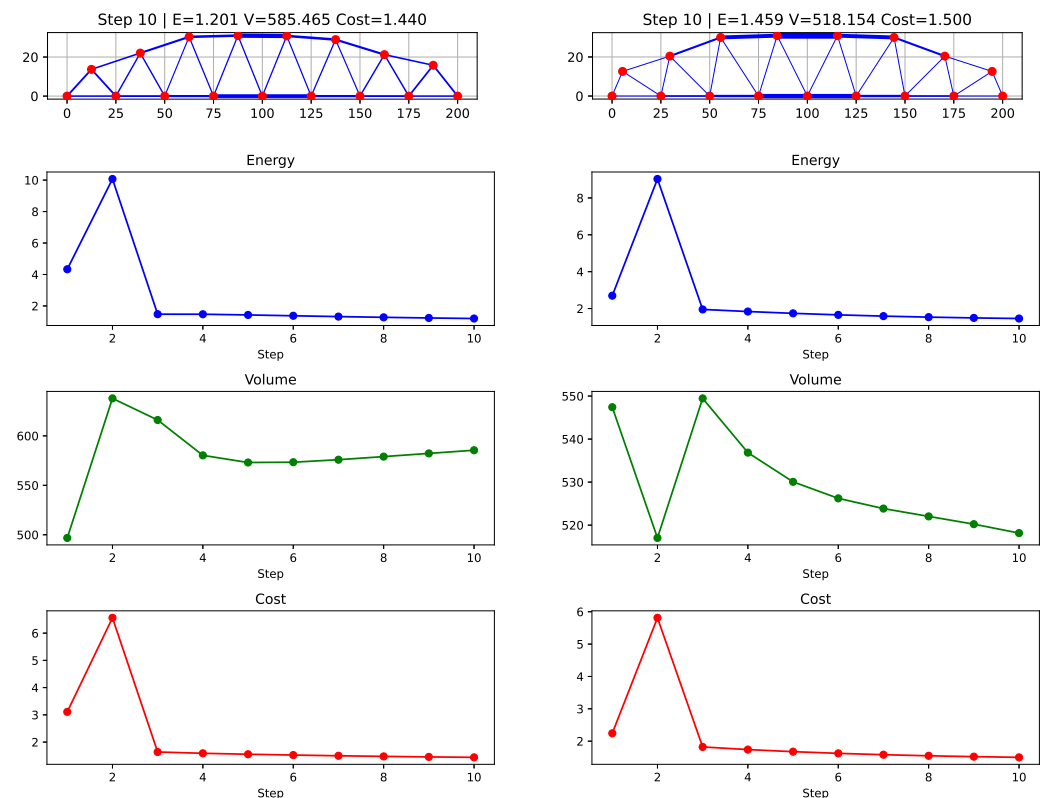


Figure 18. CGP behaviour in Environment #2 (one load and 17×30 truss). (Left): trained controller; (right): untrained controller. (Top plots) show the final geometry after ten devo steps with strain energy E , volume V and cost. (Lower plots) trace E , V and cost over the ten devo steps.

5.6. Comparison with Other Topology Optimisation Methods

To benchmark the GNN-based truss against established continuum methods, the topology optimisation study in [54] is referred to. This work utilises SolidWorks, ANSYS Mechan-

ical and ABAQUS software (<https://www.3ds.com/products/simulia/abaqus> (accessed on 11 June 2025)) to optimise three structures, including a small bridge with dimensions of $2555 \times 500 \times 600$ mm that carries a 800 kN distributed bottom load. The weight of the GNN-based truss is estimated and compared to the weight results derived by Solidworks, ANSYS and ABAQUS using the seedling parameters in Table 8 extracted from [54]. Table 9 shows the weights of the small bridge after optimisation compared to those of the GNN-based truss. For the GNN-based truss, the initial cross-sectional area of 75 mm is used. The resulting weight of the GNN-based truss is marginally comparable to the weights of the three methods reported in [54].

Table 8. Seedling and material parameters used in the GNN truss run.

Parameters	Value
Span	2500.0 mm
Height	500.0 mm
Young's modulus	210.0 GPa
Density ρ	7700.0 kg/m ³
Poisson ratio ν	0.28

As shown in Figure 19, after ten development steps, the GNN-based truss has a total volume of $V = 538,357.169$ mm³. Substitution into Equation (16) with $\rho = 7700$ kg/m³ and $g = 9.81$ m/s² gives a final self-weight of $W = 40.66$ N.

$$\begin{aligned}
 V &= 538,357.169 \text{ mm}^3, \quad \rho = 7700 \text{ kg/m}^3, \quad g = 9.81 \text{ m/s}^2, \\
 m &= V \rho \times 10^{-9} = 4.14535 \text{ kg}, \\
 W &= m g = 40.66 \text{ N}.
 \end{aligned}
 \tag{16}$$

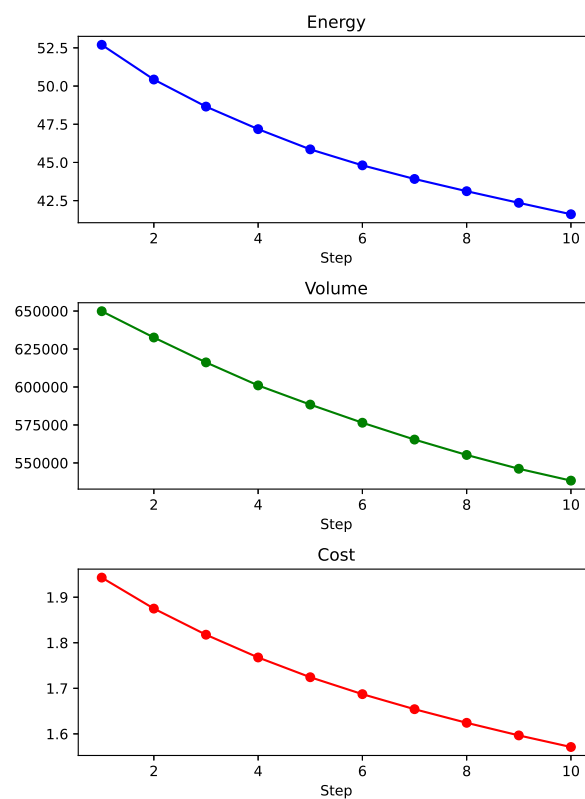


Figure 19. GNN-based truss outputs for energy, volume and cost after ten development steps based on the parameters in [54].

Table 9. Optimised weights of the small bridge and GNN-based truss.

Solver/Method	Final Weight
SolidWorks	49.7
ANSYS	55.5
ABAQUS	43.26
GNN-based truss	40.666

6. Conclusions and Future Works

A new bioinspired generative design system was proposed that utilizes the concepts of previous indirect representation approaches such as cellular representations, evolutionary development, gene regulatory networks and state-based models. The approach involves representing the design as a heterogeneous graph, where the nodes represent different cell types. These cells make growth decisions via a stateful GRN to minimize some global objectives. The novelty lies in the genetic search algorithm established to evolve graph neural networks and the Cartesian genetic programming that can learn re-deployable developmental rules for engineering problems. These approaches were tested on truss structures and were shown to be able to optimise designs in terms of shape and size by changing the vertex positions and edge cross-sectional areas of the truss structures. These growth mechanisms were able to be used individually or in unison to provide better performance. The results revealed that, although GNN exhibited superior performance in terms of accumulated reward compared to CGP, the outcomes were nearly identical. However, CGP provided the opportunity to interpret the evolution and identify the pathway that correlated inputs with outputs in truss structures via their output equations.

The growth mechanisms investigated in this paper affected the shape and size of the design structure. A future avenue of work would be to investigate methods for exploring the design space via a topology change. One way of doing this would be by using a cellular division growth mechanism, or the topology of the initial seedling could also be altered. As only truss structures were used, the proposed system should be applied to other engineering problems and structures to highlight that it is not limited to these structures. In the experiments undertaken in Section 5, a fixed number of devo steps was used. This number was selected empirically after preliminary trials showed a reasonable balance between additional performance gains and computational cost. In the future, more systematic research is required to deterministically define an appropriate value for this parameter. Research could also look into organism behaviour in detail to develop a criterion for when it should stop growing, similar to how the end of a sentence can be predicted in natural language processing (NLP). To achieve a more thorough benchmark, the suggested generative design framework must be assessed in conjunction with proven topology optimisation methods, thus clarifying its performance relative to that of standard engineering software. More work is also needed to improve the generalisability of the pre-trained seeds so that a user would only need to run a development loop for a new environment. Performance could possibly be improved by training the GRN on a larger dataset of different environments. Future work could also look into using multi-/many-objective evolutionary algorithms to generate a diverse range of solutions from a pre-trained seed. This cellular representation is agnostic and could be used to represent other entities in a supply chain or manufacturing process, for example.

Author Contributions: Conceptualization, A.M.T. and M.P.; Methodology, A.C., I.F., S.J.H., E.B. and P.G.; Software, A.C., I.F., S.J.H., E.B., P.G. and F.T.-J.; Formal analysis, F.T.-J.; Visualization, F.T.-J.; Validation, F.T.-J.; Writing—original draft preparation, A.C. and F.T.-J.; Writing—review and editing, S.J.H., E.B., P.G., A.M.T., M.P., T.R. and F.T.-J.; Supervision, A.M.T. and M.P.; Funding acquisition, A.M.T.; Project administration, A.M.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by EPSRC programme Grant Ref EP/V007335/1, “RIED: Re-Imagining Engineering Design”.

Data Availability Statement: The software and data can be found at the GitLab repository <https://gitlab.com/riedproject> (accessed on 11 June 2025).

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

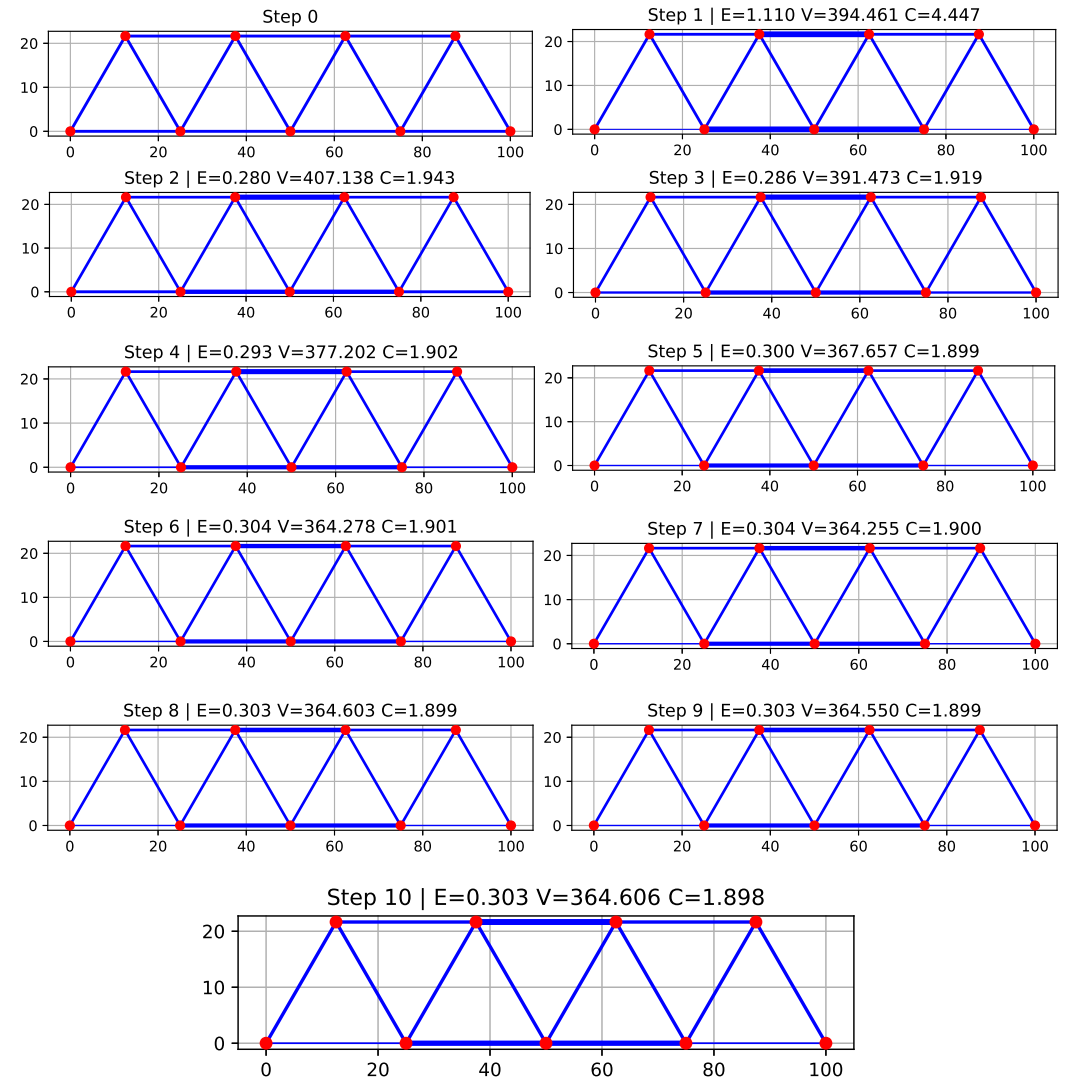


Figure A1. Development sequence over 10 development steps for the best edge-based CGP controller.

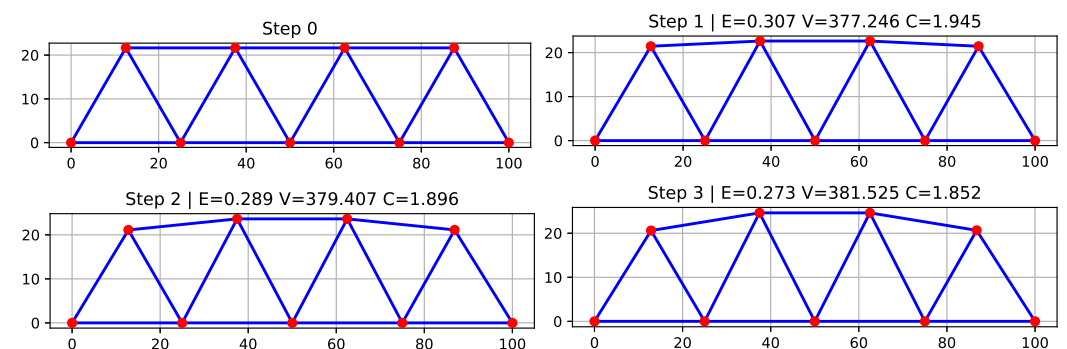


Figure A2. Cont.

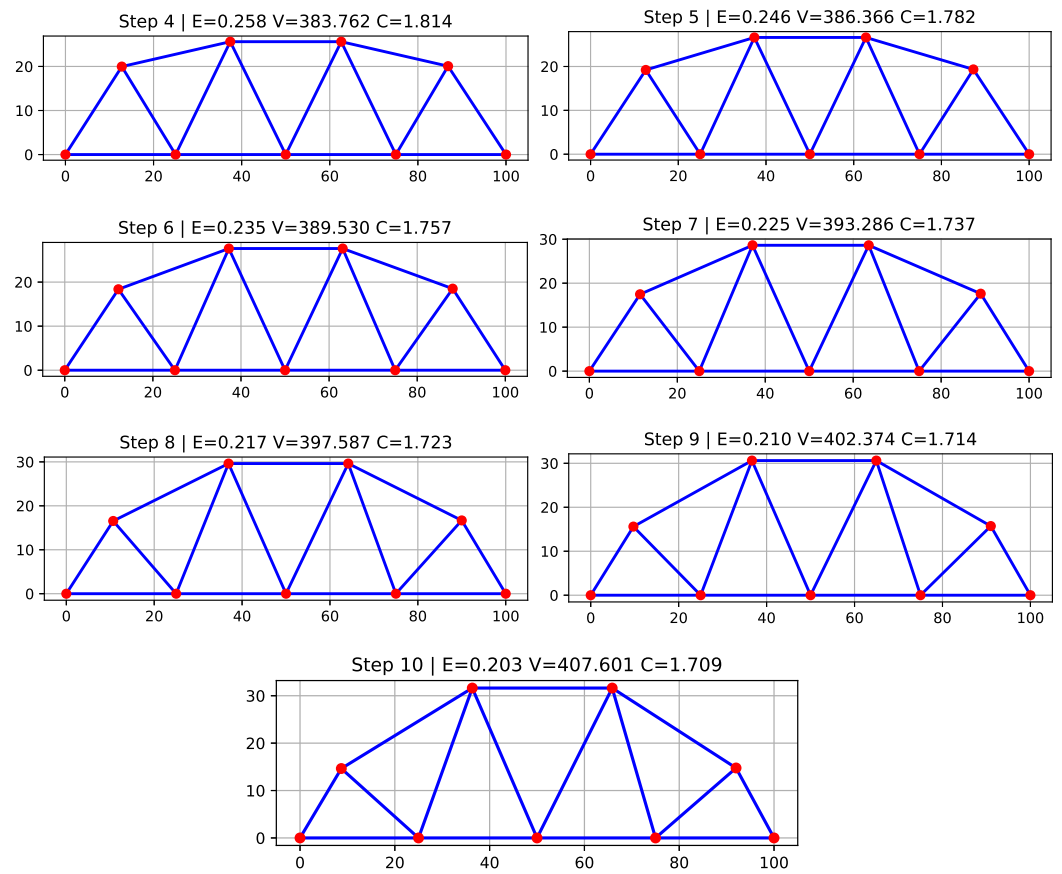


Figure A2. Development sequence over 10 development steps for the best node-based CGP controller.

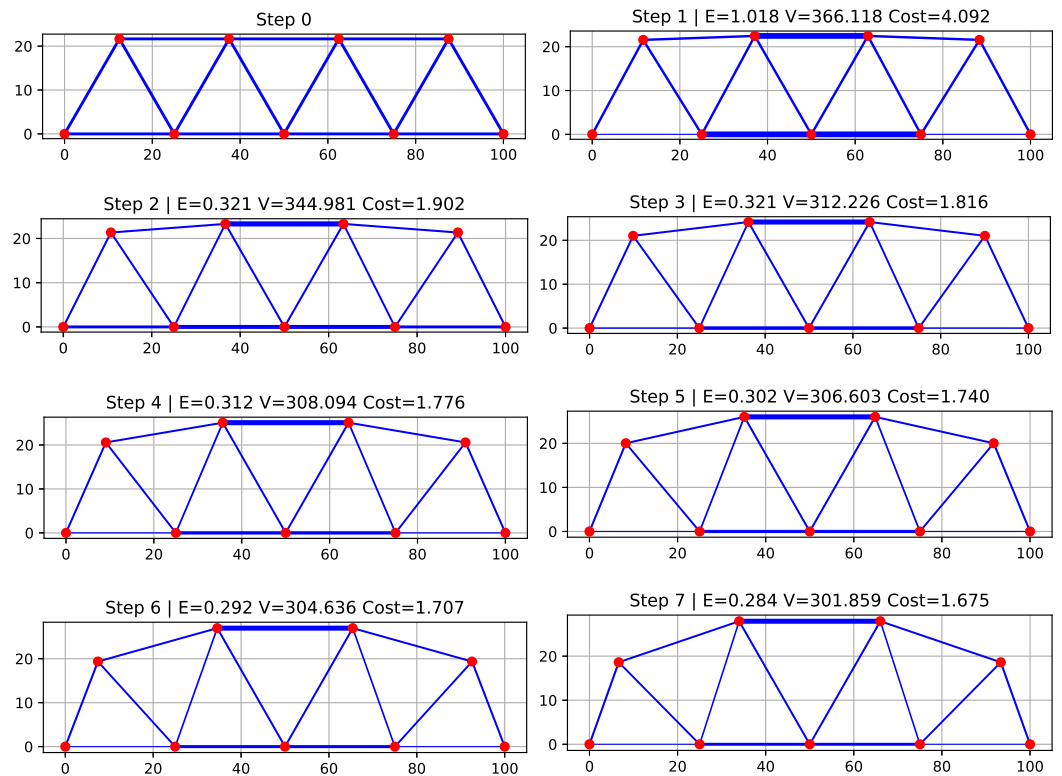


Figure A3. Cont.

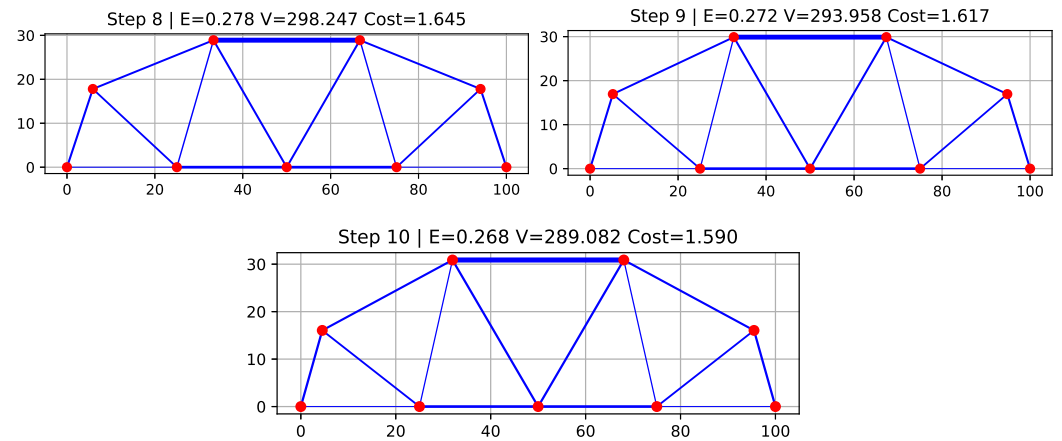


Figure A3. Development sequence over 10 development steps for the best the node-edge-based CGP controller.

Appendix B

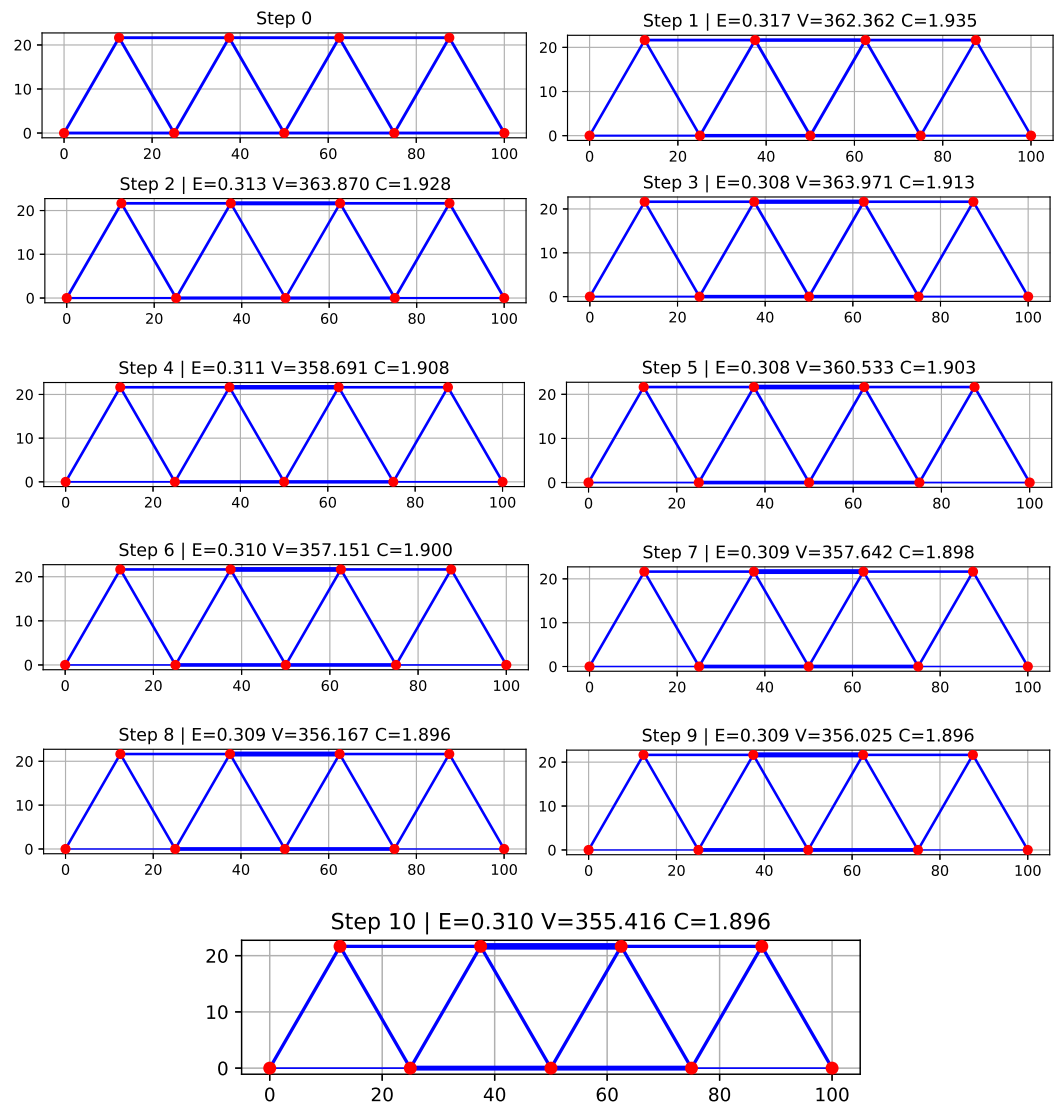


Figure A4. Development sequence over 10 development steps for the best edge-based GNN controller.

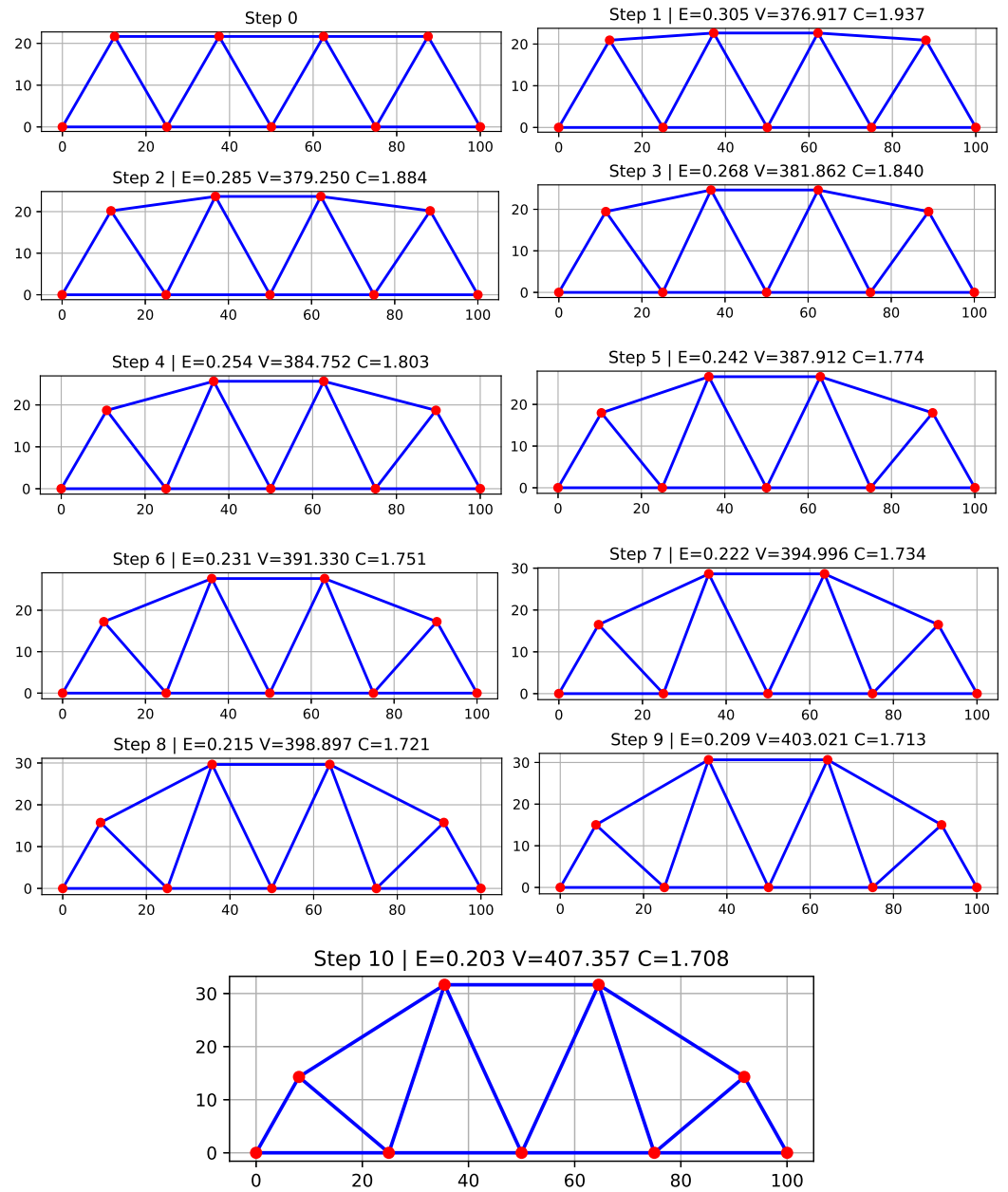


Figure A5. Development sequence over 10 development steps for the best node-based GNN controller.

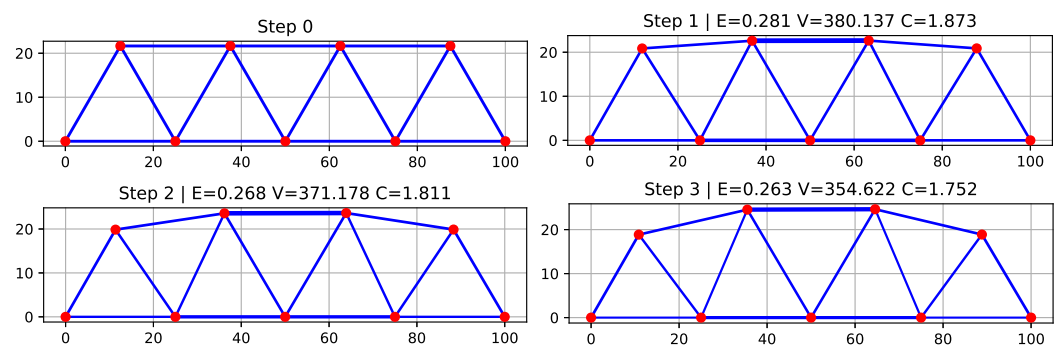


Figure A6. Cont.

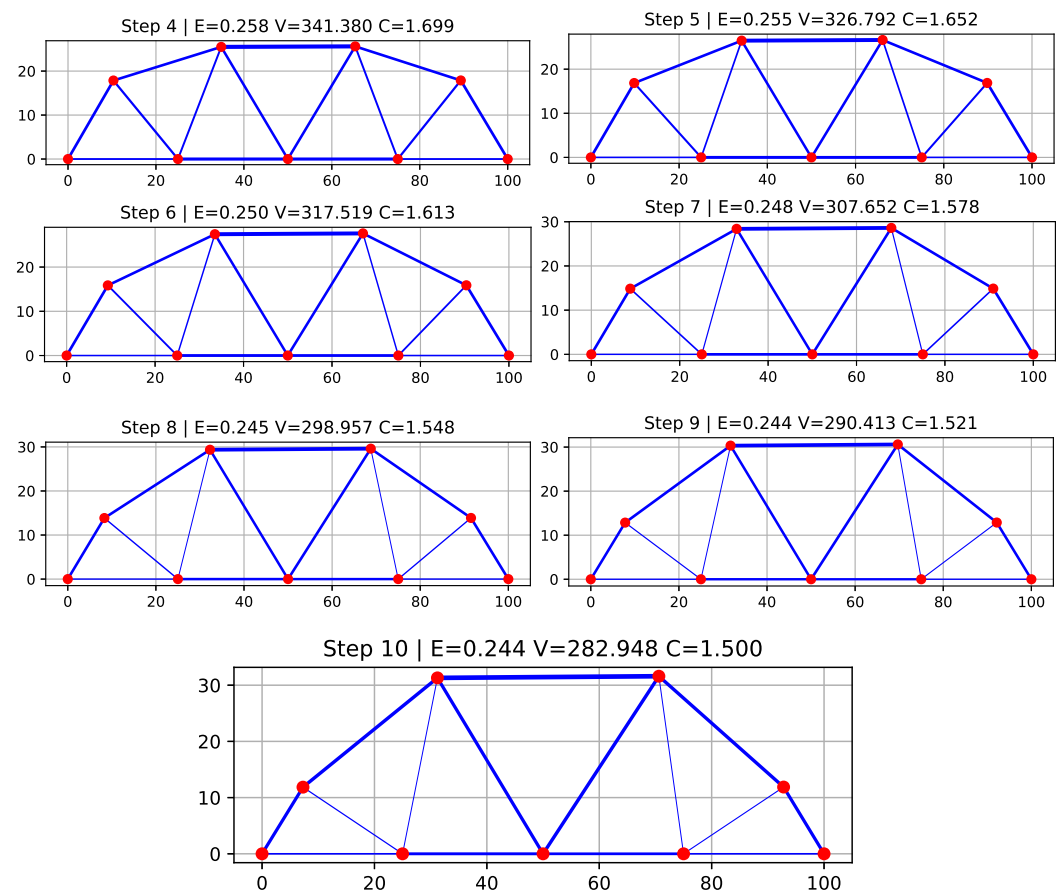


Figure A6. Development sequence over 10 development steps for the best the node-edge-based GNN controller.

References

1. Traversari, R.; Goedhart, R.; Schraagen, J.M. Process simulation during the design process makes the difference: Process simulations applied to a traditional design. *HERD Health Environ. Res. Des. J.* **2013**, *6*, 58–76. [CrossRef]
2. Han, S.; Lee, S.; Fard, M.G.; Pena-Mora, F. Modeling and representation of non-value adding activities due to errors and changes in design and construction projects. In Proceedings of the Winter Simulation Conference, Washington, DC, USA, 9–12 December 2007; pp. 2082–2089.
3. McKnight, M. Generative Design: What it is? How is it being used? Why it's a game changer. *KnE Eng.* **2017**, *2*, 176–181. [CrossRef]
4. Watson, M.; Leary, M.; Brandt, M. Generative design of truss systems by the integration of topology and shape optimisation. *Int. J. Adv. Manuf. Technol.* **2022**, *118*, 1165–1182. [CrossRef]
5. Zhang, W.; Huang, F. An introductory overview to bio-inspired generative design. *J. Mech. Sci. Technol.* **2023**, *37*, 1–5. [CrossRef]
6. Steiner, T. Artificial Evolutionary Development. 2010. Available online: <https://scispace.com/pdf/artificial-evolutionary-development-4wgppb4net.pdf> (accessed on 24 July 2025).
7. Lingyun, W.; Mei, Z.; Guangming, W.; Guang, M. Truss optimization on shape and sizing with frequency constraints based on genetic algorithm. *Comput. Mech.* **2005**, *35*, 361–368. [CrossRef]
8. Hajela, P.; Lee, E. Genetic algorithms in truss topological optimization. *Int. J. Solids Struct.* **1995**, *32*, 3341–3357. [CrossRef]
9. Sigmund, O. On the usefulness of non-gradient approaches in topology optimization. *Struct. Multidiscip. Optim.* **2011**, *43*, 589–596. [CrossRef]
10. Bendsøe, M.P.; Sigmund, O. Material interpolation schemes in topology optimization. *Arch. Appl. Mech.* **1999**, *69*, 635–654. [CrossRef]
11. Wang, M.Y.; Wang, X.; Guo, D. A level set method for structural topology optimization. *Comput. Methods Appl. Mech. Eng.* **2003**, *192*, 227–246. [CrossRef]
12. Aulig, N.; Olhofer, M. Evolutionary computation for topology optimization of mechanical structures: An overview of representations. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC), Vancouver, BC, Canada, 24–29 July 2016; pp. 1948–1955.

13. Guirguis, D.; Aulig, N.; Picelli, R.; Zhu, B.; Zhou, Y.; Vicente, W.; Iorio, F.; Olhofer, M.; Matusik, W.; Coello, C.A.C.; et al. Evolutionary black-box topology optimization: Challenges and promises. *IEEE Trans. Evol. Comput.* **2019**, *24*, 613–633. [\[CrossRef\]](#)
14. Wang, Z.; Zhang, Y.; Bernard, A. A constructive solid geometry-based generative design method for additive manufacturing. *Addit. Manuf.* **2021**, *41*, 101952. [\[CrossRef\]](#)
15. Schoenauer, M. Shape representations and evolution schemes. In Proceedings of the Fifth Annual Conference on Evolutionary Programming, San Diego, CA, USA, 29 February–3 March 1996.
16. Feng, F.; Xiong, S.; Liu, Z.; Xian, Z.; Zhou, Y.; Kobayashi, H.; Kawamoto, A.; Nomura, T.; Zhu, B. Cellular topology optimization on differentiable Voronoi diagrams. *Int. J. Numer. Methods Eng.* **2023**, *124*, 282–304. [\[CrossRef\]](#)
17. Bujny, M.; Aulig, N.; Olhofer, M.; Duddeck, F. Hybrid evolutionary approach for level set topology optimization. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC), Vancouver, BC, Canada, 24–29 July 2016; pp. 5092–5099.
18. Wang, S.; Tai, K. Graph representation for structural topology optimization using genetic algorithms. *Comput. Struct.* **2004**, *82*, 1609–1622. [\[CrossRef\]](#)
19. Tai, K.; Akhtar, S. Structural topology optimization using a genetic algorithm with a morphological geometric representation scheme. *Struct. Multidiscip. Optim.* **2005**, *30*, 113–127. [\[CrossRef\]](#)
20. Wang, N.; Tai, K. Target matching problems and an adaptive constraint strategy for multiobjective design optimization using genetic algorithms. *Comput. Struct.* **2010**, *88*, 1064–1076. [\[CrossRef\]](#)
21. Wang, S.; Tai, K. Bar-system representation for topology optimization using genetic algorithms. *Eng. Comput.* **2005**, *22*, 206–231. [\[CrossRef\]](#)
22. Sauter, M.; Kress, G.; Giger, M.; Ermanni, P. Complex-shaped beam element and graph-based optimization of compliant mechanisms. *Struct. Multidiscip. Optim.* **2008**, *36*, 429–442. [\[CrossRef\]](#)
23. Ahmed, F.; Bhattacharya, B.; Deb, K. Constructive solid geometry based topology optimization using evolutionary algorithm. In *Proceedings of Seventh International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2012)*; Springer: Berlin/Heidelberg, Germany, 2013; Volume 1, pp. 227–238.
24. Pandey, A.; Datta, R.; Bhattacharya, B. Topology optimization of compliant structures and mechanisms using constructive solid geometry for 2-d and 3-d applications. *Soft Comput.* **2017**, *21*, 1157–1179. [\[CrossRef\]](#)
25. Guo, T.; Lohan, D.J.; Cang, R.; Ren, M.Y.; Allison, J.T. An indirect design representation for topology optimization using variational autoencoder and style transfer. In Proceedings of the AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Kissimmee, FL, USA, 8–12 January 2018; p. 0804.
26. Jang, S.; Yoo, S.; Kang, N. Generative design by reinforcement learning: Enhancing the diversity of topology optimization designs. *Comput.-Aided Des.* **2022**, *146*, 103225. [\[CrossRef\]](#)
27. Riegler, G.; Osman Ulusoy, A.; Geiger, A. Octnet: Learning deep 3d representations at high resolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 3577–3586.
28. Price, M.; Zhang, W.; Friel, I.; Robinson, T.; McConnell, R.; Nolan, D.; Kilpatrick, P.; Barbhuiya, S.; Kyle, S. Generative design for additive manufacturing using a biological development analogy. *J. Comput. Des. Eng.* **2022**, *9*, 463–479. [\[CrossRef\]](#)
29. Pedro, H.T.C.; Kobayashi, M.H. On a cellular division method for topology optimization. *Int. J. Numer. Methods Eng.* **2011**, *88*, 1175–1197. [\[CrossRef\]](#)
30. Steiner, T.; Jin, Y.; Sendhoff, B. A cellular model for the evolutionary development of lightweight material with an inner structure. In Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, Atlanta, GA, USA, 12–16 July 2008; pp. 851–858.
31. Steiner, T.; Trommler, J.; Brenn, M.; Jin, Y.; Sendhoff, B. Global shape with morphogen gradients and motile polarized cells. In Proceedings of the IEEE Congress on Evolutionary Computation, Trondheim, Norway, 18–21 May 2009; pp. 2225–2232.
32. Stanley, K.O.; Clune, J.; Lehman, J.; Miikkulainen, R. Designing neural networks through neuroevolution. *Nat. Mach. Intell.* **2019**, *1*, 24–35. [\[CrossRef\]](#)
33. Aulig, N.; Olhofer, M. Evolutionary generation of neural network update signals for the topology optimization of structures. In Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation, Amsterdam, The Netherlands, 6–10 July 2013; pp. 213–214.
34. Stanley, K.O.; Miikkulainen, R. Evolving neural networks through augmenting topologies. *Evol. Comput.* **2002**, *10*, 99–127. [\[CrossRef\]](#)
35. Cheney, N.; Ritz, E.; Lipson, H. Automated vibrational design and natural frequency tuning of multi-material structures. In Proceedings of the Annual Conference on Genetic and Evolutionary Computation, Vancouver, Canada, 12–16 July 2014; pp. 1079–1086.
36. Stanley, K.O.; D’Ambrosio, D.B.; Gauci, J. A hypercube-based encoding for evolving large-scale neural networks. *Artif. Life* **2009**, *15*, 185–212. [\[CrossRef\]](#)
37. Hickinbotham, S.; Dubey, R.; Friel, I.; Colligan, A.; Price, M.; Tyrrell, A. Evolving design modifiers. In Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI), Singapore, 4–7 December 2022; pp. 1052–1058.

38. Miller, J.; Turner, A. Cartesian genetic programming. In Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation, Madrid, Spain, 11–15 July 2015; pp. 179–198.
39. Miller, J.F.; Job, D.; Vassilev, V.K. Principles in the evolutionary design of digital circuits—Part I. *Genet. Program. Evolvable Mach.* **2000**, *1*, 7–35. [[CrossRef](#)]
40. Miller, J.F. Cartesian genetic programming: Its status and future. *Genet. Program. Evolvable Mach.* **2020**, *21*, 129–168. [[CrossRef](#)]
41. Harding, S.; Miller, J.F. Evolution of robot controller using cartesian genetic programming. In *Genetic Programming, Proceedings of the 8th European Conference, Lausanne, Switzerland, 30 March–1 April 2005*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 62–73.
42. Cao, W.; Robinson, T.; Hua, Y.; Boussuge, F.; Colligan, A.R.; Pan, W. Graph representation of 3D CAD models for machining feature recognition with deep learning. In Proceedings of the International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Online, 17–19 August 2020; Volume 84003, p. V11AT11A003.
43. Colligan, A.R.; Robinson, T.T.; Nolan, D.C.; Hua, Y.; Cao, W. Hierarchical cadnet: Learning from b-reps for machining feature recognition. *Comput.-Aided Des.* **2022**, *147*, 103226. [[CrossRef](#)]
44. Jayaraman, P.K.; Sanghi, A.; Lambourne, J.G.; Willis, K.D.; Davies, T.; Shayani, H.; Morris, N. Uv-net: Learning from boundary representations. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 20–25 June 2021; pp. 11703–11712.
45. Lambourne, J.G.; Willis, K.D.; Jayaraman, P.K.; Sanghi, A.; Meltzer, P.; Shayani, H. Brepnet: A topological message passing system for solid models. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 20–25 June 2021; pp. 12773–12782.
46. Jones, B.; Hildreth, D.; Chen, D.; Baran, I.; Kim, V.G.; Schulz, A. Automate: A dataset and learning approach for automatic mating of cad assemblies. *ACM Trans. Graph.* **2021**, *40*, 1–18. [[CrossRef](#)]
47. Morris, C.; Ritzert, M.; Fey, M.; Hamilton, W.L.; Lenssen, J.E.; Rattan, G.; Grohe, M. Weisfeiler and leman go neural: Higher-order graph neural networks. In Proceedings of the AAAI conference on artificial intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 4602–4609.
48. Maas, A.L.; Hannun, A.Y.; Ng, A.Y. Rectifier nonlinearities improve neural network acoustic models. In Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, 16–21 June 2013; Volume 30, p. 3.
49. Buchanan Berumen, E.; Hickinbotham, S.J.; Dubey, R.; Friel, I.; Colligan, A.; Price, M.; Tyrrell, A. A quality diversity study in EvoDevo processes for engineering design. In Proceedings of the IEEE World Congress on Computational Intelligence, Yokohama, Japan, 30 June–5 July 2024.
50. Dubey, R.; Hickinbotham, S.; Colligan, A.; Friel, I.; Buchanan, E.; Price, M.; Tyrrell, A.M. Evolving novel gene regulatory networks for structural engineering designs. *Artif. Life* **2024**, *30*, 466–485. [[CrossRef](#)]
51. Mania, H.; Guy, A.; Recht, B. Simple random search of static linear policies is competitive for reinforcement learning. *Adv. Neural Inf. Process. Syst.* **2018**, *31*, 1805–1814.
52. Črepinšek, M.; Liu, S.H.; Mernik, M. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Comput. Surv.* **2013**, *45*, 1–33. [[CrossRef](#)]
53. Hussain, A.; Muhammad, Y.S. Trade-off between exploration and exploitation with genetic algorithm using a novel selection operator. *Complex Intell. Syst.* **2020**, *6*, 1–14. [[CrossRef](#)]
54. Tyflopoulos, E.; Steinert, M. A comparative study of the application of different commercial software for topology optimization. *Appl. Sci.* **2022**, *12*, 611. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.