Exploring the Adjacent Possible in Engineering Design via Graph Operation Programs

Babis Peteinarelis
School of Physics, Engineering and Technology
University of York
York, UK
kbc527@york.ac.uk

Simon Hickinbotham
School of Physics, Engineering and Technology
University of York
York, UK
simon.hickinbotham@york.ac.uk

Andy Tyrrell
School of Physics, Engineering and Technology
University of York
York, UK
andy.tyrrell@york.ac.uk

***Abstract*—Nature-inspired algorithms and mechanisms have been used for many years to assist the design of engineering structures. However, in most fitness landscapes in engineering, Evolutionary Algorithms converge and the result is accepted as-is. Design tasks on the other hand often prioritise novelty and diversity in solutions, hoping for innovation, not simply optimisation, needs that are satisfied by Generative Design methods usually at the cost of optimality, transparency or feasibility. Inspired by Kauffman's Adjacent Possible, this paper attempts to reimagine the popular concept in contexts that evolve structural paradigms, aiming to combine the glass-box rationale of design with the generative power of Evolutionary Methods. The presented formulation uses the graph space to represent structures and the formalism of Graph Automata to express transition rules that can reach adjacent structures based on a set of elementary graph operations. Sequences of such graph operations *(programs)* are evolved to reveal a set of fit adjacent structures. This evolutionary logic is demonstrated on a Multi-Objective Topology Optimisation scenario where a seed truss structure experiences environmental forces. The evolved programs transform the seed structure into a variety of truss designs that respond to localised strain in different ways in order to address trade-offs between the objectives. The paper illustrates the method on truss *(chassis)* design and discovers a range of Pareto-optimal adjacent designs within a program length radius, maintaining the main design intent.**

***Index Terms*—Evolutionary Computation, Generative Design, Engineering Design, Graph Operations, Graph Automata**

I. Introduction

The allure of evolutionary methods famously lies in their ability to produce unconventional solutions and navigate demanding search spaces. These properties that have been successfully used in many fields of engineering [1] are usually associated with optimisation tasks. However, not every problem in engineering can be simply reduced to an optimisation formulation, including fields that demand and prioritise innovation, such as *Structural Design*. In such areas, where the *diversity* and the *novelty* of solutions is rewarded, the evolutionary logic is superior to conventional approaches that dominate in optimisation, or even unsupervised intelligent approaches with formalisms that ignore *unknown unknowns* [2]. This transition from an optimisation task to a generative one in the evolution of structural paradigms requires a generalisable approach that emphasises the exploration of structural spaces without sacrificing exploitation. The pursuit of robust methods that embed these characteristics in design scenarios continues to be an active area of research. A large part of previous work in this area includes direct encodings and fixed-length genomic representations of structure that rearrange a standard set of *"building blocks"*. Morphogen Grammars [3] introduce morphogen gradients and use them to formulate context-free grammars that explore hierarchies of primitives in truss design optimisation.

Quality-Diversity (QD) approaches address the exploration-exploitation trade-off by combining performance with diversity metrics. For example, they have been used for Multi-Objective Optimisation in continuous control robotic tasks [4] and paired with grammatical encodings of equations to discover low mean curvature surfaces [5]. Defining explicit diversity measures is a well-established way to promote the discovery of niches: Novelty search [6] uses novelty measures in the behaviour space paired with local competition to evolve robotic organisms, while illumination algorithms like MAP-Elites [7] attempt to map the $N-$dimensional feature space by dissecting it into a grid of cells and searching for the highest performing solution in each cell. Operation-focused methods on the other hand do not evolve representations of structure, but regulate series of developmental steps that change the structure of interest, in which case, the primitives used are not structural motifs, but functions. Techniques like Cartesian Genetic Programming (CGP) [8] and representations like Gene Regulatory Networks (GRNs) [9] belong to this category, and have been shown to control the growth of engineering structures in an explainable way and with a performance comparable to neural methods such as Graph Neural Networks (GNNs) [10].

Despite the aforementioned progress, the step-by-step, domain-specific process of engineering design and the black-box, exploratory nature of Evo-Devo algorithms seem to remain unreconciled. The framework outlined in this paper attempts to combine the transparency and sequential logic of the design process with the generative power of evolutionary methods by introducing a structural design analogue of the *Adjacent Possible*. In the evolution of complex systems, the Adjacent Possible (originally introduced as a concept by Kauffman [11]) represents the set of all the possibilities

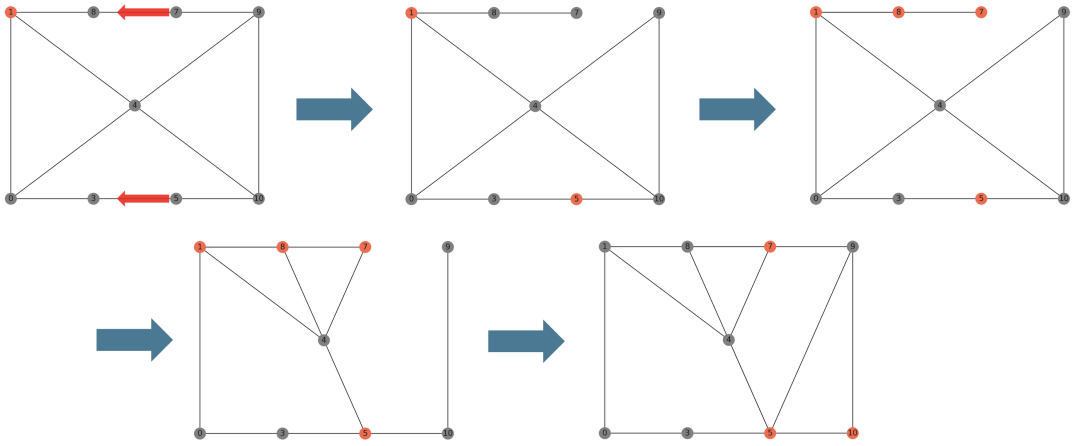

Fig. 1. Snapshots from the execution of a non-dominated program on the seedling. The program appeared in generation 69 and its sequence is `[del8,rew3,inv3,div1,del0,inv1,inv3,div5,del9,rew7,inv5,div1,inv7,inv8,rew4,inv1,inv8,rew8,inv10,rew9]`. The figure illustrates the graph after operation-node pairs `inv3,inv5,inv8,rew4,rew9`. The operation-node pairs chosen show the important structural/state changes for the example design. The state of grey nodes is 0 and the state of red nodes is 1. The first graph is identical to the seedling G^0 that was used throughout the experiment (except for one state) and the red arrows indicate the braking forces, which are at all times applied on these two nodes (nodes 5 and 7). Some changes are reversed, others are skipped ("neutral"). Although states have no physical meaning, they signal where the new connections need to be made. The resulting structure supports the load points while keeping its volume to a minimum and therefore is found in the low-volume, high-strain areas of the leading Pareto Front.

(changes in functions, characteristics *etc.*) available to individuals at a given time step. This model of emergent novelty decomposes what inherently is a design problem into elementary, incremental steps, that access new, previously unknown states via the recombination of known elements. Motivated by this observation, this paper presents a novel method to explore a wide range of suitable structures that lie within a small number of steps away from a given seed structure, in an explainable and design-friendly way. In order to realise this idea in design contexts, a framework needs to meet three general requirements: *(i)* a powerful, general representation that defines the search space and creates a substrate for evolution, *(ii)* an evolutionary search on this substrate able to discover *adjacent* structures, *(iii)* an independent fitness specific to the domain at hand that guides the search.

To this end, the present representation draws inspiration from previous theoretical work on decentralised, self-organised developmental processes on graphs [12], [13]. Structures are represented as graphs, a versatile vehicle for structure exploration that finds applications in numerous engineering problems. At the same time, sequences of graph operations are seen as "recipes" that obtain desirable graphs, creating a useful space that can be searched upon by an Evolutionary Algorithm. The proposed method is tested on a structural design task which determines the fitness function that drives the search. Consequently, our three requirements are satisfied.

II. Methodology

In Graph Automata [12], Cellular Automata are organised in graph topologies and state-dependent graph operations take place to change local structure and states alike. Importantly, it is demonstrated how *operation sequences* can be used to achieve structures with complex properties. Similarly, Generative Network Automata [13] map local states to operations to create rules with complex behaviours that rapidly develop seed graphs, drive them to extinction, or converge to intermediate structures.

The proposed method evolves similar operation sequences (hereafter *"programs"*) that act upon a base graph (seedling) to discover populations of adjacent graphs with desired properties. A formalism analogous to Graph Automata and Generative Network Automata is adopted to define those operations on graphs. The framework in principle allows for *domain-agnostic graph evolution*: individuals are evolved in the program space, structures are manipulated in the graph space and fitness is calculated in the domain space. Representation and application domain do not bias each other and fitness is treated as an independent module. Similarly to the frameworks mentioned, the current implementation represents structures as graphs with nodes that have binary states $\in \{0, 1\}$. Four basic local graph operations are introduced, that can act upon a single node each time:

i. Division (`div`): The node is divided into three new nodes that carry the same state. The resulting nodes are connected to each other in a cyclic subgraph and they also sequentially connect to the divided node's neighbourhood until they all connect to at least one pre-existing node.
ii. Deletion (`del`): The node is deleted along with the edges to its immediate neighbourhood
iii. Inversion (`inv`): The binary state of the node is inverted.
iv. Rewiring (`rew`): The node is connected to all nodes in the graph with state equal to 1, disconnected from all nodes with state equal to 0.

The operations chosen in the present formulation represent rudimentary node-centric (decentralised) changes that add or remove nodes, change their states or the overall connectivity

according to their states. Note that, although this operation set resembles the constructive/destructive graph processes seen in previous literature [12], [13], there is no limitation to the number or kind of operations that can be included, as long as they yield valid structures within the graph space.

It is now evident how a program P of length L (*i.e.* containing L such operations) can be applied on graph $G = (V, E)$ to yield a second, different graph, deterministically. Figure 1 illustrates an example of this process. A program P changes graph G in steps and as a result $G = G^t$ is dynamic during P's execution and so are its node and edge sets, V^t, E^t, $t = 1, \ldots, L$. It follows that P could be specifically constructed to reach a G^L with desired properties or, more generally, a set of programs $\{P_i\}$ with lengths $\{L_i\}$ could be separately applied to an initial graph $G = G^0$ to yield the *adjacent possible* $\{G_i^{L_i}\}$, a set of graphs with said properties. This forms the basis of the present evolutionary methodology, as the evolving individual is a program P_i consisting of operation-node pairs $op_i^t n_i^t$:

$$P_i = |op_i^1 n_i^1 | op_i^2 n_i^2 | \ldots | op_i^L n_i^L | \tag{1}$$

where, generally, $op_i^t \in \{\texttt{div}, \texttt{del}, \texttt{inv}, \texttt{rew}\}$ and $n_i^t \in V_i^t$. As in GP methods and since individuals are represented as programs, there is no distinction between genotype and phenotype. The evolving genome is also the individual, a sequence of instructions that are executed sequentially. Hence, what is being evolved is the *logic* that manipulates the structure of seed graph G^0 and not its *structure* per se. In this evolutionary scheme, it is possible for multiple different programs to lead to the same graph. This way the algorithm is allowed to discover different ways to access a desirable graph structure.

In the beginning of the evolutionary process (generation 0), a population of programs $\{P_i\}$ is initialised; their lengths and operation-node pairs are randomised. The programs are assigned an initial fitness. After that, and for every generation, programs are crossed, mutated, evaluated and selected according to given evolutionary operators and their corresponding probabilities. During its evaluation, a program P_i is applied on seed graph G^0 *(seedling)* to obtain adjacent graph $G_i^{L_i}$ on which fitness is computed by the fitness function. The fitness function operates exclusively within the graph space, so executing a program is part of its evaluation. In Graph Automata and Generative Network Automata, nodes are chosen either randomly or as part of a sequence that deterministically reaches a specific structure and graph operations are triggered by the local states in that node's neighbourhood, with the use of standard or variable state-to-operation mappings (rules). Here, on the contrary, nodes and operations form operation-node pairs that are part of the genome, therefore they are both always determined by evolution.

III. Experiments

As mentioned fitness is *independent*, and therefore *belongs to the application domain* that is examined. That is, no fitness function can be defined before a design problem is properly posed. For this reason, the evolutionary logic presented is demonstrated on a simple Topology Optimisation task. The structural paradigm used for this purpose is the *"chassis"*, a truss skeletal structure that serves as an abstract model of a car chassis. The chassis allows for the definition of tractable loading scenarios, in which the overall structure undergoes environmental stresses a road vehicle could experience. The loading scenario used here is *braking*, during which the chassis experiences sudden parallel backward forces on two points (nodes that mark the position of the two front wheels), similarly to a Front-Wheel Drive (FWD) vehicle during hard braking. The task is to discover a set of adjacent chassis structures $\{G_i^{L_i}\}$ that support the load points, directly or indirectly. The evolutionary scheme has to detect these environmental forces and find a variety of ways to respond. The evolved programs will this way develop structures that *specialise* in braking. The search starts from a seedling G^0 that qualifies as a chassis but does not support the load points substantially.

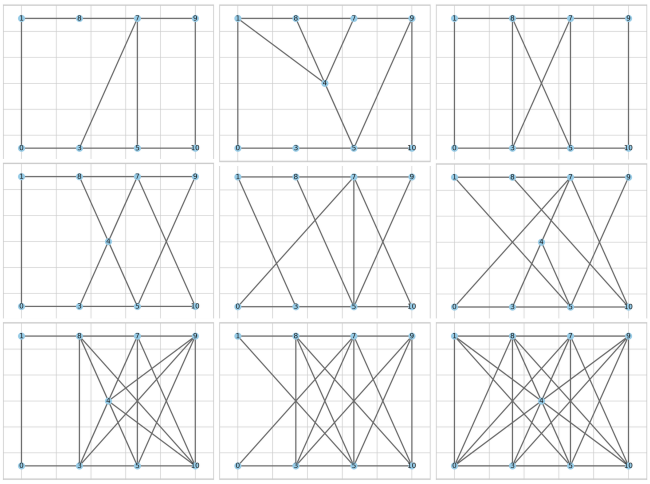

Fig. 2. Light, medium and heavy designs from the Pareto Front. Evolution finds different ways to support the joints on which the braking forces are applied.

A. Fitness

The main metrics evaluated in this simulation relate to the structure's overall strain and deformation. However, defining fitness as less strain and deformation leads to a trivial task, as any working algorithm would just simply continue adding members or joints until the end of its run and yield complex and unusable structures. For this reason, a mass/volume metric has to be taken into consideration as well. Thus the fitness of a given program P_i is calculated on the graph $G_i^{L_i} = (V_i^{L_i}, E_i^{L_i})$ it produces using three non-weighted objectives which the algorithm attempts to minimise: mean member strain energy U_{mean}, mean absolute joint displacement Δ_{mean} and total volume Vol_{total}:

$$U_{mean}(G_i^{L_i}) = \frac{1}{|E_i^{L_i}|} \sum_{e \in E_i^{L_i}} u_e \tag{2}$$

$$Vol_{total}(G_i^{L_i}) = \sum_{e \in E_i^{L_i}} A_e l_e \tag{3}$$

$$\Delta_{mean}(G_i^{L_i}) = \frac{1}{|V_i^{L_i}|} \sum_{v \in V_i^{L_i}} \delta_v \tag{4}$$

where u_e, A_e, l_e are the strain energy, cross-sectional area and length accordingly of cylindrical edge/member e and $\delta_v = \sqrt{\delta_{vx}^2 + \delta_{vy}^2}$ the total length of node v's displacement in axes x, y. u_e and δ_v are obtained for all members and joints accordingly via Finite Element Analysis (FEA) and specifically simulation of the loading scenario using truss elements in CalculiX [14]. In a truss simulation, nodes are seen as joints and edges as members. The joints are pin joints (not welded connections) and transfer tensile or compressive stress to the members. Member crossings are allowed but do not form joints; the crossing members are simulated as independent "springs". The analysis is $2-$dimensional, so all the forces are applied and distributed within the same plane.

The simulation in CalculiX returns results normally even when axial stresses exceed the critical limit of a truss, calculating u_e and δ_v regardless of whether they are dangerous. The search is unconstrained when it comes to the three main objectives, so it is able to start from theoretical, failing structures and move on to increasingly tractable ones. Of course, the loading scenario used in this experiment challenges the seedling without threatening its structural integrity. The seedling's topology G^0 and the forces applied can be seen in the first panel of figure 1. Its length is $2.5m$ and its width $1m$. Members are assumed to have a constant cross-sectional area of $1cm^2$. The braking forces are always applied on the same joints and are both equal to $10000N$. Young's modulus[1] is $70GPa$ for the chosen material (aluminium).

B. Evolutionary Hyperparameters

The core of the presented method is implemented as a custom Multi-Objective Evolutionary algorithm in DEAP [15]. The NSGA-III [16] selection strategy is used for the selection process of the algorithm. The crossover operator is implemented as one-point crossover with probability 0.8. The mutation probability is 0.3. The custom mutation process randomly chooses one of three actions at the mutation locus: *i)* insert a new randomly generated operation-node pair, *ii)* replace given operation-node pair with a randomly generated one, or *iii)* delete given operation-node pair. The population size used is 400 and evolution lasts for 100 generations. Note that no particular tuning was involved in the choice of the above hyperparameters. The initial population is randomly generated: operation-node pairs of programs are randomised, while program lengths vary from 1 to 25 with uniform probability. Dividing or removing supports (loads, fixed points) are seen as illegal actions and operation-node pairs that trigger them are considered as "neutral genes" (deactivated, however still part of the genome). Programs leading to structures that fail to meet a basic set of requirements are penalised and assigned a fitness of $[10^{10}, 10^{10}, 10^{10}]$. Penalised characteristics include empty graphs (no nodes or edges), loads not supported by at least one added new member, articulation points (points that lead to two separate graphs if removed) and structure failures/singularities that force the CalculiX solver to stop early. Overlapping members with same orientation are allowed. In practice, the simulation treats them as independent members with separate physical attributes.

[1]Young's modulus expresses the stiffness of a material. Higher means stiffer, lower means more elastic.

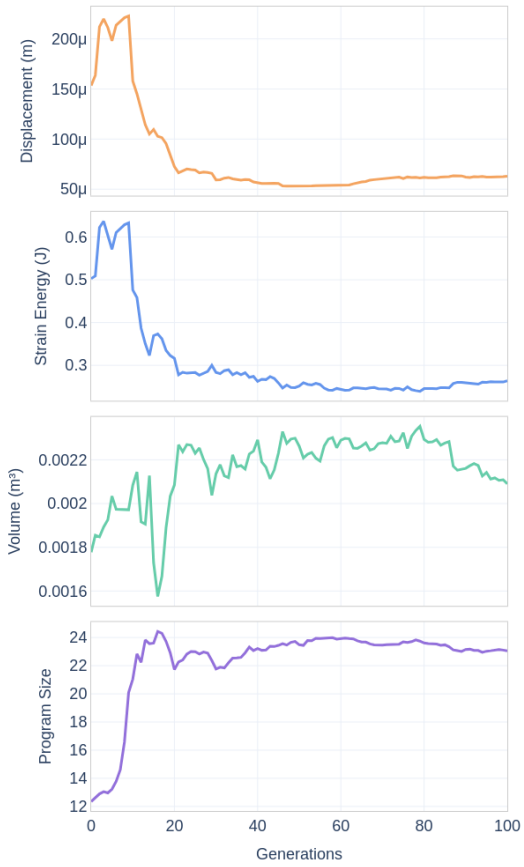

Fig. 3. Population means over generations for the three objectives (displacement in metres, strain energy in Joules, volume in cubic metres) and program size.

IV. Results

Figure 3 shows the evolution of objectives over generations, *i.e.* the population means for mean member strain energy, mean absolute joint displacement and total volume. Population displacement and strain usually follow similar curves and converge early (convergence is visible even after 20 generations). Notice that the overall optimisation of population measures such as strain energy does not seem dramatic, however the strain energy of a single structure is the mean strain energy of its members; so, given that in scenarios like braking, stress is localised and as a result only a minority of the members

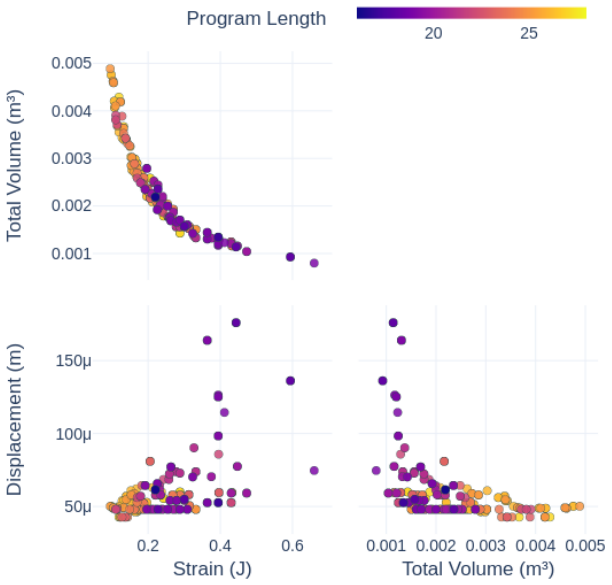

Fig. 4. Pareto-optimal solutions in the volume-strain, displacement-strain and displacement-volume planes. The colour indicates the minimum program length that reaches each structure (warmer is higher, colder is lower).

are significantly stressed, reducing the local stress enough to bring a change of order 10^{-1} in the average strain energy is a meaningful structural improvement for adjacent designs. This holds true for displacement as well, given that most points on the frame are fixed. The current implementation uses standard graph visualisation logic in NetworkX [17] to place newly added nodes as it is not equipped with a general method that places new joints appropriately. As a result, divisions in programs often lead to non-desirable structures that are often penalised heavily, with most non-dominated solutions carrying operation-node pairs with op_i^t =`div` only as neutral genes in the end of the run. This means that, although division is a useful operation in principle, this experimental setup did not favour it in practice. This doesn't stop the search from producing a variety of light, medium and heavier designs (figure 2).

Naturally, heavier designs increase the mean population volume, as the seedling is closer to lighter ones. As a result, the mean population volume increases compared to its starting point. It then reaches a plateau for generations $40 - 80$ and maintains a declining trend for the last 20 generations. Volume functions as a soft constraint: a design is allowed to be heavier only if this improves its strain or deformation considerably. Trivially minimal and rapidly growing structures are equally avoided and growth itself is neither explosive nor bounded, but *regulated*. Finally, the mean program length (mean number of operation-node pairs) starts near 12.6, rapidly increases and exhibits steady behaviour by generation 40 after mild fluctuations, converging near 23 without the use of any parsimony measures, which at first glance indicates the absence of uncontrollable bloat-like phenomena.

Penalised designs and their corresponding programs are omitted from the calculation of statistics during the run as they seriously inflate the calculated values. As a result, their presence in a generation affects the quality of statistics for that generation (small sample). This statistic *"cold start"* effect decays to zero abruptly before generation 10 (the number of penalised designs is 206 in generation 8, 33 in generation 9 and 0 in generation 10), after which point statistics become accurate and representative. The resulting Pareto Front provides a much clearer view of the quality and spread of solutions, revealing all the *trade-offs* between the objectives (Equations 2, 3, 4) which cause the characteristic curves. The trade-offs become more apparent when the objectives are compared to each other, that is when the $N-$dimensional Pareto curve is projected onto N separate basic planes. Figure 4 shows this decomposition for this problem ($N = 3$).

Volume and strain are approximately inversely proportional in designs which shows the reasonable trade-off: heavier designs have high volume but distribute strain efficiently and light designs have lower volume but worse strain management. Volume and displacement are similar, but displacement as a variable creates a rougher fitness landscape in this problem. The braking scenario does not test the displacement of joints as much as the tension/compression of members. In practice, only 2-3 points are being significantly displaced in this experiment, while strain can be distributed to a greater range of members. In addition, the framework is currently only capable of adding or removing members of a constant cross-sectional area A_e and not reinforcing -or weakening- them to support joints appropriately. This is by definition a binary structural change which shapes a sharp and binary volume-displacement trade-off. Finally, displacement and strain seem to be linearly correlated. This result is rather intuitive, as highly stressed structures tend to have both high displacements and strain energies.

Program lengths in the Pareto Front are diverse (figure 4). The maximum length is 28, the minimum 16 and the average 23.05. Despite the variety, some patterns in their distribution can be observed: short programs tend to occupy high-stress, high-displacement, low-volume areas of the objective space, while large programs are on average closer to low-stress, low-displacement, high-stress areas. Considering that the seedling is a minimal, sub-optimal design, arriving to an improved but still light design requires on average fewer instructions than reaching a significantly heavier but stiffer one. The Pareto Front includes more than 10^3 unique programs for this experiment, the number of the resulting designs is however at least one order of magnitude smaller, as different programs can in fact lead to the same structure. So, while a mapping of a single program to a graph is *one-to-one* (*i.e.* deterministic), the mapping of multiple programs to multiple phenotypes can be *many-to-one*. The algorithm discovers a variety of designs, while this redundancy helps it explore all the different ways to reach an optimal design. This can also be interpreted as a form of convergence; the Pareto-optimal designs appear as local structural attractor states that lie within a certain radius in the program space, although further investigation is needed

to understand convergence to adjacent structures.

V. Conclusions, future work

This work is driven by the general idea that by using a small set of basic structural changes for a small number of steps one can depart from a given structural state and discover a large and diverse set of *adjacent* structures. The proposed method applies this idea on graph structures using graph operations borrowed from Graph Automata formalisms and designs an evolutionary search on the space of graph operation programs. This logic is tested on a Topology Optimisation scenario where it starts from a seedling and discovers a range of Pareto-optimal adjacent designs within a program length radius, maintaining the main design intent -as it starts from a chassis and consistently arrives at valid chassis structures. The results shown are obtained with minimal tuning. The evolved programs are ordered sequences of deterministic operations that lead to improved designs and therefore by definition interpretable. Optimisation is embedded into the framework without being the sole purpose of the design process, but is rather used to drive change and reveal trade-offs in the objective space. The modular architecture of the framework disentangles program space, graph space, and domain space, laying the foundation for future advancements in generalisable, graph-based evolutionary design.

Although the results briefly demonstrate the potential of the proposed method, future work will further substantiate the underlying hypothesis of this paper. Specifically, it is crucial to investigate how the loading scenario, the choice of the seedling and the assumptions made by the FEA simulation bias the search. The simulation of more loading scenarios and the execution of multiple runs per loading scenario will ensure the robustness of the algorithm. The simulation itself must allow for more degrees of freedom in member movement. Replacing truss elements with beams will allow FEA to also simulate bending and not just tension or compression. Future work will also examine more sophisticated operation sets. The operation set is decoupled from the framework but does however impact its dynamics heavily, therefore it should be seen as an important hyperparameter. The definition of task-specific design operations will enable future versions of the algorithm to navigate the fitness landscape smoothly and effectively.

Further improvements will also involve less extreme penalisation: penalty values that are closer to non-penalised ones will enable the experimenter to include them in statistics and visualise the objectives' progress without "cold start" effects. In addition, enabling evolution to control the cross-sectional areas of members and therefore reinforce or weaken them on a case-by-case basis will create a much smoother volume-displacement trade-off and an overall more accurate Pareto Front. Finally, a more sophisticated node placement method will place newly added nodes appropriately and allow division-based operations to improve structures in the future. Ground structure methods that use standard meshes and joint locations and force-directed methods that treat nodes as independent bodies and edges as attraction-repulsion relationships are only two examples of approaches that could be used to enhance the such operations.

References

[1] A. Slowik and H. Kwasnicka, "Evolutionary algorithms and their applications to engineering problems," *Neural Computing and Applications*, vol. 32, pp. 12 363–12 379, 2020. [Online]. Available: https://doi.org/10.1007/s00521-020-04832-8

[2] J. Lehman *et al.*, "Evolution and the knightian blindspot of machine learning," *arXiv*, 2025. [Online]. Available: https://arxiv.org/abs/2501.13075v1

[3] S. J. Hickinbotham *et al.*, "Morphogenic shape grammars for the design of engineering structures," *IEEE Symposium Series on Computational Intelligence*, pp. 1–6, 2025. [Online]. Available: https://doi.org/10.1109/CIES64955.2025.11007633

[4] H. Janmohamed, T. Pierrot, and A. Cully, "Improving the data efficiency of multi-objective quality-diversity through gradient assistance and crowding exploration," *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 165–173, 2023. [Online]. Available: https://doi.org/10.1145/3583131.3590470

[5] J. T. Bishop, J. Jooste, and D. Howard, "Evolutionary exploration of triply periodic minimal surfaces via quality diversity," *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1165–1173, 2024. [Online]. Available: https://doi.org/10.1145/3638529.3654039

[6] J. Lehman and K. O. Stanley, "Evolving a diversity of virtual creatures through novelty search and local competition," *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 211–218, 2011. [Online]. Available: https://doi.org/10.1145/2001576.2001606

[7] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," *arXiv*, 2015. [Online]. Available: https://arxiv.org/abs/1504.04909v1

[8] J. F. Miller, "Cartesian genetic programming," *Natural Computing Series*, vol. 43, pp. 17–34, 2011. [Online]. Available: https://doi.org/10.1007/978-3-642-17310-3_2

[9] S. Cussat-Blanc, K. Harrington, and W. Banzhaf, "Artificial gene regulatory networks—a review," *Artificial Life*, vol. 24, pp. 296–328, 2019. [Online]. Available: https://doi.org/10.1162/artl_a_00267

[10] F. Tahernezhad-Javazm *et al.*, "Evodevo: Bioinspired generative design via evolutionary graph-based development," *Algorithms 2025*, vol. 18, 467, 2025. [Online]. Available: https://doi.org/10.3390/a18080467

[11] S. Kauffman, "Investigations," 1996. [Online]. Available: https://www.santafe.edu/research/results/working-papers/investigations

[12] K. Tomita, H. Kurokawa, and S. Murata, "Graph automata: natural expression of self-reproduction," *Physica D: Nonlinear Phenomena*, vol. 171, pp. 197–210, 2002. [Online]. Available: https://doi.org/10.1016/S0167-2789(02)00601-2

[13] H. Sayama, "Generative network automata: A generalized framework for modeling complex dynamical systems with autonomously varying topologies," *IEEE Symposium on Artificial Life*, pp. 214–221, 4 2007. [Online]. Available: https://doi.org/10.1109/ALIFE.2007.367799

[14] G. Dhondt, "Calculix crunchix user's manual version 2.22," 2024.

[15] F. Félix-Antoine *et al.*, "Deap: evolutionary algorithms made easy," *The Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, 2012.

[16] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints," *IEEE Transactions on Evolutionary Computation*, vol. 18, pp. 577–601, 2014. [Online]. Available: https://doi.org/10.1109/TEVC.2013.2281535

[17] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *7th Python in Science Conference*, 2008, pp. 11 – 15.

[18] R. Doursat, H. Sayama, and O. Michel, "A review of morphogenetic engineering," *Natural Computing*, vol. 12, pp. 517–535, 2013. [Online]. Available: https://doi.org/10.1007/s11047-013-9398-1