Proceedings of the IASS 2024 Symposium
Redefining the Art of Structural Design
August 26-30, 2024, Zurich Switzerland
Philippe Block, Giulia Boller, Catherine DeWolf,
Jacqueline Pauli, Walter Kaufmann (eds.)

AI-guided generation of reticular structures by integrating reinforcement learning with shape grammars

Vishnukumar RAJASEKAR[a,*], Juney LEE[b], Aysenur GENCSOY[a], Klaas DE RYCKE[c,d,*], Pierpaolo RUTTICO[a]

* Bollinger+Grohmann S.A.R.L.
15 rue Eugène Varlin, 75010 Paris, France
vrajasekar@bollinger-grohmann.fr

[a] Politecnico Di Milano
[b] Carnegie Mellon University
[c] Ecole nationale supérieure d'architecture Versailles
[d] Bartlett UCL

Abstract

During the early stages of design, there is a need for computational tools that impartially generate and explore diverse structural possibilities. While conventional methods based on the Finite Element Method (FEM) have limited generative capacity, a grammar-based approach—integrating structural information to generate diverse structures—significantly broadens the design space. Navigating this broadened design space involves either sorting and clustering synthetically generated datasets or guiding the system to produce feasible outcomes. This paper introduces a guiding methodology for a grammar-based system, Grammatical Design with Graphic Statics (GDGS), aimed at creating statically equilibrated, goal-oriented 2-dimensional (2D) structures. By incorporating an artificial intelligence (AI)-based technique called reinforcement learning (RL), specifically Q-learning, within this framework, the paper focuses on guiding GDGS to produce diverse yet feasible reticular structures. The synergy between grammar-based rule transformations and RL's action selection forms the foundation for deploying RL-based algorithms with grammatical design methods. The presented experiments demonstrate the potential of incorporating multiple feasibility criteria within the RL system, balancing multiple objectives and exploring alternative latent structural solutions. This approach provides a valuable tool for designers, streamlining the generation of feasible structures. The paper concludes by discussing how RL methods can create synthetic datasets of goal-oriented, high-performance structures, with the sequence of rules acting like encoded DNA, along with performance metrics.

Keywords: Reinforcement Learning, Q-learning, Shape Grammar, Graphic Statics, Generative Structural Design, Structural Geometry

1. Introduction

Generation of discrete truss structures is a problem that has been vastly researched. The advent of computational methods have helped move from the drafting board to digital screens for solving such design problems. In structural analysis, this brought forth the shift from graphical methods like Graphic Statics (GS) to numerical methods like Finite Element Modelling (FEM) primarily due to FEM's broader analytical capabilities, which extend beyond the limitations of GS in handling only axial forces [1] [2]. Additionally, FEM-based methods integrated within parametric environments facilitate rapid analysis of complex geometrical models and their variations [3]. However, these conventional parametric approaches, as illustrated in (Figure 1a), often suffer from a limited design space, resulting in a lack of diversity among potential solutions.

An alternative to this conventional parametric modelling is a grammar-based method for automatic generation of unbiased and diverse design variants [4]. Shape grammar (SG), a grammar-based design methodology, enables the creation of diverse design alternatives by applying geometric transformation rules (Figure 1b). Depending on the formulation of the grammar rules, an open-ended system is equipped to generate diverse solutions across the spectrum of the design space. The strength of such a method is its unbiased explorative capability which, when combined with GS as an analysis method, guarantees equilibrated structures at each generative step. Grammatical Design with Graphic Statics (GDGS) combines SG with GS to generate 2-dimensional (2D) reticular structures that are diverse in form and maintain static equilibrium. However, the broadened design space of GDGS demands a guiding strategy to find the structures that satisfy multiple criteria.

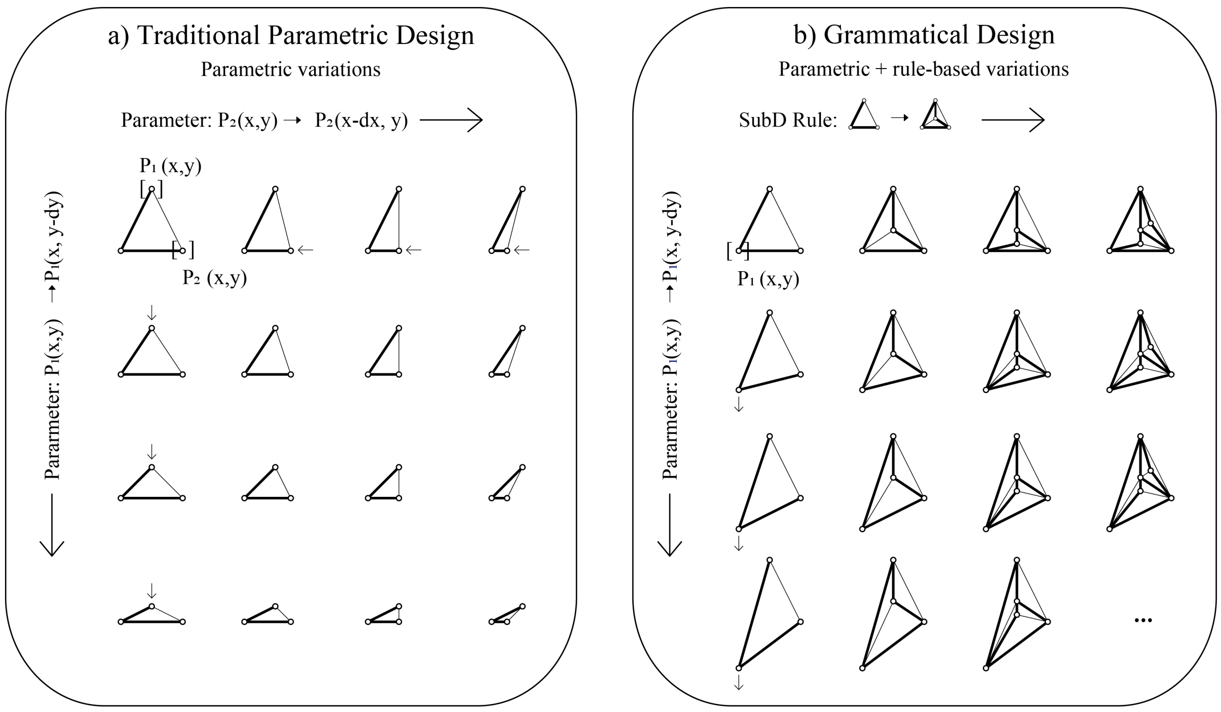
a) Traditional Parametric Design
Parametric variations
Parameter: P₂(x,y) → P₂(x-dx, y)
P₁ (x,y)
P₂ (x,y)
Pararmeter: P₁(x,y) → P₁(x, y-dy)
b) Grammatical Design
Parametric + rule-based variations
SubD Rule:
P₁ (x,y)
Pararmeter: P₁(x,y) → P₁(x, y-dy)
...

Figure 1: Comparison of parametric vs grammatical design spaces: a) Example of a traditional parametric design method which varies two parameters y and x to adjust the position of nodes P1 and P2 respectively, resulting in limited and linear variations; b) Grammatical design method that combines parametric variations with a typical subdivision rule-based grammar to generate more diverse and complex variations.

There are two strategies to find such options in a broadened design space. The first approach involves generating a dataset of all potential solutions, followed by sorting or clustering them based on specific structural criteria. This process is usually time-consuming and would require a heuristic solver [5]. A second approach, which is addressed in this paper, is to use a guiding system based on *reinforcement learning* (RL) that helps the grammar-based system to generate solutions by progressively learning the actions involved in generating goal-oriented structures. RL, a subset of *machine learning*, is an *artificial intelligence* (AI) based technique that uses an RL *agent* to learn the sequence of actions needed to produce a solution, thereby enhancing the overall efficiency and effectiveness of the grammar-based generation process. An RL agent is an autonomous entity that interacts with an environment by taking actions and receiving feedback in the form of rewards or penalties, aiming to maximise the cumulative reward over time. This learning process allows the RL agent to optimise its strategy for generating solutions in a dynamic and adaptive manner, integrating machine learning principles to continuously improve performance based on experience.

2. State of the art

Graphic statics serves as a valuable tool by furnishing visual insights that facilitate an intuitive understanding of a structural system's behaviour. Graphic statics utilises two diagrams that together describe the static equilibrium of a structure along with a set of applied loads and reaction forces. One diagram visually captures the structure's geometry, reaction forces, and applied loads, and the other represents the equilibrium of internal and external forces in the structure. These diagrams are referred to as *form* and *force diagrams*. Starting from Simon Stevin (1586) to Carl Culmann (1864), the theory of graphic statics was extensively developed and used to analyse and design a variety of structures until the adoption of other mathematical methods in the early 20th century [6]. The gradual disappearance of the method is attributed to advancements in computational (numerical) analysis methods and the laborious process of manually drawing form and force diagrams [7]. However, recent developments in powerful Computer Aided Design (CAD) based three dimensional (3D) environments like Rhinoceros has marked the resurrection of graphical methods in the form of computational graphic statics. Some of the notable examples include RhinoVAULT [8], Interactive Graphic Statics (IGS) [9], 3D Graphic Statics (3GS) [10], PolyFrame [11] and GSDesign [12].

In a parallel development, the field of architectural design has witnessed the rise of generative design methodologies. Frazer [13] points out the emergence of generative design in architectural design was during the 1970s. Since its introduction, many research projects utilised different approaches, like cellular automata (CA) and shape grammar (SG), to help designers create new design schemes based on the rules or constraints [14]. Shape grammar is a computational formalism through which shapes are derived from a language (design space) of shapes by the successive selection and application of specific sequences of transformation rules [15]. It is important to note that the grammar rules themselves define the scope of the shape language i.e. all possible shapes that can be generated from those specific sets of rules. Each shape is thus obtained through a distinctive path of state-action from a decision tree of the available grammar.

Shape grammars have played a significant role in structural engineering by providing a formal method for generating complex shapes and structures, evolving into "functional" grammars as introduced by William Mitchell, which integrate performance criteria directly into the rule sets for a holistic approach to design. Cagan et al. [16] introduced *shape annealing*, an optimization technique which combined the principles of shape grammars with *simulated annealing* (SA), to produce optimally directed structures. SA is a probabilistic optimization technique inspired by the annealing process in metallurgy, used to find an approximate global optimum in a large search space [17]. Until then, shape grammar lacked the decisive capacity needed to produce appropriate problem-specific shapes. Hence, shapes were produced through an exhaustive enumeration of possible derivations of rules or randomly sampling them [16]. In structural design, automatic generation of discrete structures using shape annealing was first addressed by Shea [18]. Drawing upon linguistic theory, structural essays were developed as blueprints for structures by relating rules syntax to structural form and rule semantics to their objective functions. The geometric relationships in the rules help generate forms of preferred visual styles while the objectives are used as an implicit guiding methodology to vary the metrics like structural efficiency, economy and utility of the generated structure [19]. As a result explicit domain knowledge is required to formulate the rules and syntax of the grammar for specific typologies of structures. Shape annealing generates a structure using SG, analyses it using FEM and optimises it using SA to minimise cost functions based on structural behaviour, geometric constraint and economic considerations to reach unknown yet optimal structures.

Lee et al. [4] combined shape grammars and graphic statics into a design methodology called Grammatical Design with Graphic Statics (GDGS), which automatically generates diverse yet statically equilibrated discrete structures. GDGS integrates the geometric generative power of shape grammar with the analytical potential of graphic statics into a simultaneous design process. The structures are generated in a bottom-up manner, node by node, while incrementally resolving all force vectors acting at each specific node. Incorporating graphic statics within the generative process not only helps achieve nodal equilibrium at each step but also supplants the need for FEM analysis, implicitly carrying the internal forces information necessary for optimizations. Despite generating diverse and novel structures, due to the random parameters that define the bounds of the rules, the structures tend to fall outside the boundary domain (Figure 2a). Additionally, the author points out, minimising overlapping members (Figure 2b), incorporating buckling and self weight constraints as few issues to be resolved in future research.

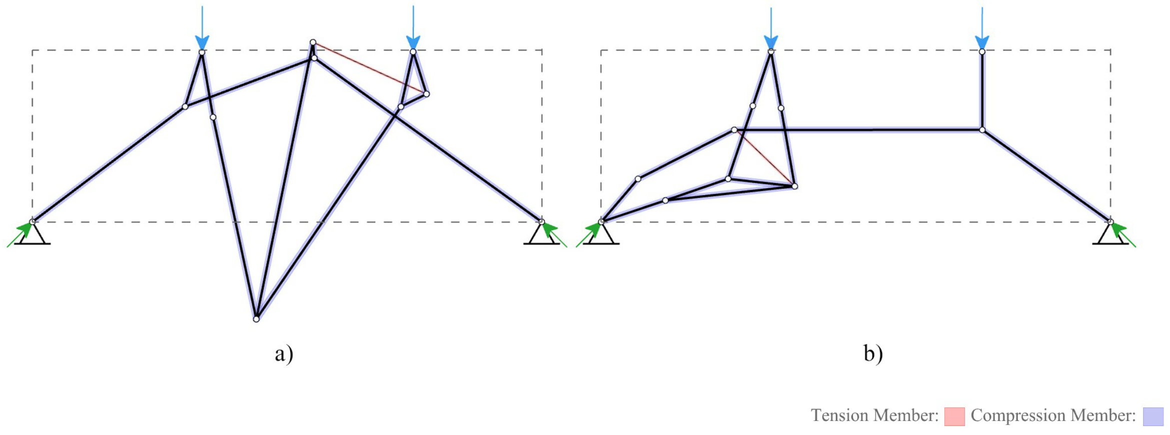

Figure 2: Potential areas of development in GDGS: a) Node falling outside the boundary domain; b) Overlapping members in the generated structure.

More recent developments in the domain of grammatical design aimed at generating discrete equilibrium structures include Policy-based Exploration of Equilibrium Representations (PEER) framework, a design framework aimed at generating variant bar network topologies in static equilibrium [20]. This framework is developed into a grasshopper plugin called Libra [21], which facilitates the sequential exploration of network structures through a set of rules that make up a policy. While the tool lacks an open-source API, automation of the generation process can be achieved using an evolutionary solver to search for the best policies. The method also includes features like backtracking actions and result visualisation with performance metrics, which allow quantitative comparison. Tam et al. combined *Rules-based Graphic Statics* (RGS) with *Geometric Deep Learning* and RL to generate trans-topological performance-oriented network patterns for Reticulated Equilibrium Shell Structures (RESS) [22]. The paper demonstrates the successful integration of traditional analysis methods like GS with modern computational techniques to generate 2.5D structures under multiple load-cases.

This paper's original contribution lies in the application of an AI framework that harnesses RL, serving as a guiding methodology to achieve specific design objectives. The earliest application of combining SG and RL was observed in the domain of spatial planning. This integration has been pivotal in advancing automated design processes, particularly in generating optimised floor plans. Ruiz-Montiel et al. pioneered the use of *Q-learning*, a specific type of *temporal-difference* (TD) learning algorithm with minimal convergence requirements [23]. TD learning is a type of RL method that updates the value of a state based on the difference between predicted and actual rewards over time [24]. It combines ideas from stochastic methods like *dynamic programming* and *Monte Carlo methods*, using observed rewards to adjust the predictions for future rewards incrementally. More recently, *Graph Convolutional Networks* (GCNs) [25] and *Policy Gradient based Artificial Neural Network* [26] have been employed for sizing optimization of free-form lattice shells and form-finding of doubly-curved shell structures using an RL agent.

3. Methodology

3.1 Conceptual Overview

This paper proposes a novel method that combines GDGS with a reinforcement learning technique called Q-learning to guide the generative process towards producing goal-oriented structures. As all the fundamental concepts of GDGS are described in a detailed manner in Lee [4], a detailed overview of GDGS is beyond the scope of this paper.

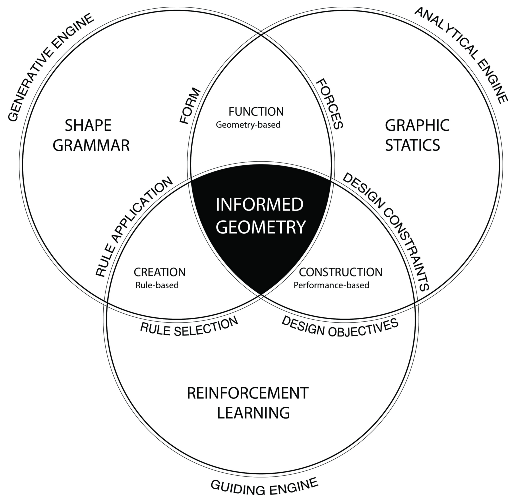

Figure 3: Informed geometry at the intersection of grammatical design, graphic statics and reinforcement learning along with their overlapping attributes

The conceptual foundation of this paper is depicted in Figure 3, which presents a framework for achieving "Informed Geometry" through the integration of shape grammar, graphic statics, and reinforcement learning. SG focuses on rule-based form generation, GS on analyzing forces and constraints, and RL on guiding the design process towards specific objectives. Their intersection ensures the creation of goal-oriented structures that are both diverse yet feasible according to user-defined objectives. This combined approach balances creative, analytical, and guiding methodologies to produce well-informed designs.

3.2 Technical Overview

Given that GDGS operates by sequentially applying grammatical rules and subsequently updating the assembly's state, it can be likened to a RL framework, which similarly relies on sequential decision-making processes. In RL, an agent learns how to choose an action from its action space, within a particular environment, to maximise rewards over time. The bottom-up approach of evaluating each action taken by an RL agent enables it to select specific grammar rules and apply them iteratively to learn the best action to be taken given a particular state. The guiding engine proposed in this paper further encapsulates the existing GDGS class objects into an agent and environment class based on the RL framework. The RL agents decide the placement of nodes and set the parameters at each time step in the problem. Their actions are defined by the selection of one of the applicable rules at each timestep to ensure that the force acting on the node is equilibrated thereby generating a part of the network.

3.2.1 Q-learning, an AI-based technique

The guiding engine utilises a reinforcement learning technique called *Q-learning* to calculate the rewards that drive the choice of rule selection. Q-learning is a model-free reinforcement learning algorithm used to find the optimal action-selection policy for a given finite Markov decision process (MDP) [24]. A Markov Decision Process (MDP) is a mathematical framework where decisions are made sequentially in a system of states, with each decision influencing the transition to the next state based on actions taken and associated probabilities [27]. The basic idea is to learn a policy, which tells an agent what action to take under what circumstances, to maximise the cumulative reward. The RL agent in the method uses a Q-table with dimensions |S| x |A|, where |S| is the number of states and |A| is the number of actions, to store the Q-values for each state-action pair. Q-value is the expected cumulative future rewards that an agent can obtain by taking a particular action in a specific state while following a policy. In each time step, the agent must decide whether to explore new actions or exploit the current knowledge. This is often done using an epsilon-greedy strategy [28], where with probability ε, a random action is chosen, and with probability 1-ε, the action with the highest Q-value is chosen. Once an action is chosen, the Q-value of the current state is updated using the Bellman equation which is fundamental to Q-learning:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q\ (S_{t+1}, a) - Q(S_t, A_t)] \tag{1}$$

In equation 1, $Q(S_t,A_t)$ is the current Q-value for the state-action pair. R_{t+1} is the observed reward. S_{t+1} is the next state. α is the *learning rate* which controls the impact of the new actions and their rewards. γ is the *discount factor* which accounts for future rewards.

3.2.2 Framework

The environment (Figure 4a) simulates the structural assembly generation process, using generative methods of the GDGS framework. Its functionalities include restoring the assembly to its initial state, monitoring the state to obtain the list of applicable grammar rules (Figure 4b) and checking if the generation is complete. Each step involves applying an action and returning the next state and reward, computed based on factors like node positions relative to the boundary, member intersections, and assembly length differences.

The Q-learning agent learns optimal strategies for assembly generation, relying on the environment for state information and the computational methods of GDGS for actions in the form of grammatical rules. Initialised with hyperparameters like learning rate and discount factor, it balances exploration (trying new actions) and exploitation (using known actions to maximise rewards). As the training progresses, the rate at which the agent explores is reduced, thereby increasing the chances of obtaining maximum rewards. The Q-values are computed based on the immediate reward and the estimated future rewards, thereby refining the agent's understanding of the optimal policy over time. During the training, for each *episode* a structural assembly is generated. An episode is a complete sequence of interactions between the agent and the environment, starting from an initial state and ending when a terminal state is reached. At each time step within the episode, an action is selected (Figure 4c) and applied to the assembly for which the rewards are calculated, and Q-value are updated in the Q-table (Figure 4d). An episode is thus comparable to one loop in the training process where a single structure is generated. Saving and loading the Q-table preserve learned knowledge for future use which are to be implemented in the future extensions of this work. Collectively, these attributes and methods define the Q-learning agent's adaptive and learning-driven behaviour, allowing it to navigate the complexities of structural assembly generation through iterative interactions with its environment.

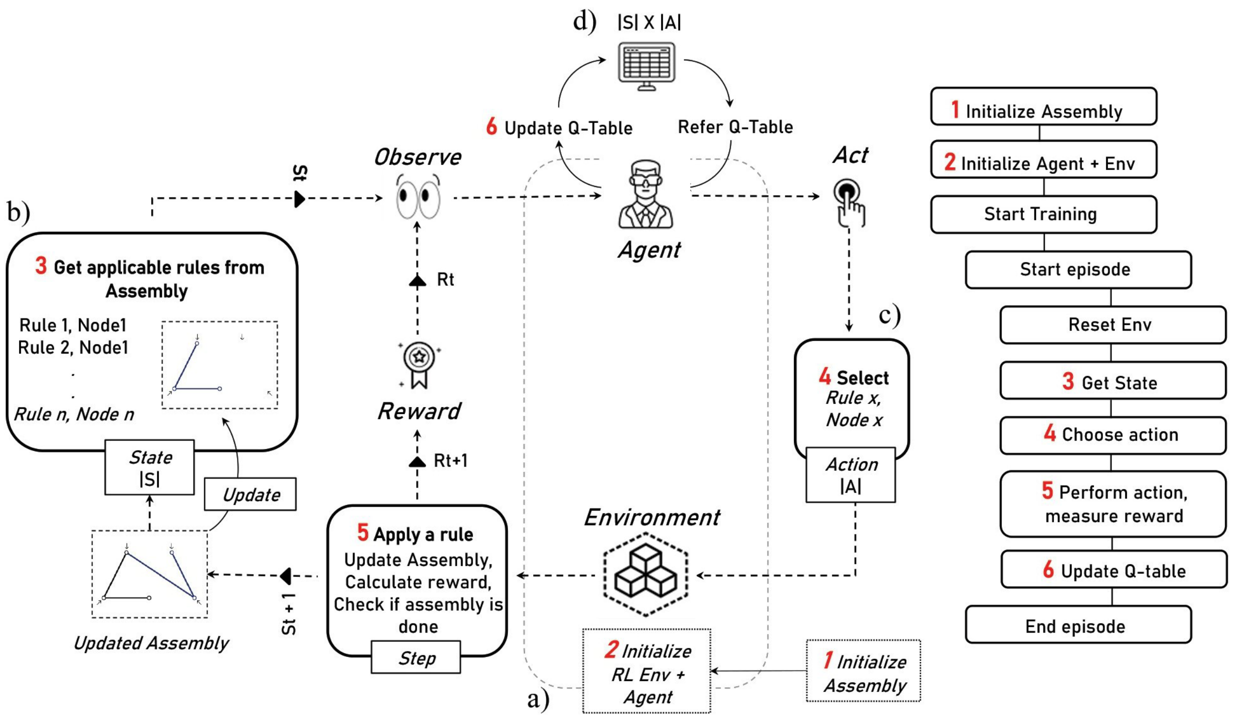

Figure 4: Schematic framework of the Q-learning algorithm: a) The environment encapsulates the GDGS assembly, acting as a simulation environment; b) The state is a collection of applicable grammar rules; c) An action is the selection of a grammar rule to apply to the structure; d) The agent uses the Q-table to choose the action with the highest reward and updates the Q-table with the current state-action pair's Q-value.

3.2.3 Dynamic State

The *state* is an abstract concept that represents a description of the current situation of the environment. States are often defined by a set of discrete features that capture the key aspects necessary for the agent's decision-making, rendering them crucial to the learning process. In this research, the states are represented as all applicable node rules along with their node indices. In GDGS, a rule originally consists of the rule object, its parameters and the node object to which the rule is applied. The parameters are generated randomly for each rule making them variable at a different time step even if the same rule is generated for the same node. As the node instances are custom objects, their value in the memory changes with each iteration. Storing the nodes themselves as a state feature would make them incomparable as python dictionary keys. As the states also act as keys in the q-table for storing the q-value, they need to be comparable in another instance of computation. Hence, just the rule name and the node index are converted into a tuple to be used as a state representation. The states are dynamic, as the number of rules that are applicable change with each application of a rule. The decision of representing all available actions as a state is due to the continuous nature of the node coordinates which are difficult to be compared when stored as keys in the q-table. Storing the rules themselves along with their node index discretises the state space, making them comparable while storing q-values.

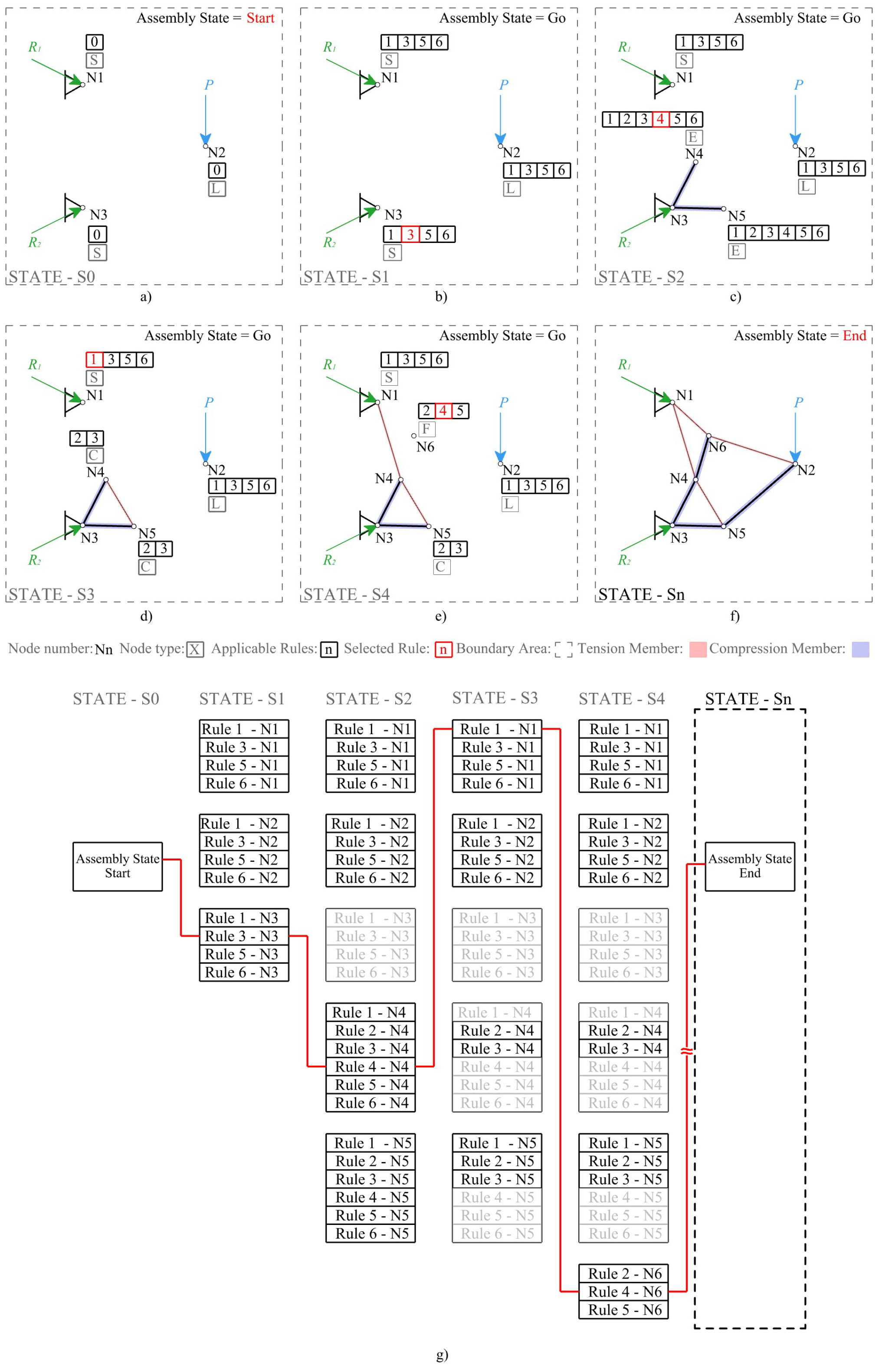

Figure 5: Sequential generative process occurring within a single episode of the RL training process: a-f) State of the structural assembly from initial state S0 to terminal state Sn, marking the structure’s completion; g) State-action pairs corresponding to the assembly, with the state represented as a collection of applicable rules and node indices.

The generation of the structural assembly starts at state S0 when the assembly's state is set from *start* to *go* (Figure 5a). Here, states S1, S2, ..., Sn correspond to the environment's state at each time step in the generation process. At state S1 (Figure 5b), there are 4 applicable rules for each of the three nodes, making a total of 12 rules. The agent chooses rule 3 to be applied at node N3. Rule 3 equilibrates node N3 by splitting it into two new nodes, making node N3 inactive in the next state. In the next state, S2 (Figure 5c), there are four active nodes with 4 applicable rules to nodes N1-N2 and 6 applicable rules to nodes N4-N5. This change in the number of applicable rules to nodes N4 and N5 is due to their node type. Notice that the state size has increased by 8 values, making a total of 20 rules, as the rules associated with node N3 are removed and those for nodes N4 and N5 are added. At state S2, rule 4 is chosen and applied to node N4. Rule 4 connects nodes N4 and N5, changing their type from end to corner. This leads to a condition where only two rules can be applied to nodes N4 and N5 in the next state, S3 (Figure 5c). State S3 has a total of 12 rules: 5 rules for nodes N1-N2 and 2 rules for nodes N4-N5. Similar to previous steps, the agent chooses an action, and the environment updates states accordingly, making a new set of actions available to the agent in the successive state. The process is repeated until the episode ends, which is marked by the equilibration of all forces acting on all the nodes, resulting in the completion of the structure.

3.2.4 Reward

The rewards are calculated and assigned to the agent for each of its actions. Through these rewards, the agent understands the value of each action it is taking at each time step. Rewards are based on the specific goals of the training process. Since this research aims to guide the system towards realistic structures, rewards are based on adhering to boundary conditions, avoiding overlaps and reaching a user specified length for the final assembly. In this thesis, the reward is calculated in the following manner:

- Node Position Reward: For each node in the assembly, if the node is outside the boundary, a reward of -1 is added. If the node is inside the boundary, a reward of +1 is added. This encourages the algorithm to keep nodes within the boundary. Additionally, the episodic loop is broken if a node is placed outside the boundary. This is done to reduce the computation time, by avoiding exploration in areas where the solutions are not feasible to the specific problem.
- Intersection Penalty: The algorithm checks for intersections between members of the assembly. For each intersection found that is not at the endpoints of the members, the agent is penalised with a reward of -1. This discourages the algorithm from creating assemblies where members intersect.
- Length Reward: The algorithm calculates the difference between the target length input by the user and the actual length of the assembly (length). If the length difference is 0 (i.e., the assembly length is exactly the target length), the reward is 1. If the length difference is greater than 0 (i.e., the assembly length is not the target length), the reward is negative *length difference*. This encourages the algorithm to create assemblies that match the target length. The length reward is normalised to be between -1 and 1. It ensures that the reward for the length of the assembly is on a similar scale to the other rewards and penalties.
- Total Load Path: The material efficiency of the design solutions can be incorporated in the form of the overall strain energy (total load path) of the structural assembly. This gives direct control over the material volume that the final solution should achieve. This is incorporated in the form of a design goal to be achieved by the RL agent. In the following equation, the total volume or load path can be calculated as follows in terms of the locations of the nodes in the force diagram, where V represents the total load path or volume of the structure, σ is a constant that represents the allowable stress, P_i is the internal force and L_i is the length of the i^{th} member, respectively.

$$\min_x V = \max_x \frac{1}{\sigma} \sum P_i L_i$$

The node position reward, intersection penalty, normalised length reward and total load path are added together to get the final reward and assigned to the agent. The rewards are then used to update the Q-value of the current state-action pair in the Q-table. The Q-value is the expected future rewards, and is used by the agent to select actions. The new estimate is based on the reward for the current action and the maximum Q-value for the next state. This act of updating the value is a key part of the Q-learning algorithm. To track the convergence of the training process, the average reward obtained by the agent over multiple episodes during training is used. If the average reward stabilises or reaches a plateau, it may indicate convergence. Plotting the average reward over time can help visualise the trend.

4. Results

This section discusses a simple three-point cantilever problem set up with two supports and one load. The benchmark is selected as a Michel truss which was designed by (Michell 1904). Michell's Truss Theory, developed in the early 20th century, focuses on finding the most material-efficient truss configurations. This theory is based on the concept that an optimal truss should distribute the material (members) in such a way that it minimises the total potential energy of the structure subject to certain constraints.

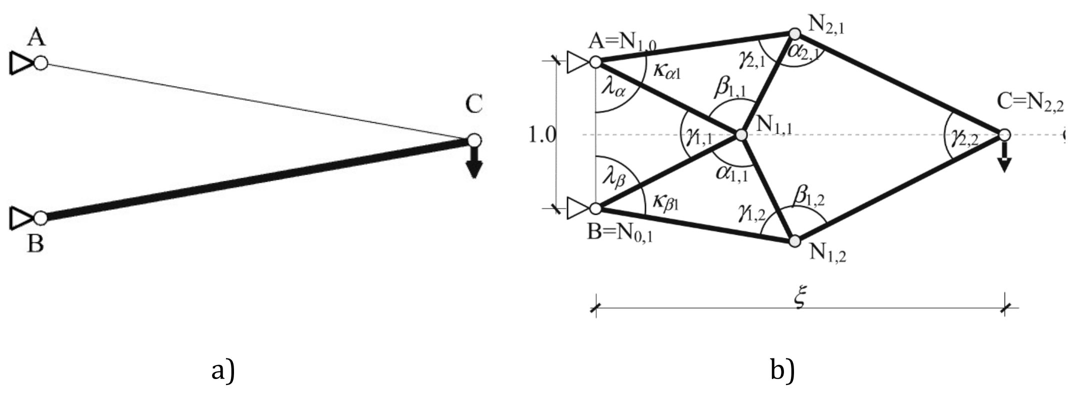

Figure 6: Three-point cantilever problem: a) Simplified cantilever solution for the three-point problem; b) Optimised geometry for the three-point problem [29]

The figures that follow illustrate the results obtained from the training process. It is important to note that the training process was significantly short due to computational time constraints. This can be attributed to the state representation in the RL algorithm. States are generally vectorised in the machine learning process, and future research could address this to improve efficiency. Furthermore, the integration of Q-learning with the GDGS system needs better error-handling capabilities. For episodes above 500, the computational time required is significant. The results showcase that the system initially explores options leading to overlapping members, which is not the case in the later results. With a lower number of episodes (500), the training process indicates that the agent is indeed selecting actions based on the rewards. Training the algorithm for an extended period can result in more diverse solutions. The results shown in Figure 7 indicate that the agent's trend of increasing the rewards is successful and converges. Additionally, it can be seen that the total load path is minimised as well. In Figure 8, the first three images indicate numerous overlapping members resulting in a low reward. Following this, the latter results show that the agent adheres to multiple criteria given. One observation is that as the agent tries to converge to user-defined goals, the diversity in the solutions might reduce. This happens because the agent learns to consistently select the actions that maximise rewards according to the predefined goals, leading to less variation in the solutions. Essentially, the agent becomes more focused on exploiting the known optimal strategies rather than exploring new possibilities, which can reduce diversity.

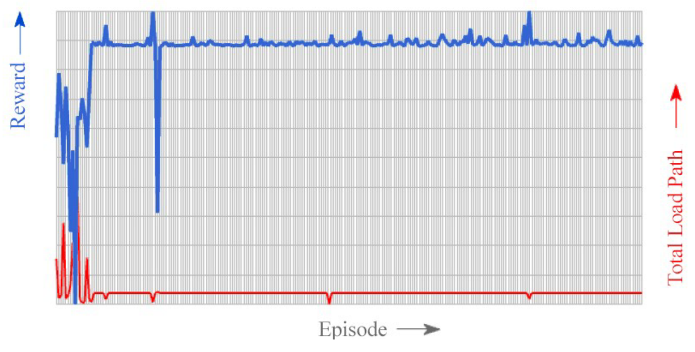

Figure 7: Plot showing the reward gained over training process along with target length

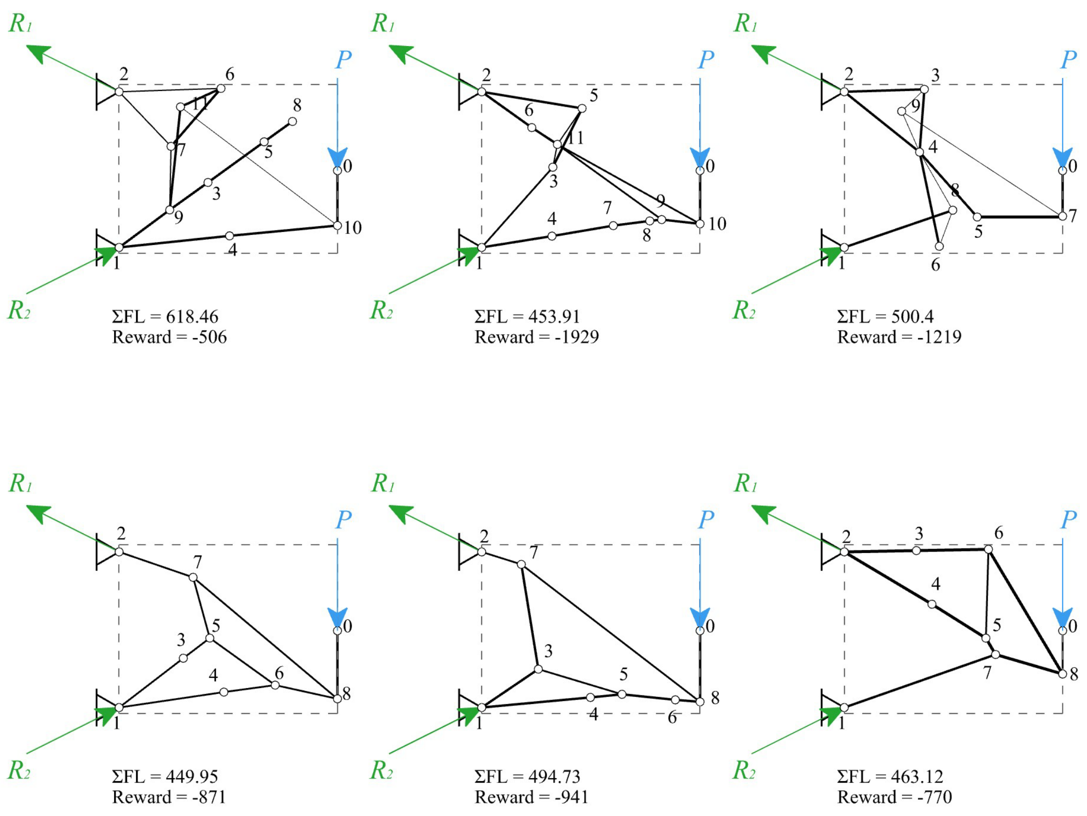

Figure 8: Results from the training process indicating the adherence to boundary condition and reduction in total load path

5. Conclusion

Incorporating a machine learning (ML) framework into design workflows for automatic generation of discrete structures is a compelling and innovative approach. RL can act as a helpful guide in design scenarios, especially when problems involve sequential decision-making. In this study, a Q-learning algorithm was integrated with a generative system to guide the system towards feasibility. From the significantly short amount of training time, the algorithm produced results that ensure that the structure is generated within the boundary domain while achieving the target length. The state representation that is incorporated in this paper is a new method which needs further investigation in different scenarios to evaluate its robustness. A robust state representation would be resilient to changes and capable of adapting to diverse challenges without significant degradation in training output. Q-learning which does not utilise a neural network is a simplified learning technique. Given the relative success of this method, a possible extension could be to integrate neural networks such as Deep A Networks (DQN) or Proximal Policy Optimization (PPO) which are more stable and can handle the dynamic state space and action spaces. To evaluate the capabilities of the GDGS system without an ancillary guiding system, a preliminary test was conducted to select only those designs that satisfy the boundary conditions. The result indicates that GDGS can be used with unsupervised machine learning frameworks that could sort and cluster the solutions. The agent should also be able to select the rule parameters in future iterations along with better reward formulation. RL can generate a collection of high-performing structures that are both diverse and near the global optimum, forming a synthetic dataset. This dataset, along with the grammar rules representing the structural "DNA," can be used to train a more robust model. By encoding geometrical data and shape grammar rules into text embeddings, Large Language Models (LLMs) can be leveraged to analyze patterns, optimise rule sets, and suggest novel structural designs for new boundary conditions. Since RL needs to be retrained whenever the boundary conditions change drastically, using LLMs can quickly adapt to new conditions by leveraging their pre-trained knowledge and providing insights based on the encoded grammar rules and geometrical data.

Acknowledgements

This template was adapted from the one used for IASS-IACM 2008 with modifications modelled after templates for recent IASS annual symposia.

References

[1] Y. Gao, Y. Shao and M. Akbarzadeh, "Application of Graphic Statics and Strut-and-Tie Models Optimization Algorithm in Innovative Timber Structure Design," *Buildings,* 2023.

[2] W. Zalewski and E. Allen, Form and Forces: Designing Efficient, Expressive Structures, Wiley, 2009.

[3] C. T. Mueller, "Computational Exploration of the Structural Design Space," 2014.

[4] J. Lee, C. Mueller and C. Fivet, "Automatic generation of diverse equilibrium structures through shape grammars and graphic statics," *International Journal of Space Structures,* 2016.

[5] K. S. Ochoa, P. O. Ohlbrock, P. D'Acunto and V. Moosavi, "Beyond typologies, beyond optimization: Exploring novel structural forms at the interface of human and machine intelligence," *International Journal of Architectural Computing* , 2020.

[6] W. F. Baker, L. L. Beghini, A. Mazurek , J. Carrion and A. Beghini, "Maxwell's reciprocal diagrams and discrete Michell frames," *Springerlink,* 2013.

[7] J. Lee, L. Enrique, T. Van Mele and P. Block, "Geometry-based Teaching of Structures Through Computational Graphic Statics," *Proceedings of the IASS Annual Symposium 2020/21,* 2021.

[8] M. Rippmann, L. Lachauer and P. Block, " Interactive Vault Design," *International Journal of Space Structures*, 2012.

[9] A. R. Maia , J. Lee, T. Van Mele and P. Block, "An interactive implementation of algebraic graphic statics for geometry-based teaching and design of structures.," *International Fib Symposium - Conceptual Design of Structures,* p. 447–454, 2021.

[10] J. Lee, "Computational Design Framework for 3D Graphic Statics," 2018.

[11] A. Nejur and M. Akbarzadeh, "PolyFrame, Efficient Computation for 3D Graphic Statics," *Computer-Aided Design,* vol. 134, 2021.

[12] F. Bruinsma, "StructuralComponents 8," 2021.

[13] J. Frazer, "Creative design and the generative evolutionary paradigm," 2002.

[14] H. Zhen, W. Yan and G. Liu, "A performance-based urban block generative design using deep reinforcement learning and computer vision," 2020.

[15] S. G and J. Gips, "Shape Grammars and the Generative Specification of Painting and Sculpture," *Proceedings of IFIP Congress 71,* 1971.

[16] J. Cagan and W. J. Mtchell, "Optimally directed shape generation by shape annealing," *Environment and Planning,* vol. 20, pp. 5-12, 1993.

[17] K. S, G. J. C D and V. M P, "Optimization by Simulated Annealing," *Science,* vol. 220, pp. 671-680, 1983.

[18] K. Shea, "Essays of Discrete Structures: Purposeful Design of Grammatical Structures by Directed Stochastic Search," 1997.

[19] K. Shea and J. Cagan, Languages and semantics of grammatical, 1999.

[20] I. Mirtsopoulos and C. Fivet, "Structural topology exploration through policy-based generation of equilibrium representations," *Computer-Aided Design,* vol. 160, 2023.

[21] I. Mirtsopoulos, "https://www.food4rhino.com/en/app/libra".

[22] K.-M. M. Tam, D. Kudenko, M. Khosla, T. Van Mele and P. Block, "Performance-informed pattern modification of reticulated equilibrium shell structures using rules-based Graphic Statics, CW Networks and Reinforcement Learning," *Proceedings of the IASS 2022 Symposium,* 2022.

[23] M. Ruiz-Montiel and J. Boned, "Design with shape grammars and reinforcement learning," 2012.

[24] M. Hu, "Temporal Difference Learning," *The Art of Reinforcement Learning,* 2023.

[25] C.-t. Kupwiwat, K. Hayashi and M. Ohsaki, "Sizing optimization of free-form lattice shells using deep deterministic policy gradient and graph convolutional networks," *Proceedings of the IASS Annual Symposium 2023 ,* 2023.

[26] G. Mirra and A. Pugnale, "Enhancing interactivity in structural optimisation through reinforcement learning: an application on shell structures," *Proceedings of the IASS Annual Symposium 2023* , 2023.

[27] M. L. Puterman, "Stochastic Models," *Stochastic Models,* pp. 331-434, 1990.

[28] R. S. Sutton and A. G. Barto, "Reinforcement Learning - An Introduction," 2018.

[29] A. Mazurek and W. F. Baker, "Geometrical aspects of optimum truss like structures," *Structural and Multidisciplinary Optimization,* 2011.