Investigation of a string-based topology finding framework for structured surface patterns using computer algorithms

Tohma Kobayashi

Master Thesis in Structural Engineering
Faculty of Civil Engineering and Geosciences
TU Delft

Matriculation number	5850916
Supervisor	Dr. Robin Oval
Advisor	Dr. Charalampos Andriotis and Dr. Trayana Tankova

6th December 2024

Tohma Kobayashi:
Investigation of a string-based topology finding framework for structured surface patterns using computer algorithms
Master Thesis, Technical University of Delft, 2024.

Declaration of Independence

I hereby certify, Tohma Kobayashi,

1. that I complete this work independently and without unauthorized help and sources other than those specified.
2. I also declare that I have provided the university with a simple usage agreement right for the purpose of checking using plagiarism software.

Delft, 6th December 2024

Tohma Kobayashi

Abstract

This research addresses the implementation of learning algorithms and generative design in string-based topology exploration methods. It aims to generate diverse structural patterns for shells and surface structures that align architectural, engineering, and construction objectives. By integrating reinforcement learning (RL) and quad-mesh grammars, surface topology is explored through quantitative metrics, demonstrating the strength and generality of this approach. The research ultimately promotes creative exploration during the conceptual stages of structural design, emphasizing collaboration between form-designers and form-analyzers to harness emerging computational techniques.

The quad-mesh grammar was first formulated within a Markovian decision framework to integrate with open-source RL Python packages. States, actions, and rewards were defined with sufficient generality to avoid over fitting while evaluating the RL agent's ability to navigate between two specified string-action sequences and their associated mesh layouts. Initially, simple tasks involving four design steps were tested, followed by more generalized target terminal states with longer design sequences. The impact of different reward structures and varied model parameter setups on convergence and accumulated rewards was also analyzed.

The findings indicate that reward functions based solely on topological and grammatical characteristics did not fully guide the agent from an initial coarse mesh to a target state. However, extended design episodes demonstrated potential for improved RL outcomes. The DQN struggled with non-optimal policies due to negative rewards and sparse positive reinforcement, suggesting that customized model architectures or alternative RL algorithms could enhance performance. The exploration phases yielded suboptimal but diverse mesh configurations, highlighting the need for additional structural and geometric parameters, as well as more complex grammar operations to improve diversity while mitigating computational challenges. These insights underscore the importance of balancing feasibility, exploration, and optimization in computational design workflows.

Acknowledgements

Cameron, thank you for teaching me how to dream with purpose.

Arja, thank you grounding me in reason when clarity was hard to find.

Valentin, thank you for the endless supply of coffee and inspiration.

Mom, Dad, and Mei, thank you for always supporting me and the quiet strength of your love.

Robin, thank you for your patience and the care you've given me through every step.

To all of you, and many others, my gratitude runs deep –your presence has shaped me in ways I'll carry forever.

Contents

1 Introduction

Global challenges present significant constraints on structural design, requiring solutions that ensure safety, durability, and sustainability while balancing material efficiency with ecological and aesthetic considerations. Computational methods have revolutionized this process, allowing engineers to navigate competing objectives and achieve designs that are both elegant and practical. A pivotal aspect of these advancements is the strategic use of patterns, which establish structural relationships within complex geometries and efficiently organize forces and materials to meet the ever-increasing demands of modern building practices. This research explores how the convergence of patterns and computational intelligence has driven a paradigm shift in structural design, enabling the creative and efficient exploration of solutions that address the pressing needs of a changing world.

1.1 PATTERNS

Patterns are omnipresent, shaping both organic and artificial systems. Whether in crystal lattices, genetic compositions, or architectural designs, patterns organize basic units into structured, repeated arrangements that provide clarity and order as seen in Fig. 1.1. Their wide use and identification across innumerable branches of knowledge, at varying hierarchies on micro and macro scales, allude to their fundamental role in shaping our collective experience. These patterns not only underpin our cognitive frameworks for understanding the natural world but also play a critical role in organizing engineering and design principles, particularly within structural systems.

In structural design, patterns extend beyond mere visual repetition; they define the relationships between vertices, edges, and faces, altogether composing the geometrical and topological elements of structures, embodying material systems, force equilibrium, and surface maps for certain typologies [36]. These arrangements directly impact crucial performance factors such as mechanics, sustainability, and cost efficiency, as well as fabrication and assembly methods. In transforming patterns into physical structures, designers engage in a balance

[36] Oval. 2019. *Topology Finding of Patterns for Structural Design.*

between internal forces, a fundamental principle in form-finding techniques and structural optimization.

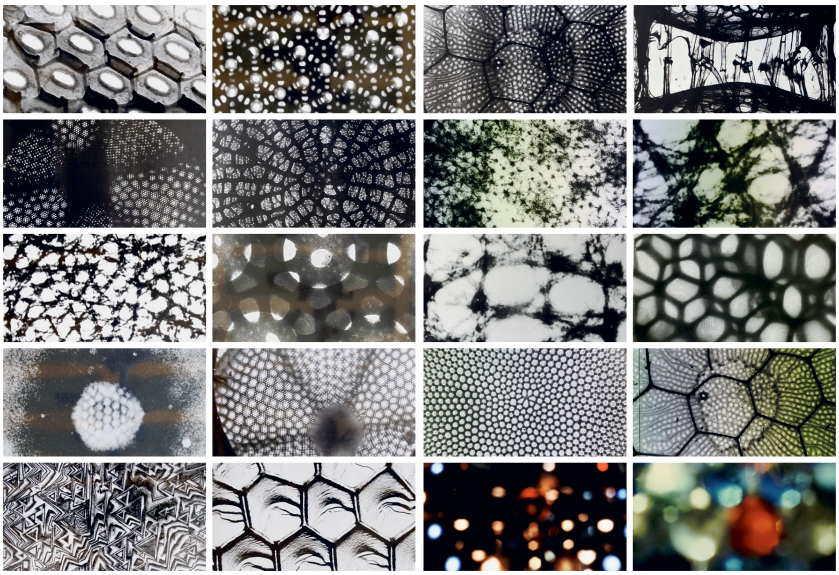


Figure 1.1: Organic and Inorganic Patterns, adapted book spread from ©2020/2021 Lars Müller Publishers, Zurich, and Daniel López-Pérez.

1.2 INTEGRATED TOOLS

As design problems grow increasingly interdisciplinary, computational strategies have an amplified role in bridging the gap between conceptual ideas and their physical manifestations. Traditionally, the separation of form-giving and form-analyzing tasks has restricted early-stage collaboration between designers, where the involvement of structural engineers often comes later in managing construction details once the peak of design freedom has passed, as illustrated in Fig. 1.2. Mueller mentions how the separation of assignments is further accentuated by the existing computational design tools; architects typically use geometry-based software that allows inquisitive and arbitrary manipulation of form, while engineers rely on analysis-based programs that often require a technical expertise and predefined geometries, thus limiting their applicability during the conception of form [29].

[29] Mueller. 2014. *Computational Exploration of the Structural Design Space.*

However, digital tools, such as Karamba3D, BHoM, and Tekla, attempt to dissolve these barriers by integrating geometrical form and structural behavior, enabling the synthesis of elegant and materially efficient designs using a transdisciplinary model, organizing data sets under a single framework to embrace collaboration. Modeling strategies such as AiCAD embrace this philosophy by

utilizing a single integrated model within a CAD environment, compressing information for all necessary purposes, and facilitating both pre- and post-processing [6]. Besides the separation of domains, Mirra and Pugnale, [27], have remarks on inhibitors that also exist at the interface between the human and machine during the prototypical form development process using computer models and simulated environments:

> "[However] computational and performance-oriented design workflows based on classical parametric modeling and optimization algorithm disrupt rather than support the spontaneous and mostly unconscious humans process of synthesis… Human designers do not need an explicit and analytic definition of design spaces to be able to explore and test formal solutions within such spaces… The processes of human thoughts are complex and flow more like a river than follow a set of instruction."

[6] Block et al. 2016. Integrated design and analysis of structural membranes using the Isogeometric B-Rep Analysis.

[27] Mirra and Pugnale. 2023. Enhancing interactivity in structural optimisation through Reinforcement Learning: an application on shell structures.

While geometric variation and performance are crucial elements that must be meticulously examined through the integration of structural principles into the conceptual design of architecture, the fluid interaction between the abstract concepts and quantitative analysis requires specific proficiency and expertise that cannot be overlooked. This highlights the need for computational tools that not only support but also enhance human creativity, enabling designers to interact fluidly with formal solutions without the constraints of predefined parameters.

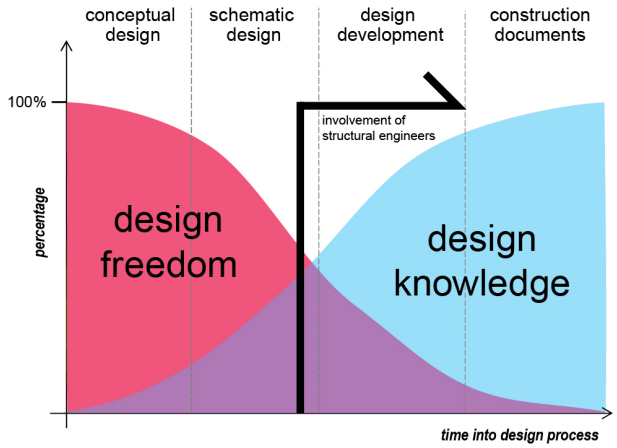


Figure 1.2: Relationship between design freedom and design knowledge in building design projects. Adapted from (Mueller, 2014).

1.3 DIGITAL SEARCH METHODS

The exploration of structural design increasingly relies on digital search methods to address the complexities of modern design challenges. As sustainability and efficiency take precedence, these methods empower designers to iteratively explore equilibrium solutions that define the geometrical and topological arrangement of structural systems while meeting the intricate demands of project specifications, building codes, buildability, and aesthetics. While parametric models allow for real-time manipulation of geometric and spatial configurations, they often fail to consider topological connectivity as a parameter. This limitation constrains the flexibility of design exploration, particularly within discretized systems, where adaptable and innovative connectivity solutions are crucial. Addressing this gap presents a significant opportunity to identify alternative workflows in structural design.

Classical optimization approaches in structural design rely on smooth gradients and well-defined parameters, inherently restricting exploration to predefined search spaces. While effective for deriving unique solutions within a fixed scope, these methods limit the potential for exploring diverse design alternatives. Grammar-based generative approaches present compelling alternatives, operating on rule-based frameworks that do not require predefined performance criteria. Their combinatorial nature enables the generation of diverse and unexpected solutions, bypassing the restrictions of deterministic methods and facilitating the exploration of unstructured design spaces that lack clear boundaries.

However, as the design space generated by grammar-based methods expands exponentially with the number of adopted rules, selecting an appropriate search method becomes essential for effectively navigating and evaluating design alternatives. Combining grammar-based approaches with advanced techniques such as machine learning (ML)—particularly reinforcement learning (RL)—offers a powerful strategy for optimizing within expansive, combinatorial design spaces. RL's capacity to balance exploration and exploitation enhances the discovery of novel and effective solutions while systematically guiding the search toward desired performance criteria. The potential of RL is underscored by its significant breakthroughs in the domain of games, exemplified by DeepMind's AlphaGo, which has cemented games as powerful benchmarks in the field [44]. The strategic complexity of games not only provide a robust platform for evaluating RL performance across different scenarios but also offers well-defined rules, objectives, and rewards that align seamlessly with the RL learning process. This synergy highlights the applicability of RL in navigating similarly structured yet complex design spaces in structural exploration.

[44] Silver. 2016. Mastering the game of Go with deep neural networks and tree search.

1.4 PROBLEM STATEMENT

This thesis aims to contribute to contemporary research on computer-aided workflows at the heart of architectural and engineering design by exploring a generative framework for topology exploration of quad meshes in shell-like structures. The focus is on string-coded operations of formal grammars, which encode rule-based algorithms into alphabetic characters. This study builds directly on Robin Oval's proposal to investigate whether a computational algorithm can effectively produce structured surface patterns with greater flexibility [36]. Current stages of development have produced an encoding strategy; however, no machine process has yet integrated it.

The proposed encoding strategy involves a carefully crafted grammar tailored to the design space of quad meshes, promoting exploration by incorporating sufficient knowledge to modify and maneuver the topology of a predefined mesh. The quad mesh grammar used in this project generates string-coded sequences based on a set of rules, aligning with the iterative nature of RL, where actions are performed sequentially. While innovative, this design approach faces limitations due to the lack of isomorphism between string representations and quad mesh topologies in addition to the absence of context-sensitive operations. RL's ability to balance exploration and exploitation will be crucial for discovering novel and effective rule combination, while fine-tuning the reward function will help guide the agent towards generating outputs that meet specific performance criteria.

1.5 OBJECTIVES

The overarching objective of this thesis is to analyze the role of generative design in facilitating exploration of solution spaces for quad-mesh patterns in shell-like structures. This work deliberately excludes considerations of geometrical and mechanical characteristics to narrow its focus on establishing a simplified model that integrates RL concepts with quad-mesh grammar. By critically evaluating current state-of-the-art ML strategies and implementations, the project seeks to provide the first milestone toward developing a comprehensive framework for optimizing structural pattern topologies through RL. The scope of the research is limited to quad-mesh patterns, chosen for their unique ability to harmonize geometric form and topological representation with structural behavior in surface structures. While constrained by the time available for the Master's program, this thesis investigates a streamlined workflow to explore how RL can effectively navigate and optimize topological characteristics. The outcomes of this study aim to serve as a starting point for future work in generating diverse and high-performing patterns in this domain.

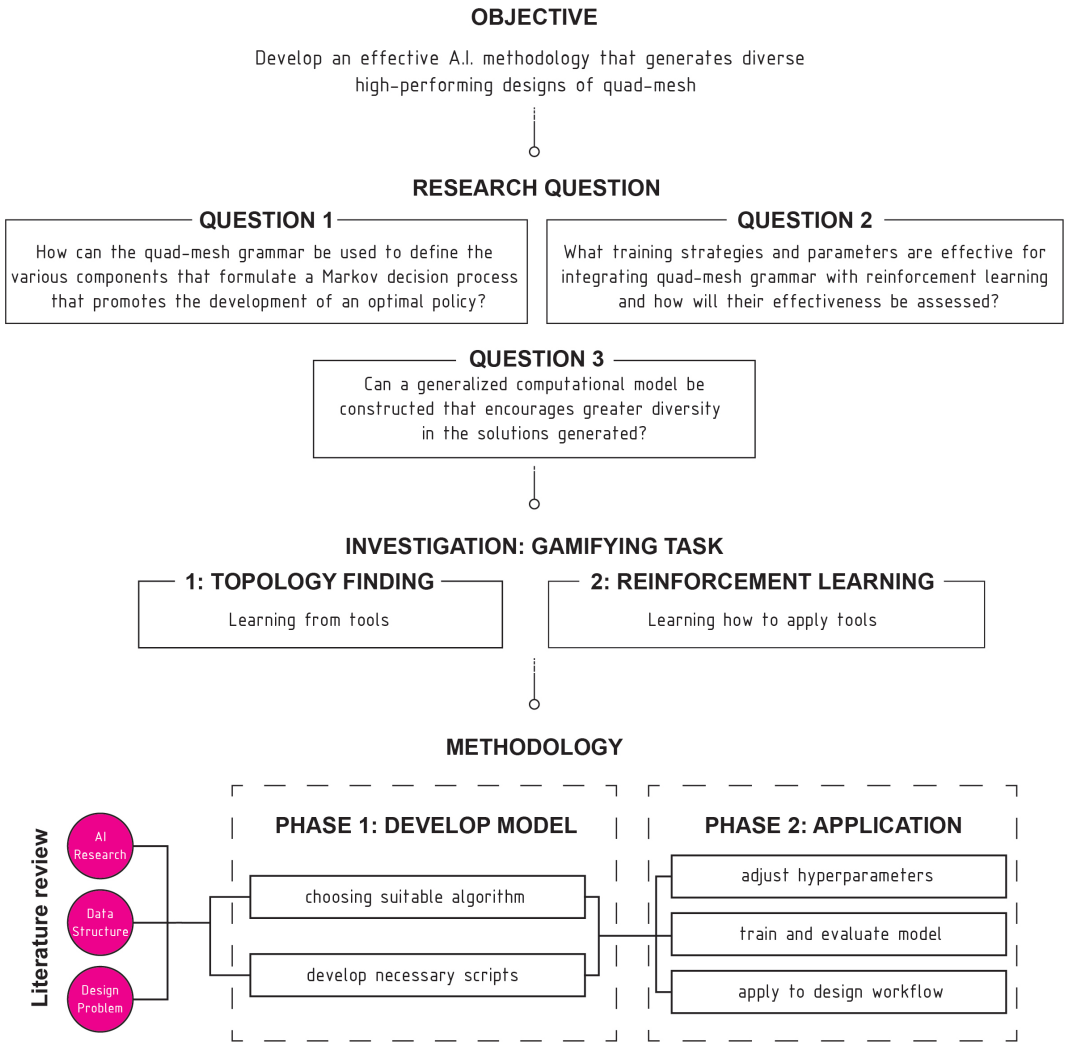
1.6 RESEARCH QUESTION AND THESIS OUTLINE

How can reinforcement learning be effectively integrated with quad-mesh grammar and what method can be applied to explore diverse structural patterns?

- How can the quad-mesh grammar be used to define the various components that formulate a Markov decision process that promotes the development of an optimal policy?
- What training strategies and parameters are effective for integrating quad-mesh grammar with reinforcement learning and how will their effectiveness be assessed?
- Can a generalized computational model be constructed that encourages greater diversity in the solutions generated?

The concept of 'gamifying task' applies to the present work, where tailoring the quad-mesh grammar and fine-tuning its constitutive functions, such as reward functions, becomes an intricate game in itself. The outlined questions will help guide the investigation on the tools to use and how they can be effectively applied to 'gamify' the task in two main phases.

1. Phase 1: Development of a framework
    a) Selection of suitable algorithm
    b) Develop necessary scripts
    c) Establish compatibility of quad mesh grammar and RL
2. Phase 2: Application
    a) Adjustment of hyperparameters
    b) Train and evaluate model
    c) Attempt generalization


OBJECTIVE
Develop an effective A.I. methodology that generates diverse high-performing designs of quad-mesh
RESEARCH QUESTION
QUESTION 1
How can the quad-mesh grammar be used to define the various components that formulate a Markov decision process that promotes the development of an optimal policy?
QUESTION 2
What training strategies and parameters are effective for integrating quad-mesh grammar with reinforcement learning and how will their effectiveness be assessed?
QUESTION 3
Can a generalized computational model be constructed that encourages greater diversity in the solutions generated?
INVESTIGATION: GAMIFYING TASK
1: TOPOLOGY FINDING
Learning from tools
2: REINFORCEMENT LEARNING
Learning how to apply tools
METHODOLOGY
Literature review
AI Research
Data Structure
Design Problem
PHASE 1: DEVELOP MODEL
choosing suitable algorithm
develop necessary scripts
PHASE 2: APPLICATION
adjust hyperparameters
train and evaluate model
apply to design workflow

"Nature has a very basic pattern governing frequencies and energy event magnitudes."

"Nature has mathematic behaviors."

"There is nothing in Nature but Structure."

From *Nature*, Fuller and Applewhite

2

Literature Review

2.1 DESIGN OF PATTERNS

2.1.1 TESSELLATION

Tessellations, or tilings, subdivide an entire plane into non-overlapping modules without gaps. Their origins are deeply intertwined with the development of art, religion, and culture throughout civilization, as well as with the mathematical explorations in the pioneering works of Kepler [16]. The Archimedean tilings, illustrated in Fig. 2.1, showcase three distinct examples of elementary monohedral units and their recombination into semi-regular tessellations. Both regular polygonal tilings —such as triangles, squares, and hexagons— and more complex irregular tessellations —like Kagome and Voronoi patterns— are appealing for civil and material engineering purposes as templates for assembling topologically interlocked systems as seen in Fig. 2.2 [52]. These patterns serve as a foundation for discussing the material and geometrical properties that emerge when translating them into physical and non-Euclidean dimensions, a process that involves the retention of topological regularity and the loss of some geometrical regularity [36].

[16] Grünbaum and Shephard. 1987. *Tilings and patterns.*

[52] Williams and Siegmund. 2021. Mechanics of topologically interlocked material systems under point load: Archimedean and Laves tiling.

[36] Oval. 2019. *Topology Finding of Patterns for Structural Design.*

2.1.2 STRUCTURAL DESIGN

Transforming geometrical patterns into physical structures involves an energetic counterpart, characterized by the equilibrium of tensile and compressive forces. This balance underpins design procedures like form-finding, and its inverse, driven by the feedback between the shape and the internal stress distribution [6]. Patterns also play a crucial role in the digital realization of complex free-form surfaces. When a surface is projected into a distinct parameter domain, discretization occurs in the form of meshes that map the surface. This technique has numerous applications, including spline surfaces and finite element analysis (FEA) [13].

[6] Block et al. 2016. Integrated design and analysis of structural membranes using the Isogeometric B-Rep Analysis.

[13] Floater and Hormann. 2005. Surface Parameterization: a Tutorial and Survey.

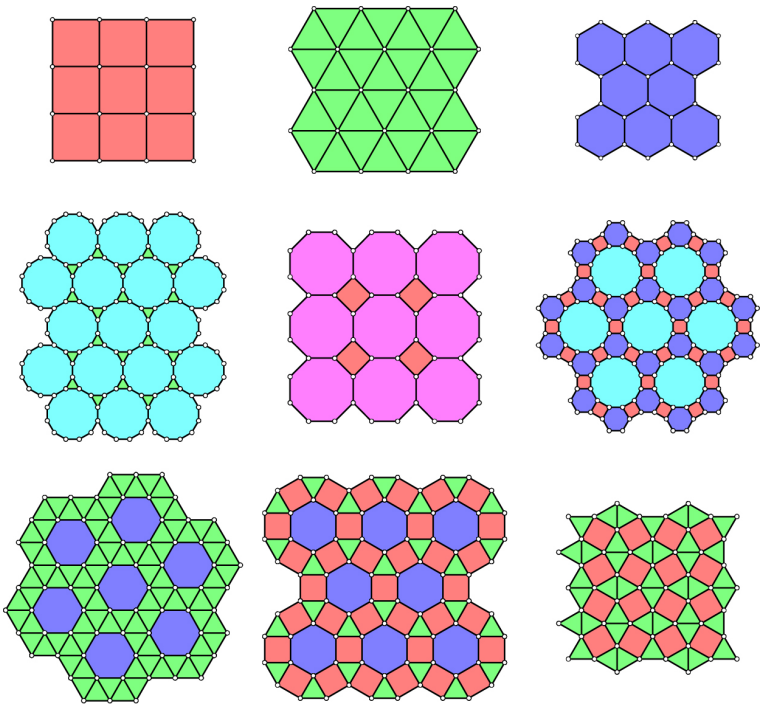


Figure 2.1: Regular and semi-regular 2D Tessellations (Oval, 2019).

MESH PARAMETERS

Meshing is a subdivision technique that affects the accuracy, convergence, and efficiency of finite element model (FEM) simulations. While most commercially available software operate as black boxes, user typically have control over a few key parameters, including the mesh density and element shape. Mesh density determines the size of the stiffness matrix and the level of detail captured in the simulation. For complex geometries, finer mesh sizes are recommended, though they come with increased computational demands. Consequentially, sensitivity analyses are often conducted to find the optimal balance for the convergence of a solution.

The shape feature, which defines the repeated geometrical unit and meshing mechanism used in the discretization, also plays a crucial role. Common geometric forms of mesh elements are illustrated in Fig 2.3 and their selection have implicit impacts on solution accuracy, numerical stability, and mesh organization. For instance, quadrilateral elements create structured grids that are easier to generate and handle, while triangular elements are preferred in unstructured meshes due to their flexibility in accommodating complex shapes [32]. Creating

[32] Okereke and Keates. 2018. *Finite Element Applications.*

effective meshes involves decisions that align with the specific goals of the simulation while minimizing inherent errors in the program.

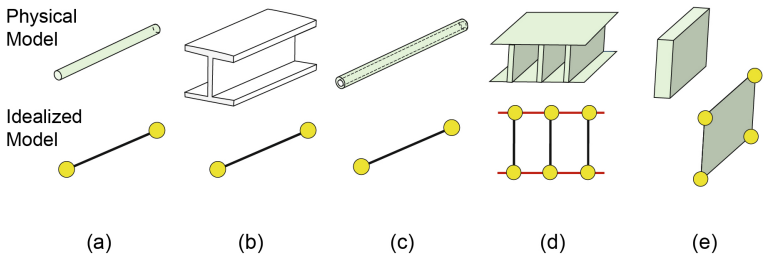


Figure 2.2: Idealized structural elements: (a) bars/trusses, (b) beamsbeams, (c) pipes, (d) webbed section, (e) shear panel elements (Okereke and Keates, 2018).

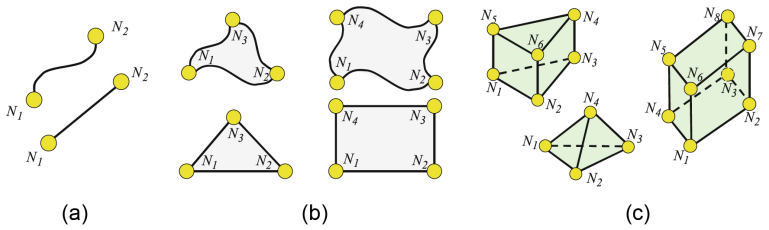


Figure 2.3: Finite elements shapes used in the FEM process: (a) 1D, (b) 2D, (c) 3D (Okereke and Keates, 2018).

MESH GENERATION

Meshing also refers to the intricate set of rules and commands encoded within an algorithm that governs the subdivision process. Delaunay triangulation is often employed to efficiently define a 3D system with a minimal number of triangular elements [10]. In physical problems, sharp edges, notches, interfaces, and holes are often local regions of interest for designers, as these locations can be prone to potential failure mechanisms like stress concentration. A technique known as mesh refinement can be applied at these discrete locations to improve local mesh resolution and achieve simulation convergence. However, this spatial transition can inject irregular vertices into the tessellation, complicating the matrix. It's important to note the challenge of incorporating manual interaction into automated meshing methods, as manually editing connectivity is rarely an intuitive task [7].

[10] Edelsbrunner. 2001. *Geometry and Topology for Mesh Generation.*

[7] Bommes et al. 2013. Quad-Mesh Generation and Processing: A Survey.

2.1.3 SINGULARITY

Singularities imply a breaking of preexisting patterns, which impacts the relationship between local and global orders. These irregularities are not merely theoretical; they emerge across diverse domains, such as in biological morphogenesis, where they shape evolutionary trajectories, and in crystalline structures as dislocations, influencing the overall properties and behavior of organic materials [19]. The presence of a singularity is critical to the discussion of the topological construction of 3D spaces and objects, including the design of buildings and infrastructures; local irregularities in material systems, when uncontrolled, dictate the flow of forces and can lead to inefficiencies or undesirable complications. However, harnessing and controlling singularities provides a powerful means to introduce flexibility in design, enhancing both mechanical and material efficiency, as well as aesthetic innovation.

[19] Isaeva et al. 2012. Topological singularities and symmetry breaking in development.

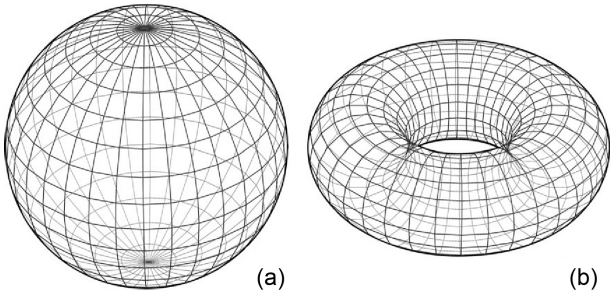


Figure 2.4: Vector field on a surface: (a) singularities on a sphere, (b) field without singularities on a torus (Isaeva et al., 2012).

Pier Luigi Nervi's ribbed system, for instance, eliminated structural redundancy and optimized stability in his concrete structures by controlling local force flows. Nervi's floor design for the Gatti Wool Factory incorporates repeated modular elements achieved through pre-fabrication. The singularities reveal themselves at the intersection of the ribbed stiffeners that follow the isostatic patterns of the principle bending moment, as seen in Fig. 2.5 [4]. While his reliance on labor-intensive form work renders his approach less economically practical today, the advent of digital fabrication techniques offers unprecedented control over complex patterns. These techniques enable the precise customization of unique structural elements, allowing designers to automate segments of construction that were once only feasible through intensive manual processes [2]. Thus, controlling singularities is not just a matter of mechanical efficiency but also a pathway to re-imagining the scope of architectural expression in the digital age.

[4] Billington et al. 2013. The Ribbed Floor Slab Systems of Pier Luigi Nervi.

[2] Ayres et al. 2024. *Fabricate.*

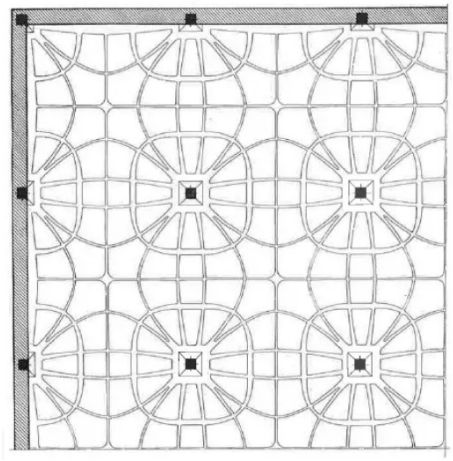


Figure 2.5: Gatti Wool Factory floor plan: isostatic lines (Nervi, 1966)

2.1.4 GEOMETRY AND TOPOLOGY

Further investigation of pattern application in structural design reveals the entwined nature of topology and geometry. In the previous discussion of FEM applications, the geometrical aspects included the whole object and its subdivided parts, while the topology was demonstrated by their connectivity. Together, these aspects define the precise form and interrelationship between components, providing the foundation for a coherent structural system.

SURFACE STRUCTURES

Surface structures, characterized by their curved surfaces, carry loads primarily through in-plane forces, minimizing bending and resulting in a high strength-to-weight ratio. The inherent strength of this structural typology stems from its geometry, which efficiently distributes internal loads along paths that align with the structure's natural shape, offering fundamental stability against buckling. Historically, these forms have been central to significant architectural achievements across many of the world's civilization, such as the Hagia Sophia and the Pantheon (Fig. 2.6), serving not only religious and symbolic purposes but also responding to the need for large, open spaces that accommodate mass gatherings. Their endurance over centuries speaks to the durability of this structural archetype. Visionaries like Gaudi, Candela, and Isler expanded on these favorable characteristics by developing a deep understanding of the mechanical principles behind similar structures, often utilizing physical models to guide their designs [15].

[15] Gohnert. 2022. *Shell Structures: Theory and Application.*

Today, a resurgence of interest in similar typologies and other antique forms, such as vaults, is driven by advancements in computational design and digital fabrication technologies, which have made the realization of non-standard geometries far more feasible. However, designing and fabricating these structures presents significant challenges due to the intricate relationship between structural analysis and physical realization. The highly statically indeterminate behavior of shells complicates force distribution, making accurate discretization essential to capturing both their geometry and behavior. A well-chosen discretization pattern ensures that the model closely approximates the shell's curvature, essential for precise simulations. Factors like mesh regularity, density, and placement of singularities are critical to achieving computational efficiency, especially in localized regions. This precision not only optimizes for material usage and structural performance but also for ensuring the practicality of fabrication.

Figure 2.6: Historical precedents: Armadillo Vault (Rippman et al., 2016), Pantheon ("Pantheon (Rome) - Right side and front" by NikonZ7II, licensed under CC BY-SA 4.0.), Deitingen Süd Raststätte ("File:Deitingen Sued Raststaette, Schalendach 03 09.jpg" by , licensed under CC BY-SA 3.0.), Hagia Sophia (image by author).

Heinz Isler's meticuluous approach, as explained by himself in [30], exemplifies the complexities involved. Although time consuming, Isler calculated precise forms, defining hundreds of coordinates to achieve the desired shape, with no two curves alike in asymmetric constructions. In one particularly challenging project, he had to manage 400 points, where adjusting a single point affected all the others. Using specialized measuring devices, Isler derived curvature data from physical experiments, transferring these measurements directly to the construction site. One such completed structure is illustrated in Fig. 2.6. While Isler relied on painstakingly manual calculations to define and translate curvature data into construction-ready forms, modern tools provide a much more streamlined workflow.

[30] Nordenson. 2008. *Seven Structural Engineers: the Felix Candela Lectures.*

The Armadillo Vault by the Block Research Group (BRG) exemplifies how contemporary digital tools extend and transform the principles pioneered by engineers like Isler. The tessellation of the structure, shown in Fig. 2.7, begins with a computationally optimized thrust surface derived from local force flow. The design aligns the tessellation pattern with the trajectories of internal forces, ensuring stability and load-bearing efficiency. The quad-mesh representation of this surface serves to guide the layout of the tessellation but also identifies areas of singularity where the force trajectories converge or diverge.

Unlike Isler's monolithic shells, the discretization of surface structures in the Armadillo Vault serves a dual purpose: enhancing the understanding of structural performance and improving construction feasibility. By experimenting with these patterns, the designers can adjust spacing and alignment to balance material efficiency with structural performance. Contemporary technology also enabled the direct translation of digital designs into fabrication-ready components. Unlike Isler's labor-intensive process of deriving curvature data, BRG used a combination of automated and manual workflows to generate geodesic curves and precise joint geometries for the tessellation [41]. These patterns were further refined to ensure a staggered, interlocking configuration of voussoirs. Near singularities, digital tools facilitated nuanced modifications to the patterns, allowing the structure to respond dynamically to localized force flows without disrupting the overall visual rhythm. The vault's tessellation patterns are a result of integrating computational design, structural analysis, and digital fabrication constraints, exemplifying a modern approach to the timeless interplay between form, function, and aesthetic.

[41] Rippmann et al. 2016. The Armadillo Vault: Computational Design and Digital Fabrication of a Freeform Stone Shell.

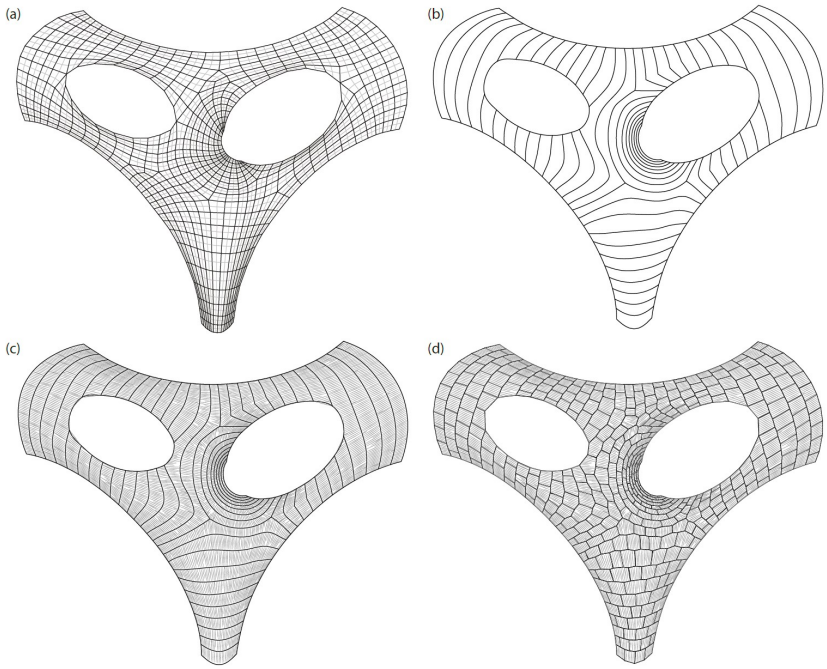


Figure 2.7: Armadillo Vault tessellation patterns (Rippman et al., 2016).

QUAD-MESH

The most common polygonal mesh representation for surfaces are triangles and quadrilaterals, with the choice between them depending on the specific application. This section highlights some of the benefits and disadvantages of using quad-meshes in the context of designing shell-like structures.

By definition, a quad-mesh is a tessellation composed entirely of quadrilaterals. However, there is a continuum of variations within this category, including semi-regular and unstructured quad-meshes, as illustrated in the top row of Fig. 2.8. In a regular quad-mesh, verticies typically have a valence of four. Three types of regular boundary vertices can be identified: regular non-corner boundary (three valence), convex corner boundary (two valence), and concave corner boundary vertex (four valence). The paths of edges connecting an irregular vertex to either another singularity or to the boundary are referred to as separatrices (integral lines), shown in the bottom row of Fig. 2.8 [7].

One advantage of quad-mesh is their bidirectional nature, which align well with cross fields that have two parameterization directions, such as principal curvatures and principal stresses. In contrast, triangle meshes require an arbitrary third direction [36]. Additionally, triangular meshes are always flat and

convex, supporting linear interpolation of discrete functions defined at vertices and easily projected onto a plane [7]. For numerical analysis, such as FEM, approximation using isometric quads is computationally more efficient as it reduces the total number of elements needed compared to triangular meshes [7]. However, a disadvantage of using quad-meshes is that they can be more challenging to generate and manipulate, particularly when creating complex geometries. This complexity can lead to difficulties in maintaining a regular mesh structure, potentially resulting in irregularities that complicate both the design and subsequent analysis.

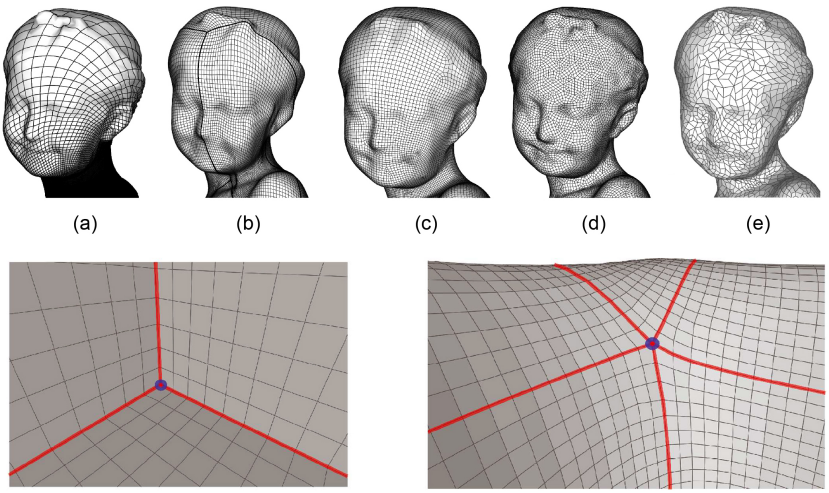


Figure 2.8: Top row: Quad-mesh categories: (a) regular, (b), semi-regular, (c) valence semi-regular, (d) unstructured (e) unstructured and irregular. Bottom row: Left: Three separatrices, stemming from a valency 3 singularity; Right: Five separatrices, stemming from a valency 5 singularity (Bommes et al., 2013).

Another significant difference between triangular and quadrilateral meshes is their fabrication feasibility, especially in steel and glass gridshells. Triangular elements are inherently planar, which simplifies their fabrication and assembly. In contrast, quadrilateral elements are often non-planar, especially when used in complex geometries, which increases the fabrication cost due to the need for curved or twisted elements. Additionally, the use of triangular glass panels, which are typically cut from quadrilaterals, can introduce redundancy due to the need to accommodate the quadrangular shapes [30].

PERFORMANCE METRIC

The quad-mesh framework not only informs the tesselation layout but also serves as a foundation for evaluating the relationship between topological patterns and structural performance. The arrangement of singularities, edge connections, and face orientations within these patterns plays a critical role in determining how efficiently forces are distributed across the structure. This insight underscores the importance of performance metrics in the design process.

Assigning specific performance metrics to design patterns is essential for evaluating their effectiveness across various objectives. This flexible and versatile approach allows designers to apply these metrics to a wide range of structural forms and criteria. By systematically assessing how changes in topology and geometry influence overall performance, this method opens up new possibilities for search and optimization techniques in the generative design of singularities in structural patterns [31].

[31] Nourian et al. 2023. Generative Design in Architecture: From Mathematical Optimization to Grammatical Customization.

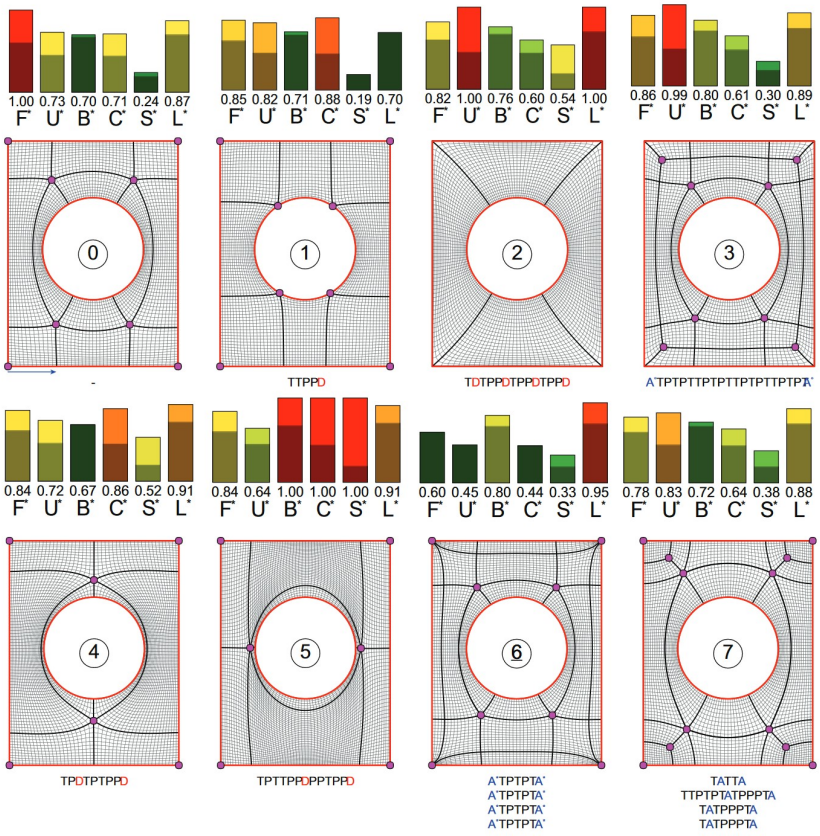


Figure 2.9: Performance evaluation and comparison of varying quad-mesh patterns for the gridshell of the British Museum (Oval et al., 2023).

In previous works, the gridshell of the Great Court Roof at the British Museum was used as a case study to examine the impact of topology on multi-objective trade-offs, as shown in Fig. 2.9 [37]. The steel beams and glass were arranged in a quad-mesh pattern, with various grammar rules used to generate variations of the original layout. The goal was to create distinctly different patterns in terms of topology and performance, emphasizing the influence of pattern topology on the examined objectives. Structural performance was evaluated using FEA software, focusing on weight minimization and both ultimate (ULS) and serviceability (SLS) requirements. This included second-order analysis for stiffness, strength, and stability, alongside fabrication objectives such as average panel curvature, skewness, and edge length standard deviation. The six performance metrics were normalized and visualized using a color gradient from red (worst) to green (best). The study concludes that no single design excelled in all metrics, leaving the designer to explore the Pareto front of trade-offs to select the most suitable design.

[37] Oval et al. 2023. A vector encoding for topology finding of structured quad-based patterns for surface structures.

2.2 COMPUTATIONAL METHODS

Computational methods have revolutionized the way designers and engineers approach complex problems, enabling the exploration of innovative solutions that were previously unattainable. These methods have their roots in design sciences, which formalize the theoretical and methodological foundations of human reasoning. Building on these principles, computational tools like generative design have opened new frontiers in structural design, allowing for the automated generation and optimization of complex structures. This section explores these key aspects, beginning with a theoretical background of automated reasoning systems, followed by machine learning applications in structural design and an examination of the role of reinforcement learning in modern structural design workflows.

2.2.1 THEORETICAL BACKGROUND

COMPUTER-AIDED DESIGN FOUNDATIONS

The evolution of computational methods in structural engineering during the 20th century was marked by significant milestones, beginning with the development of numerical analysis techniques such as the finite difference and finite element methods. The advent of digital computers and programming languages enabled numerical solutions to complex problem, laying the groundwork for the application of computational design on a larger scale with greater precision. In parallel to the emergence of structural optimization techniques, the 1960s saw rise of geometry-based computer-aided design (CAD) systems powered by

advancements in graphical displays. Digital 2D drawings and basic 3D wireframe modeling emerged as practical tools, evolving by the late 1970s and early 1980s to include sculptured surfaces defined by B-splines, Bezier splines, and other parametric surface descriptions [45]. The capability to represent complex geometries, significantly broadened the utility of CAD in structural engineering, however, these geometry-focused systems remained limited in addressing higher-level design aspects, such as the interplay of structural features or the integration of functional and behavioral constraints.

[45] Smithers. 1985. AI-based versus geometry-based design or Why design cannot be supported by geometry alone.

MODELING CONTEXTUAL REASONING PROCESSES

Incorporating reasoning about structural behavior into purely geometry-based computational models required bridging multiple levels of abstraction to explicitly define collective features and their sub-features in structural design. This challenge aligned with the emergence of early AI techniques, which sought conceptually and computationally efficient knowledge-encoding frameworks, marking a transition toward methods capable of richer structural descriptions [45]. According to Smithers, the early successes of AI-based approaches, grounded in mathematical optimization and constraint satisfaction, spurred their evolution along three interdependent trajectories. The first focused on modular organization of diverse knowledge sources, enabling the representation, and reasoning of domain-specific knowledge. This led to innovations such as blackboard systems, expert systems based on production rules, and structured object representation systems, which supported interactive inferencing, logical rule coordination, and hierarchical organization of design information, respectively [45]. The second trajectory aimed to transcribe the cognitive problem-solving procedures to automate reasoning, emphasizing heuristic knowledge inherent in engineering design. The third trajectory explored integrated system architectures, coupling diverse reasoning systems, automating their control, and facilitating human interaction, marking a significant advance toward comprehensive design support systems. Together, these developments underscore the foundational role of early AI in transforming computational tools for engineering design.

Building on these early advancements, knowledge-based models (depicted in Fig. 2.10) in structural engineering began to emerge, offering targeted solutions to specific design and analysis challenges. For example, SACON, adapted from the EMYCIN framework, acted as a consultation agent to guide finite element analysis [3]. By assisting users in decomposing structures into substructures and estimating stresses and deflection, it recommended analysis strategies tailored to engineering tolerances. Similarly, systems like HI-RISE facilitated preliminary design for high-rise buildings, focusing on lateral and gravity load-resisting systems, while SPERIL extended expert systems to earthquake damage assessment, incorporating uncertainties in structural damage evaluation [24] [20]. Although

[3] Bennett et al. 1978. SACON: A Knowledge-Based Consultant for Structural Analysis.

[24] Maher. 1985. HI-RISE and beyond: directions for expert systems in design.

[20] Ishizuka et al. 1982. Rule-based damage assessment system for existing structures.

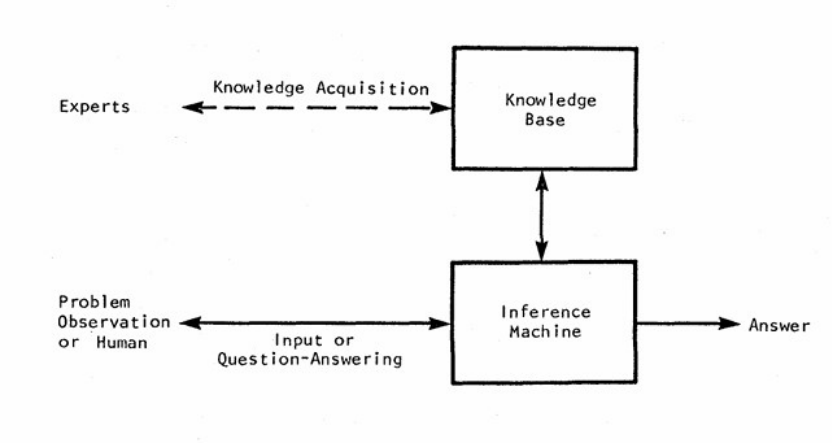


Figure 2.10: Workflow of expert systems (Ishizuka et al., 1982).

these systems were limited in their capabilities —providing basic solutions that often required refinement by experienced designers— they highlighted the potential of structured knowledge bases and rule-based reasoning to automating aspects of complex design processes. However, their broader impact was constrained by the technological limitations of the era, including underdeveloped AI techniques and challenges in extracting and integrating expertise from human professionals. Despite these obstacles, these prototypes laid crucial groundwork, serving as precursors to advanced systems that followed in later decades.

The 1990s introduced neural networks (NN) and creative design frameworks as new paradigms for addressing the complexity of reasoning in design through pattern recognition techniques. For example, Sun et al. applied NNs to simple tasks such as concrete beam design and plate analysis, noting their potential for tackling more difficult or time-intensive engineering problems [48]. Gunaratnam and Gero shifted the focused to discovering conceptual relationships that inform preliminary design decisions, emphasizing the need for computational models to support not only solution searches but also exploratory processes that restructure knowledge [17]. Geo further argued that design is not merely problem-solving but an evolving, goal-driven process influenced by physical realities, designer perceptions, and computational limitations [14]. By framing design variables as structure, behavior, and function, he underscored the importance of integrating these dimensions into computational models to reflect the dynamic and contextual nature of design [14]. This set the stage for the transition into the 21st century, where models increasingly aimed to capture the complexity, creativity, and adaptability required by modern structural engineering applications.

[48] Sun and Vanluchene. 1990. Neural Networks in Structural Engineering.

[17] Gunaratnam and Gero. 1994. Effect of Representation on the Performance of Neural Networks in Structural Engineering Applications.

[14] Gero. 1994. Towards a model of exploration in computer-aided design.

2.2.2 GENERATIVE STRUCTURAL DESIGN

While computational tools have demonstrated significant potential for accelerating design exploration, digitalizing the design process alone is insufficient to ensure the generation of truly novel configurations or to fully explore the design space [18]. Generative design addresses this limitation by combining computational rigor with creative flexibility, transforming the art and science of design to tackle modern design challenges with greater accessibility and relevance.

[18] Ioannis and Corentin. 2020. Design space exploration through force-based grammar rule.

Given the diverse interpretations of generative design across multiple disciplines —each emphasizing distinct goals, processes, and technologies ranging from algorithm exploration and optimization to creativity and design automation— this work adopts Nourian et al.'s definition as applied in the architectural domain. Their definition frames generative design as a spectrum of systematic approaches for exploring, synthesizing, and deriving design configurations using mathematical, grammatical, and gamified processes, illustrated in Fig. 2.11 [31]. These methods enable systematic navigation from abstract functional requirements to concrete forms by mapping design spaces and their performance duals using tailored simulations for optimization, customization, or participatory needs.

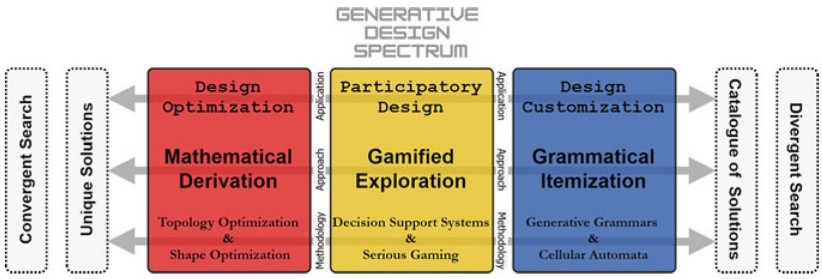


Figure 2.11: Generative design spectrum (Nourian et al., 2023).

MATHEMATICAL DERIVATION FOR OPTIMIZATION

Mathematics-based approaches discover optimal configurations of discrete variables using numerical and gradient-based derivations. Shape optimization, for instance, leverages Dirichlet energy integrals to minimize energy states refining the material usage, while topology optimization redistributes materials within planar or volumetric boundaries under governing load cases to achieve desired structural behavior [31]. Designs reveal themselves as objective functions that minimize cost or maximize utility converge based on the selected performance criteria and constraints. For example, Isozaki and Sasaki performed evolutionary structural optimization and shape-analysis techniques to design the Qatar

National Convention Centre [21]. Laarman applied similar mechanics-based techniques, developed in the automobile industry, to sculpture his Bone Chairs to achieve both functional efficiency and aesthetic innovation [22]. These applications, illustrated in Fig. 2.12, demonstrate one of many capabilities of modern computational tools to generate efficient, high-performance solutions tailored to diverse design challenges.

[21] Januszkiewicz and Banachowicz. 2017. Nonlinear Shaping Architecture Designed with Using Evolutionary Structural Optimization Tools.

[22] Laarman. 2017. *Joris Laarman Lab*.

Figure 2.12: Qatar National Convention Centre ("IMG_6478" by trevor.patt, licensed under CC BY-NC-SA 2.0) and Bone Chair ("Laarman Bone-Chair 01" by Saimmad, licensed under CC BY-SA 4.0)

Despite the strength of numerical models at optimizing measurable performance metrics, they often struggle to formulate subjective or intangible criteria —such as aesthetic, culture, and comfort–– as explicit mathematical functions. This disadvantage is exacerbated during early conceptual phases where the argument for the construction of space and volumes lacks. When a set of descriptions are decided, reliance on explicit parameters can inadvertently narrow the focus of optimization, excluding broader considerations like design diversity or user experience [1]. Additionally, gradient-based methods, especially in topology optimization, involve time-consuming, resource-intensive calculations. While effective for single-objective solutions, they struggle with multi-objective scenarios, often converging to local optima. The relaxation of discrete variables can also necessitate further post-processing for practical use. Moreover, mathematical derivation limits creative exploration, prioritizing unique solutions over diverse design alternatives, making it difficult to tackle open-ended design problems that value exploration and customization. Combining these methods with generative approaches, such as grammatical or gamified systems, can help balance optimization with creativity [31].

[1] Adeli. 1986. Artificial intelligence in structural engineering.

GRAMAMTICAL ITEMIZATION

Grammatical itemization, lying at the opposite end of the spectrum in generative design, uses rule-based frameworks to flexibly explore and produce diverse ranges of design alternatives. Rule-based design is built from the concept of formal grammars, introduced by Chomsky in the 1950s, which applies a finite set of rules to words, allowing for the creation of an infinite number of sentences [9]. Lindenmayer's L-systems expanded on formal grammars to algorithmically describe the growth of plants with iterative rules that generate corresponding shapes, as shown in Fig. 2.13, depicting the potential for geometry generation [40]. Stiny and Gips' introduction of the shape grammar formalism, to encode a language of shapes to the alphabet of symbols used in formal grammars, inspired subsequent derivatives that embedded structural knowledge in the rules [46]. These examples include the shape annealing approach for 2D trusses, trans-typological exploration of bridge designs, as well as force grammar approach that incorporated graphic statics to generate 3D spatial structures (Fig. 2.14) [43] [29] [23].

[9] Chomsky. 1956. Three models for the description of language.

[40] Prusinkiewicz et al. 1997. L-systems: from the Theory to Visual Models of Plants.

[46] Stiny and Gips. 1972. Shape Grammars and the Generative Specification of Painting and Sculpture.

[43] Shea et al. 1997. A Shape Annealing Approach to Optimal Truss Design With Dynamic Grouping of Members.

[29] Mueller. 2014. *Computational Exploration of the Structural Design Space.*

[23] Lee et al. 2016. Automatic generation of diverse equilibrium structures through shape grammars and graphic statics.

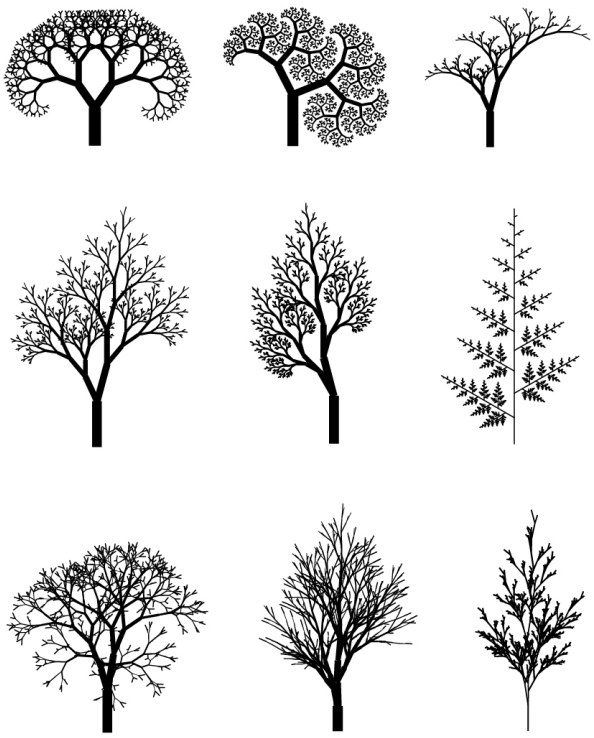


Figure 2.13: Application of L-systems to the modeling of plants (Prusinkiewicz et al., 1997).

While grammar-based methods do not require predefined performance criteria, its effectiveness depends on the quality and completeness of the defined rule sets. The rules embed constraints directly into the grammar, ensuring that all generated designs meet basic requirements, such as constructability or adherence to specific-geometric parameters. If these rules are poorly conceived or inflexible, the resulting combinatorial design space is predisposed to undesirable outcomes [29]. Grammars with numerous or overly flexible rules can lead to a combinatorial explosion of alternatives, making it difficult for designers to evaluate or navigate the space of possibilities as well. Without additional tools for filtering or prioritizing designs, the process risks becoming unmanageable or under-explored. Additionally, the lack of integration of performance criteria necessitates supplementary analysis tools to assess performance metrics [23]. Although algorithmic search or AI-driven exploration can mitigate these challenges, they require sophisticated implementations that may add complexity to the design workflow.

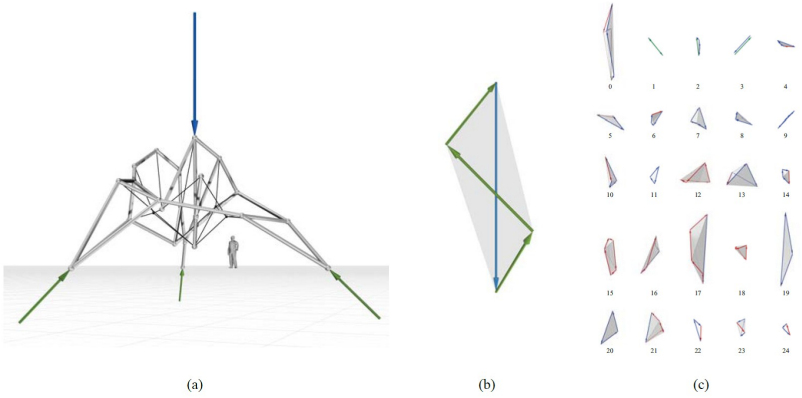


Figure 2.14: 3D Spatial structure generated using force grammars (Lee et al., 2016).

2.2.3 MACHINE LEARNING

Machine learning (ML) is a subset in AI that has gained torrential prominence in recent years for their ability to recognize patterns within data under conditions of uncertainty. They have found broad scientific application, including structural engineering, to predict and assess structural conditions and performance [47]. Furthermore, ML algorithms enhance and refine generative design solutions efficiently by identifying patterns in structural performance or user preferences, thereby accelerating innovation and improving outcomes.

[47] Sun et al. 2021. Machine learning applications for building structural design and performance assessment: State-of-the-art review.

DEEP ARCHITECTURE

Their current popularity across a wide variety of disciplines can be attributed to Deep Learning (DL), a technique developed in the breakthrough of computer vision alongside advancements in processing capabilities for larger models. In 2012, AlexNet outperformed contemporary ML techniques in ImageNet, a large-scale classification challenge, by using convolutional neural networks as opposed to fully connected neural networks. Neural networks (NNs) are graph-based decision-making models composed of a vast array of computational relationships that connect the input and output layers through multiple layers that attempt to replicate the structures of biological neurons found in the brain; the term 'deep' commonly referring to the depth, or number of hidden layers, ranging from a single digit to thousands, depending on the complexity of the given task [39]. The strength of convolutional networks was its superior inductive bias that embodied prior knowledge in its architecture; it exploited spatial relationships between nearby pixels and processed local image regions independently, using shared and fewer parameters across the whole image [39]. Although there was a general trend in the following years to keep improving the performance of classification tasks by increasing the depth of hidden layers, as shown in Fig. 2.15, subsequent experiments began to reveal that indefinite depth increments over-complicated the system making them difficult to train. This led to modifications in the form of residual connections and normalization layers, eventually catalyzing application of DL models in a broader range of topics, such as object detection through the application of transfer learning [39].

[39] Prince. 2023. *Understanding Deep Learning.*

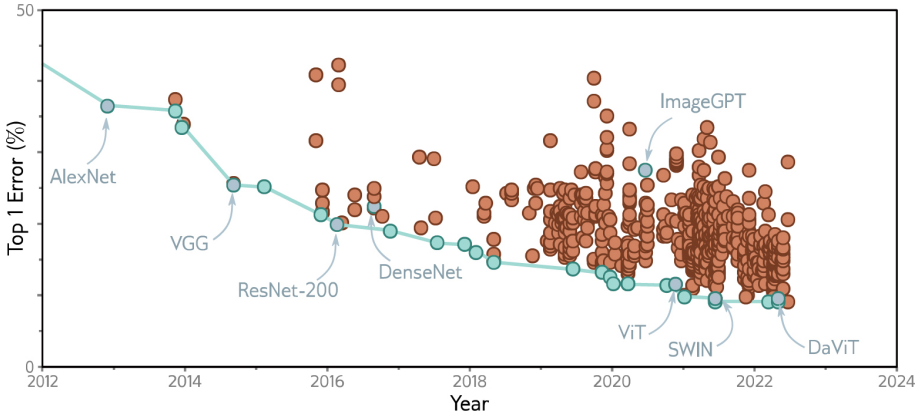


Figure 2.15: ImageNet performance (Prince, 2023).

Malaga (2022), [25], provides a thorough review of the historical development of AI's integration into structural design, which he defines as:

[25] Málaga-Chuquitaype. 2022. Machine Learning in Structural Design: An Opinionated Review.

> "...the process by which the number, distribution, shape and size of structural elements and their connectivity is determined so that a given design objective is achieved while meeting several constraints and serviceability and resistance."

According to Malaga, AI has the potential to streamline the often-time-consuming process of searching for solutions, where engineers frequently settle for the first sub-optimal design that satisfies all hard constraints. This design process typically involves managing hundreds of structural sections and referencing thousands of pages of building codes, which can be overwhelming and inefficient. In such contexts, ML algorithms are particularly beneficial to civil engineers as they allow access to complex, multi-dimensional spaces that are defined by the multiple objectives, often far beyond what human cognition alone can manage.

However, despite the proven benefits of AI, Malaga highlights that there are still limited efforts to use these tools to fully automate structural design. Most existing implementations tend to focus on optimizing individual structural sub-assemblies, without fully considering the building structure as a cohesive and integrated unit. He also identifies significant challenges in the parameterization of entire building structures, which require larger datasets and considerable processing power to account for the numerous possible element positions, sizes, and connection types within interconnected systems. In contrast, shells, vaults, and other spatial structures have provided fertile ground for design optimization, as these forms can be comparatively easier to discretize and are typically single-layered, making parameterization much more straightforward.

In the sections that follow, examples that emphasize the potential roles of 'engineering intuition' and 'creativity' in applying ML to the design of spatial structures are reviewed. These examples explore integrated computational strategies with state-of-the-art workflows that encourage rethinking traditional design approaches through the use of meta-heuristics, showcasing the potential for ML to transform the field of structural design.

EXAMPLE 1: DESIGN SUBSPACE LEARNING (DANHAIVE AND MUELLER, 2021)

Danhaive and Mueller highlight the inefficiencies of manual editing and optimization in parametric design, which often fail to effectively explore design variations or address qualitative aspects critical in architectural and structural design. To address this, they propose integrating generative modeling and AI, specifically conditional variational auto-encoders (VAEs), within a performance-driven framework to enhance creativity without constraining designer decisions. Their methodology employs an iterative sampling algorithm that prioritizes high-performing samples through filtering gates while maintaining diversity for

downstream exploration. By using surrogate models to approximate nonlinear data manifolds and iteratively refining samples based on true objective functions, this approach ensures a balance between exploration and exploitation of the design space, guided by performance thresholds and growth rates.

Once a dataset is collected, a performance-conditioned VAE creates a reduced, continuous latent space for intuitive exploration of design variations. The VAE employs NNs for encoding and decoding, minimizing reconstruction loss while conditioning on normalized evaluation scores to disentangle performance contours. This enables designers to explore sub-optimal solutions that may satisfy unformulated objectives while balancing ease of exploration and diversity in design options. However, this approach requires large datasets, making it most suitable for cases where thousands of solutions can be simulated.

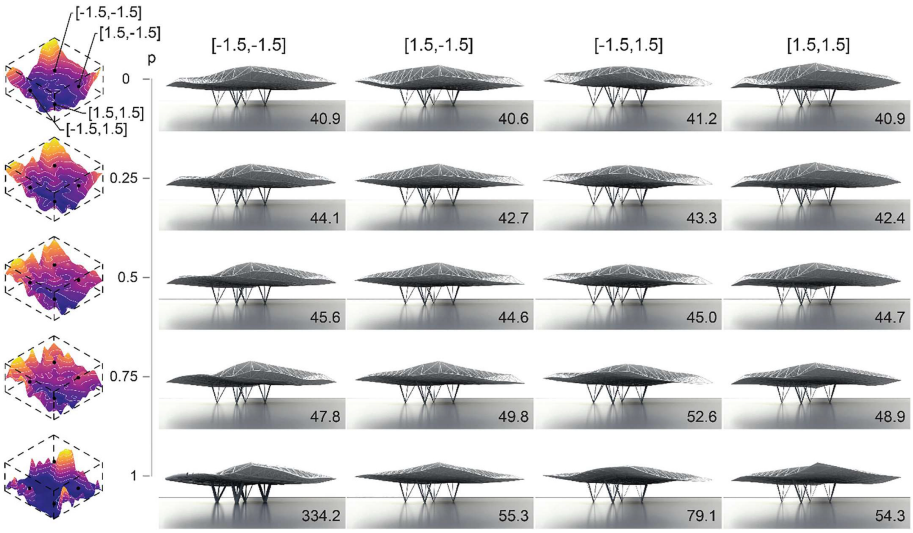


Figure 2.16: Morphological and performance evolution of 4 designs in the latent space as p increases from 0 to 1 (Danhaive and Mueller, 2021).

In a case study of a long-span roof, the algorithm evaluated design candidates based on material efficiency relative to the structure's footprint, using Karamba3D for optimization and FEA for validation. The latent space captured diverse roof configurations that met structural performance requirements, with designs optimizing truss depth and maximizing efficiency in high-bending moment areas. The study demonstrates how AI-driven generative frameworks can integrate qualitative and structural considerations, ensuring both creativity and performance in design processes.

EXAMPLE 2: FORM FINDING AND EVALUATING THROUGH ML (ZHENG, 2019)

Graphic statics is a geometry-based structural design and analysis method that has gained renewed attention due to advancements in computational power. Design variations can be generated under equilibrium, where the interior faces of the polyhedral force geometries can be adjusted through subdivision rules to modify the complexity of the form. Zheng's work integrates aesthetic considerations into the framework for exploring design variations by bridging graphic statics to an architect's subjective preferences through NNs to develop a ML model [53]. Zheng analyzed the ML's capability to detect hidden patterns that connect personal design choices to structural performance using a dataset of 400 force and form diagram pairs generated using randomized boundary tetrahedrons and iterative subdivision rules. Zheng adopted a scoring mechanism to record the architect's preferences, further simplifying the data collection process by grouping the samples into 200 batches, each consisting six forms, to account for variations during selection due to mood and patience. Two additional comparative tests were conducted, in which the tester selected the most complex and the simplest forms from each batch to help quantify personal preferences into more objective traits.

[53] Zheng. 2019. Form Finding and Evaluating Through Machine Learning: The Prediction of Personal Design Preference in Polyhedral Structures.

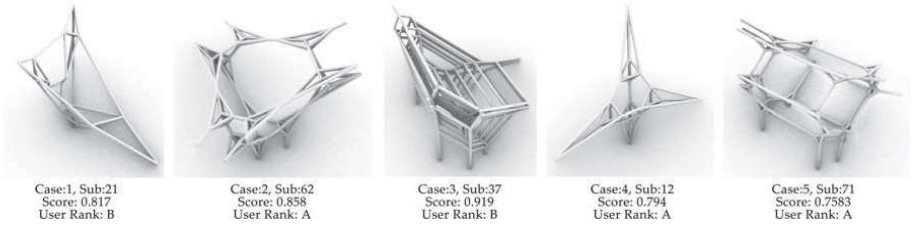


Figure 2.17: Form finding by the highest score for 3 random boundary conditions (Zheng, 2019).

To train the model, a vector-based NN was used to map form-to-preference scores. The polyhedral form data was encoded into 21 numbers, representing the boundary tetrahedron coordinates and subdivision rules. The output was a single real number reflecting the form's score. The NN architecture comprised of two hidden layers with 50 neurons each, using a sigmoid activation function. The model was trained using a mean squared error (MSE) loss function to minimize the gap between predicted and actual scores. The comparative models were successfully verified, with the trained NN assigning appropriate scores to both visually complex and simpler forms, demonstrating its ability to evaluate polyhedral designs. However, the preference model required additional validation for taste, so the tester re-evaluated the forms using a scale from A to D, where A represented perfection and D indicated dislike. After further investigation, the

NN successfully learned the specific architect's design preferences, offering a novel method for quantifying subjective preferences, which is often challenging to articulate through numerical data alone.

2.2.4 REINFORCEMENT LEARNING

Reinforcement learning (RL) is a distinct subset of AI that differs from supervised and unsupervised learning by focusing on training agents to map situations to sequential decisions through interactions with their environment. Formally, this process is framed as a Markov Decision Process (MDP), providing a mathematical foundation for decision-making as shown in Fig. 2.18 [49].

[49] Sutton and Barto. 2018. *Reinforcement learning: an introduction, Second edition.*

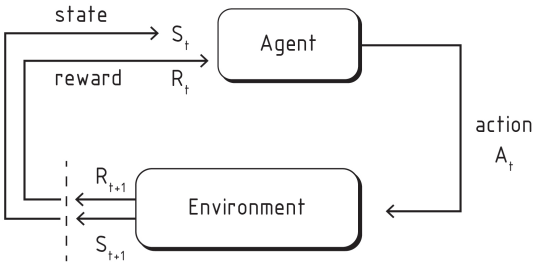


Figure 2.18: Markov decision process. Adapted from (Sutton and Barto, 2018).

This method aligns with generative design methods categorized under gamified exploration in Fig. 2.11. Gamified exploration focus on participatory environments where designers iteratively navigate decision-making processes using predefined rules and simulations to evaluate the consequences of their actions [31]. Reinforcement learning enhances gamified exploration by integrating artificial agents that autonomously explore design spaces or assist human participants in decision-making. However, the success of RL in such contexts relies on carefully crafted game mechanics, scoring systems, and simulation models, as well as the ability to balance abstraction and complexity. Combining RL with other generative design approaches, such as grammatical itemization, offers a promising pathway for addressing these challenges and expanding the effectiveness of generative design across diverse applications.

STATES AND OBSERVATIONS

The state encompasses all relevant descriptions of the environment, whereas an observation may convey an incomplete version of the state, omitting certain information. The data relayed to the agent can take various forms, such as a finite graph representing a planar truss [35], pixel-based image observations [28],

[35] Ororbia and Warn. 2023. Design Synthesis of Structural Systems as a Markov Decision Process Solved With Deep Reinforcement Learning.

[28] Mnih et al. 2013. Playing Atari with Deep Reinforcement Learning.

or numerical lists of vertices, edges, and faces for a shell structure [50].

[50] Tam et al. 2022. Performance-informed pattern modification of reticulated equilibrium shell structures using rules-based graphic statics, CW networks and reinforcement learning.

ACTION SPACES

The action space defines all possible move-sets available to the agent, with its nature varying depending on the environment. In Atari games, for example, the input is discrete and limited by the controller options, whereas, an autonomous driving agent operates within a continuous action space, with an infinite number of possible actions. Different design tasks prioritize specific characteristics, such as high safety performance in intelligent transportation or high diversity and accuracy in recommendation systems [54]. These varied requirements necessitate different algorithms tailored to the complexity of the actions performed.

[54] Zhu et al. 2021. An Overview of the Action Space for Deep Reinforcement Learning.

For an agent to achieve its expected goal, all effective actions must be valid and complete. If certain actions are not properly programmed, the agent may face a higher probability of failure in completing the design task [54].

REWARD FUNCTION

The reward function is a critical component in the performance of any RL model, as the agent's goal is to maximize cumulative reward over its trajectory. In RL, it is generally assumed that a predefined reward function exists, as is the case in board games where performance is easily measured through outcomes such as wins, losses, or points. Once the goal is defined, a key challenge lies in how to allocate rewards. Shaped rewards offer incremental feedback for states that progress toward the end goal, even if the policy has not yet reached a complete solution. However, this approach can introduce bias into the learning process. In contrast, sparse rewards provide feedback only at the goal state, offering no intermediary rewards. Additionally, discounting can be employed to regulate reward distribution across a sequence of actions, reducing the influence of rewards from earlier actions.

POLICIES AND VALUE FUNCTIONS

A policy, π, either deterministic or stochastic, defines the learning agent's behavior, linking perceived states of the environment to actions to be taken when in these states. The reward signal defines the short-term goals of how good or bad an outcome is making it the primary basis for altering the policy. At each time step, the environment sends to the agent a signal symbolizing the immediate intrinsic desirability of an environmental state. On the other hand, a value function specifies what is good in the long-run. The value of a state corresponds to the total amount of reward an agent can expect to accumulate over the future, by taking into account the states that are likely to follow initialization

and the rewards available in those states. Thus, making value functions harder to determine. Both policy and value functions are the learning objectives in RL.

The value function of a state, s, under a policy, π, is the expected return when starting in s and following the π thereafter:

$$v_\pi(s) \doteq \mathbb{E}[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s], s \in S \tag{2.1}$$

The action-value function represents the value of taking action a in state s under π:

$$q_\pi(s,a) \doteq \mathbb{E}[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a] \tag{2.2}$$

The value functions v_π and q_π can be estimated from observation. If an agent follows policy π and computes the average of actual returns following each encountered state, this average will converge to the state's value, $v_\pi(s)$, as the frequency of that state's occurrence approaches infinity. Similarly, by maintaining separate averages for each action taken in each state, these averages will converge to the corresponding action values, $q_\pi(s,a)$. Such estimation methods are referred to as Monte Carlo methods, as they involve averaging over numerous random samples of actual returns [49]. In scenarios with large number of states, the agent can alternatively represent v_π and q_π as parameterized functions and adjust these parameters to align with the observed returns.

The value function satisfies recursive relationship to describe the value of a state, s, and its possible successor states, s'. When Eq.2.1 is decomposed into the immediate reward and the discounted value of its successor states, the Bellman equation is obtained, averaging over all the possibilities and weighting each by its probability of occurring [49].

These mechanisms collectively enable behavior control and knowledge updates, which are the distinctive characteristics of RL. This framework is conceptually intuitive, as it parallels human learning process. The following examples serve to further showcase implementation of RL in the field of structural design.

EXAMPLE 1: EXPERTISE, PLAYFULNESS, ANALOGICAL REASONING (MIRRA, 2023)

Mirra's work explores playfulness in the autonomous development of design strategies through free exploration. In this approach, agents were not informed about precedents but instead learned to design through trial and error [26]. The agent interacted with the design game using a Deep Q-Network, trained to approximate the optimal policy determined by selecting the action with the highest Q-value for each state.

[26] Mirra. 2023. *Expertise, playfulness, analogical reasoning.*

The environment for this experiment was a virtual drawing board, where the agent's actions corresponded to placing structural nodes on a 32x32 canvas. Each pixel on the canvas represented a discrete point on a 2D grid, allowing structural nodes to be placed and connected to form a 2D frame structure. Initially, the canvas contained two support nodes and one obstacle. The agent's cursor started at the first support node in the bottom-left corner, and at each time step, it moved by a single pixel, progressively extending the frame.

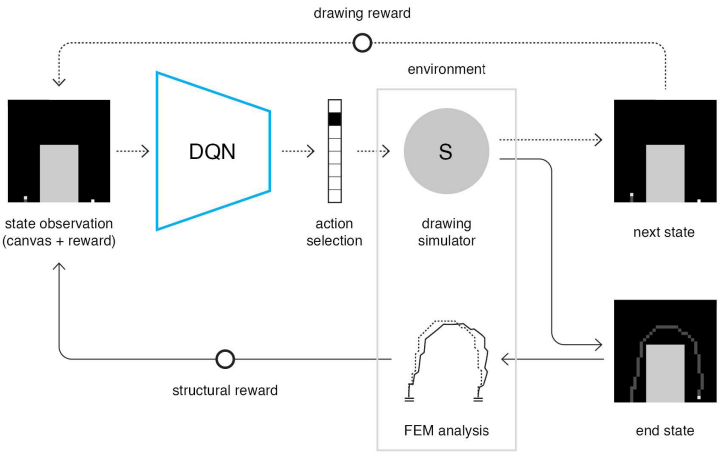


Figure 2.19: Interaction of the DQN with the environment components to collect design experiences in the form of state transitions and rewards (Mirra, 2023)

To refine the model, the environment incorporated dynamic elements, including agent laziness, collisions, and task completion. Small to medium negative rewards were assigned for each action or when the cursor attempted to place a node on a occupied position, encouraging exploration of new regions of the canvas. Large negative rewards were given for collisions, either with the obstacle or the canvas edges. A positive reward was awarded when the agent successfully reached the second support node. Crucially, structural rewards were only allocated upon reaching this node, incentivizing the completion of meaningful structural connections. Each sequence of state transitions resulting from the agent's actions was defined as a design episode. These episodes terminated if the cursor collided with an edge or obstacle, reached the second support node, or exceeded 300 actions. At the end of each design episode, the environment reset, clearing the canvas, repositioning the cursor, and generating a new obstacle defined by two opposite corners of a rectangle. This setup increased the complexity of the task, exposing the agent to varying boundary conditions and encouraging the development of more adaptable and resilient design strategies.

Mirra's analysis highlights how the environment dynamics prompted the agent to naively respond to the design task while also driving the convergence of the training process. Notably, the definition of the reward function had a profound influence on the policy learned by the agent. This conditioned the agent's exploration, rather than fostering genuine exploration, underscoring the challenges of designing reward systems in reinforcement learning for creative tasks.

EXAMPLE 2: ENHANCING INTERACTIVITY IN STRUCTURAL OPTIMIZATION THROUGH RL (MIRRA AND PUGNALE, 2023)

In this second example, the workflow comprises: (1) a 3D modeling environment, (2) a conversion process for constructing a structural model, and (3) a FEM solver for structural performance analysis. The agent captures the 3D environment's state as a 2D depth map, which encodes the height of a complex surface, decoded as a 8x8 matrix of values corresponding to the z-coordinates of equally spaced control points used to construct a NURBS surface.

A different RL approach, policy gradients, was employed to handle the complexity of the action space. Unlike the first example, the agent performs a sequence of sub-actions characterized by a tuple of three elements: first, a vector point P is selected from the 8x8 matrix of control points; second, the agent determines the direction of translation d (either -1 or 1); and third, it specifies a vector length from an array of 10 discrete values, defining the extent of movement along the z-axis.

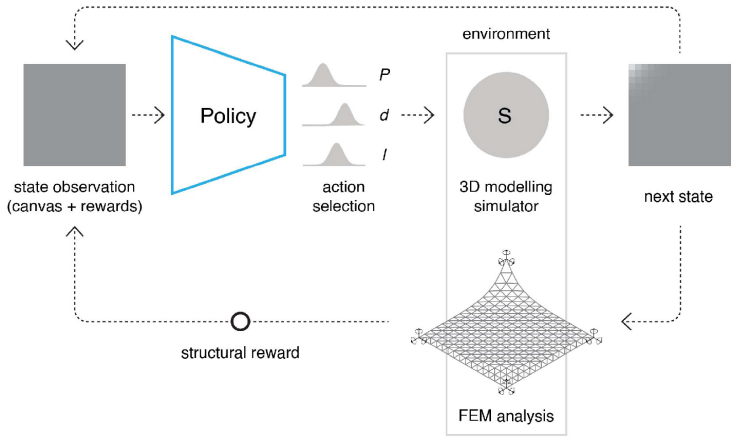


Figure 2.20: Interaction of the PPO with a 3D modeling simulator (Mirra and Pugnale, 2023).

The structural model for FEM analysis was constructed from the updated observations of the canvas. All mesh faces were converted into 5 cm thick concrete shell elements, with the corners of the mesh used as fixed support nodes. A uniformly distributed gravity load of $2kN/m^2$ was applied, and the structural reward was derived from the maximum displacement of the concrete shell.

The environment offered two distinct design tasks. When the canvas grid was initialized as a flat shell, the task became entirely exploratory, as the agent progressively deformed the shell to produce a structurally sound design. Alternatively, if the canvas grid was initialized with non-zero values, the agent refined a predefined shell design to enhance its structural performance. This second approach required the designer to create the input design using the same types of actions available to the agent, which could be limiting. To accommodate the increased complexity of agent actions in this scenario, the maximum number of iterations was kept between 10 and 20. Simplifying the action space reduced the need for extensive specification of environment dynamics and ensured the design strategy learned by the agent was not overly conditioned by the setup.

At the end of each episode, structural rewards were redistributed into one-step reward increments. This redistribution allowed the agent to observe the immediate impact of its actions, measuring how much a single design move contributed to reducing the maximum displacement of the shell. in the context of the shell design task, this approach provided the agent with a clearer understanding of the relationship between specific design adjustments and structural performance outcomes.

3

Markovian Decision Framework

This chapter examines the key concepts and computational characteristics of the methods used to facilitate exploration of the topological design space of quad meshes. It begins by outlining the specifics of the quad-mesh grammar, followed by a discussion of a suitable RL algorithm for the study. To fully harness the learning capabilities of the selected RL agent, the chapter addresses appropriate knowledge representation strategies and establishes a preliminary generative model within a Markovian framework. The components of the MDP are designated with particular emphasis on critical features, such as topological and string properties, to intentionally reduce the complexity of the design problem by disregarding specific loading and boundary conditions. The following chapter will evaluate the extent to which the identified features are sufficient for the RL agent to adopt policies that generate diverse mesh layouts through innovative string-action sequences, as illustrated below.

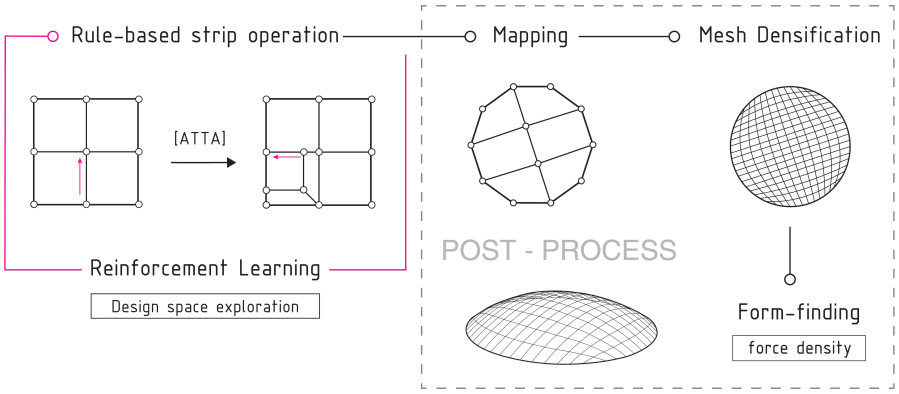


Figure 3.1: General workflow depiction.

3.1 METHODS

3.1.1 QUAD-MESH GRAMMAR

The term quad-mesh grammar originates from one of Oval's researched strategies, which decouples the topology and geometry of a design by translating topological modification and movement operations into a set of rules represented by alphabetic letters. This approach, rooted in the bottom-up philosophy, leverages the potential to generate a diverse array of mesh topologies through the infinite combination of these letters.

The rules are applied at a coarse resolution on quad-meshes before the sequence of densification and geometrical processing begins. Its mechanism operates on the strip structure which is defined by the relationship between pairs of opposite edges across the quad-faces as shown in Fig. 3.2. This approach offers an indirect way to control singularities by altering the connectivity, and "is tailored for the exploration of a comprehensive design space constrained to quad-meshes" [37].

[37] Oval et al. 2023. A vector encoding for topology finding of structured quad-based patterns for surface structures.

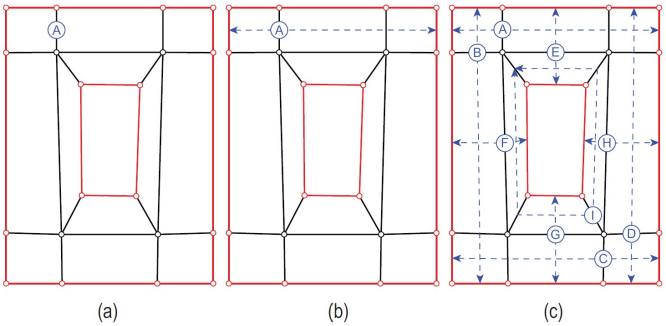


Figure 3.2: Strip structure (Oval, 2019).

A marker is constrained by the initial topology of the input coarse quad-mesh with specified orientations. Each edge connecting two vertices produces a pair of half-edges, oriented in opposite directions, as indicated by the blue arrows in Fig. 3.3. These half-edges are also associated with an adjacent face on their left side, which establishes directionality for any sequences of movement operations. To perform string-based operations, a marker is first placed onto a vertex; it can then perform two types of movement and two types of modification commands:

Turn (T) – moves the marker to the next edge on the leftward face with an anticlockwise rotation, as depicted by the blue arrows.

Pivot (P) – rotates the marker clockwise to the next edge on the rightward face, as indicated by the orange arrows.

Add (A) – adds a strip along a sequence of edges.

Delete (D) – removes the strip perpendicular to the marker's position.

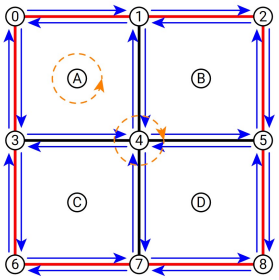


Figure 3.3: Half-edge mesh data structure with edge and node orientation (Oval, 2019).

To illustrate using Fig. 3.3, performing two 'T' commands from point 6, followed by a 'P' command and two additional 'T' commands, moves the marker to points 7, 4, 1, and finally to 0. These two movement operations enable the selection of specific mesh elements for strip modification. To successfully add a strip, in terms of computational operations, two 'A' commands must be implemented. The first toggles the collection of a polyedge, which refers to a sequence of edges. After successive combinations of 'T' and 'P' the second 'A' command ends the collection and inserts a corresponding strip along its traveled path. Additional parameters (*) and (○) allow for the addition of strips with poles and closed strips, respectively.

In Fig. 3.4, strip addition along the polyedge A-B-C introduces pairs of new vertices, prime and double prime, which replace and are positioned on the left and right sides of the original vertex, to establish a new quad-strip. The strip addition also introduces singularities at non-boundary nodes B' and B" with corresponding valencies of five and three edges. Overlapping polyedges can also be modified as demonstrated in Fig. 3.5.

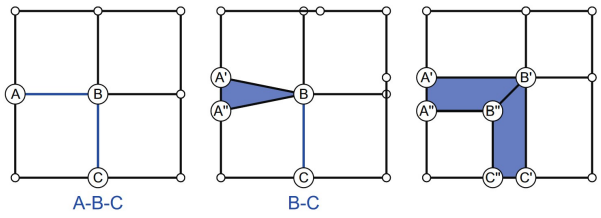


Figure 3.4: Strip addition introducing singularities (Oval, 2019).

Two additional computational rules are mentioned here. Firstly, after the application of each rule, the strip data is updated, changing the labels of the vertices and faces to accommodate the topological modifications; however, the labels of the strips are preserved to keep strip attributes and enable the combination of multiple rules, for instance. Secondly, before completing the deletion maneuver, the marker is adjusted using pivot operations until it aligns with the polyedge corresponding to the strip that will be collapsed. In Fig. 3.6, a deletion operation is applied during polyedge collection before the second add operation. The collected polyedge is updated, replacing and removing redundant vertices. The edge [0,1] is collected with the first 'A' command. When the subsequent strip outlined by nodes 1,2,5,4 is deleted, the old polyedge is updated to [0,6]. Finally, when the second addition is implemented, a new strip, outlined by 8,9,11,10, is introduced.

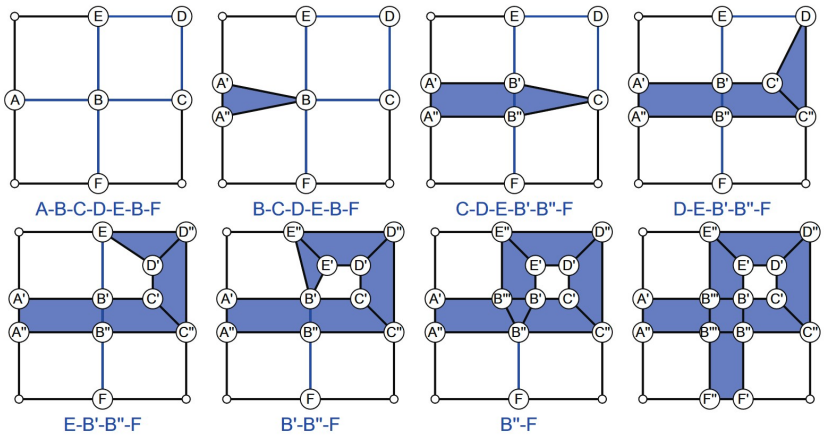


Figure 3.5: Self-overlapping strip addition (Oval, 2019).

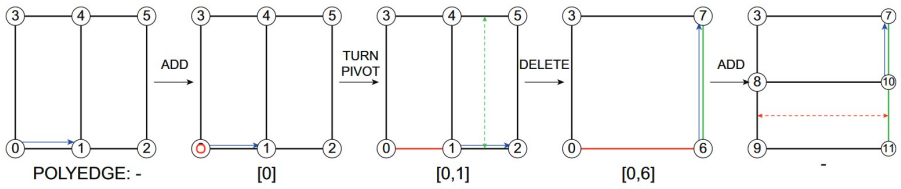


Figure 3.6: Update of vertices labeling when executing strip deletion during strip addition maneuver (Oval, 2019).

COMPAS_QUAD

The algorithms for the briefly introduced quad-mesh grammar have been developed as part of COMPAS, an open-source, python-based computational framework, developed by BRG at ETH Zurich. It is a robust, fully-agnostic system that integrates computer science libraries and packages into architectural workflows. Its core data structure enables engineering calculations and fosters collaboration across diverse academic backgrounds, programming skills, and computational expertise. COMPAS facilitates cross-platform use and integration with external tools, enhancing cooperation. This capability is exemplified in projects such as the Striatus Bridge and Phoenix Bridge, where COMPAS was used to link separate computational pipelines from ZHACODE and incremental3D [38].

[38] Parametric-Architecture. 2024. 66 - Philippe Block - Computational Design, AI, Compas, Digital Fabrication, 3D-Printing, BRG.

3.1.2 RL ALGORITHMS

OPTIMAL POLICIES AND VALUE FUNCTIONS

The Markov decision process (MDP) provides a mathematical model for decision making:

$$M = (S, A, R, P, \rho_0, \gamma) \tag{3.1}$$

where: S = states
A = actions
R = rewards
P = state-transition probability
ρ_0 = initial state distribution
γ = discount rate

An MDP is defined by a finite set of actions, states, and rewards, where the dynamics depend on the sequence of previous interactions. The environment's dynamics are represented by the joint state-transition and reward probability:

$$p(s', r|s, a) \doteq Pr[S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a] \tag{3.2}$$

From this, the marginal state-transition probability (s'|s,a) can be obtained by summing over all possible rewards $r \in R$. The expected reward for a state-action pair is given by:

$$r(s, a) \doteq \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] \tag{3.3}$$

The Bellman equation for the value function, v_π, is expressed as:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}[G_t|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{3.4}$$

Similarly, the Bellman equation for the action-value function, q_π, is given by:

$$\begin{aligned} q_\pi(s,a) &\doteq \mathbb{E}[G_t | S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r|s, a)[r + \gamma q_\pi(s', a')] \end{aligned} \tag{3.5}$$

For finite MDPs, the Bellman equations are extended to derive an optimal policy that maximizes the expected cumulative return:

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \tag{3.6}$$

$$q_*(s,a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')] \tag{3.7}$$

ON-/OFF-POLICY LEARNING

On-policy algorithms refer to approaches in which the policy used to interact with the environment is the same policy that the agent is learning from. In contrast, off-policy methods involve learning about an optimal policy or a different target policy while using a separate behavior policy to explore the environment in a suboptimal manner during training. The latter offers greater flexibility in exploration, as they allow the agent to learn from data generated by previously stored experiences enabling efficient use of past interactions.

MODEL-BASED VS. MODEL-FREE

In the classification of RL, selecting the most suitable algorithm often depends on the learning objective—whether it involves evaluating the policy, value function, action-value function, or environment model. The flow chart below, adapted from OpenAI, presents a non-exhaustive taxonomy of the RL methods based on broad categorization, with the first branching point being the type of model employed [33].

[33] OpenAI. 2018. Spinning Up: A Taxonomy of RL Algorithms.

An environment model is a function that predicts all possible state transitions and rewards, allowing the agent to plan ahead by evaluating different trajectories and selecting the best course of action. A notable example is AlphaZero, which uses a model of the game board to significantly improve sample efficiency over other methods that do not rely on a model [44]. However, ground-truth models of the environment are often unavailable, requiring the agent to learn from experience. This approach can introduce challenges, such as learned biases that may cause the agent to perform poorly in real-world scenarios due to over fitting.

[44] Silver. 2016. Mastering the game of Go with deep neural networks and tree search.

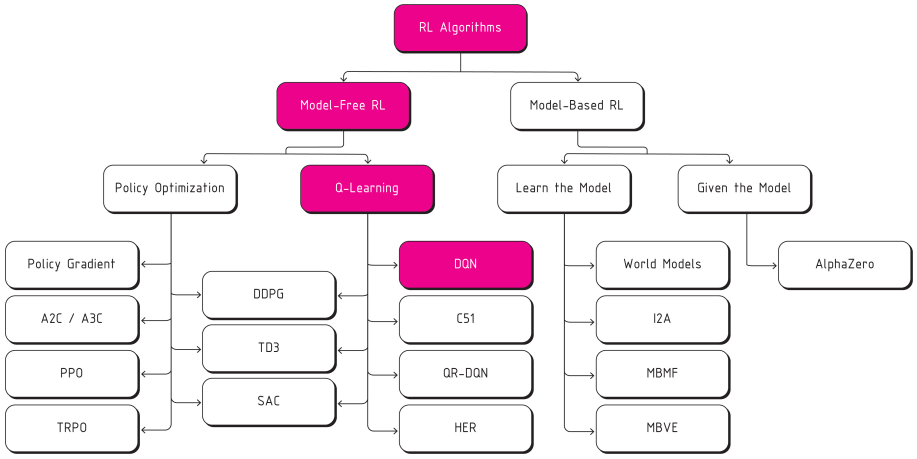


Figure 3.7: RL Umbrella Adapted from (OpenAI, 2018).

Model-free methods are more common because they are easier to implement and tune. These approaches do not require pre-defined state-transition probability functions and instead learn policies directly through interactions with the environment. While model distinctions are crucial and model-based methods often lead to significant performance improvements, their adoption is limited by the time and computational effort needed to define and utilize innumerable state transitions.

POLICY OPTIMIZATION

Policy optimization is a class of techniques that directly adjusts the policy by explicitly representing it as a parameterized function, $\pi_\theta(a|s)$. The parameters, θ, are adjusted to improve the performance objective, $J(\pi_\theta)$, typically using stochastic gradient ascent (SGA). This method is generally performed on-policy, and an approximator is learned to guide the policy updates. Algorithms such as Actor-to-Critic (A2C) use SGA to maximize performance, while Proximal Policy Optimization (PPO) stabilizes learning by optimizing a surrogate objective function [42]. This approach is particularly effective for high-dimensional and continuous action spaces, making it suitable for complex environments such as advanced robotics and sophisticated games, including Dota 2 and Super Smash Bros [12][34].

[42] Schulman et al. 2017. Proximal Policy Optimization Algorithms.

[12] Firoiu et al. 2017. Beating the World's Best at Super Smash Bros. with Deep Reinforcement Learning.

[34] OpenAI et al. 2019. Dota 2 with Large Scale Deep Reinforcement Learning.

Q-LEARNING

Methods known as Q-learning refer to RL algorithms that focus on learning the optimal action-value functions, based on the Bellman equations, to derive the policy. The agent updates its estimates of q-values, and this optimization

is almost always performed off-policy. Notable algorithms include Deep Q-Networks (DQN), which use NNs to approximate the q-values and are known for their sample efficiency in comparison to policy optimization approaches. Q-learning methods excel in scenarios with discrete action spaces, making them ideal for small- to medium-sized problems such as grid-world environments or Atari games [28]. The distinction between the two methods lie in their optimization targets. In Q-learning, the goal is indirectly optimized, which increases the potential for failure modes but offers significant gains in sample efficiency. Additionally, a range of algorithms exists that interpolate between the two approaches, balancing the trade-off between principled logic and sample efficiency.

[28] Mnih et al. 2013. Playing Atari with Deep Reinforcement Learning.

Q-NETWORK LOSS

The key difference between the Q-learning and DQN lies in their representation and update mechanism for the values of state-action pairs. Q-learning is a tabular method, where the q-values (Eq. 3.7) are stored in a table, while DQN extends this framework by incorporating NNs to approximate the q-values, enabling the algorithm to handle large or continuous state spaces. In DQN, two NNs are used: the online Q-network and the target Q-network.

The online Q-network (also called the policy network) outputs predicted values for state-action pairs based on parameterized weights, θ. Given a current state, s, and action, a, the network computes the q-value for all possible actions as follows:

$$Q_{\text{pred}}(s, a; \theta) = Q_{\text{online}}(s, a; \theta) \tag{3.8}$$

Its output reflects the agent's expectation of the rewards available when executing the next action in a sequence. The target Q-network has the same architecture but uses a separate set of weights, θ^{-}. This target network provides stable reference values for the online Q-network. The target q-values are calculated using the immediate reward, r, and the discounted maximum future reward for the next state s', using the target network, by the following equation:

$$Q_{\text{target}}(s, a) = r + \gamma \max Q_{\text{target}}(s', a'; \theta^{-}) \tag{3.9}$$

The difference between the two networks lies in the update frequency of the parameterized weights. While the online network updates frequently with new incoming experiences, the target network's updates are deliberately delayed to provide stability during training. This delay ensures that the target values remain consistent as a reliable reference point, preventing the q-values from shifting too rapidly. In the DQN algorithm, the process alternates between predicting q-values and updating them, using the target network to compute stable target values.

During the training process, the learning occurs when the weights of the online networks are updated. This is done by minimizing the difference between the predicted and target values, also known as the temporal difference (TD) error or q-loss. The update rule improves prediction accuracy over time and is governed by the following equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(Q_{\text{target}}(s,a) - Q(s,a)) \tag{3.10}$$

where: α = learning rate
$Q_{\text{target}}(s,a) - Q(s,a)$ = TD error

The q-loss, is typically derived using mean squared error (MSE). However, in stable-baselines3, Huber loss is used instead, as it is less sensitive to outliers, which helps stabilize the training process. The Huber loss is defined as follows:

$$L_\delta(x) = \begin{cases} \frac{1}{2}x^2 & \text{for } |x| \leq \delta \\ \delta(|x| - \frac{1}{2}\delta) & \text{for } |x| > \delta \end{cases} \tag{3.11}$$

The Huber loss for a batch is given by:

$$\text{Loss} = \frac{1}{N}\sum_{i=1}^{N} L_\delta(Q_{\text{pred}}(s_i,a_i) - Q_{\text{target}}(s_i,a_i)) \tag{3.12}$$

where: x = TD error
δ = Threshold between quadratic and linear loss transition
N = Batch size sampled from replay buffer

Huber loss behaves similarly to MSE for small errors, penalizing them quadratically. For larger errors, its piecewise form transitions to a linear function, reducing sensitivity to outliers. This helps prevent large TD errors from disproportionately affecting the gradient updates, thereby supporting more stable training. Once the loss is computed, the network weights are updated using back propagation over batches of experience stored in the replay buffer.

LIMITATIONS

The strength of RL algorithms lies in their flexibility to adapt to diverse scenarios. For example, multiple agents can be incorporated by assigning numerous reward signals to the RL model, enabling exploration of spatial configurations in urban environments [51]. However, developing the RL program can be time-intensive, as real-world problems are often challenging to replicate virtually due to their complex hierarchical relationships. Furthermore, simulation environments come

[51] Veloso and Krishnamurti. 2020. An Academy of Spatial Agents - Generating spatial configurations with deep reinforcement learning.

with high research costs and lengthy iteration times, which can hinder the practical implementation of RL [54]. Even when an agent has access to a complete and accurate environmental model, it may not fully leverage this advantage due to limitations in memory and computation per time step [49].

[54] Zhu et al. 2021. An Overview of the Action Space for Deep Reinforcement Learning.

[49] Sutton and Barto. 2018. *Reinforcement learning: an introduction, Second edition.*

GYMNASIUM AND STABLE-BASELINES3

The primary Python libraries utilized for the RL algorithm are OpenAI's Gymnasium and stable-baselines3. A custom training environment will be developed in Gymnasium, which provides a flexible API for single-agent RL environments, supporting key functions such as make, reset, step, and render. At the core of Gymnasium is the Env class, a Python class representing an MDP. Although pre-built environments such as Classic Control, Box2D, MuJoCo, and Atari exist, a custom environment will be constructed for this project. Once initialized, the environment is reset to produce the initial observation, which, can be either predefined or generated with a randomized seed. The agent then takes actions in the environment using the step function, causing the environment to update, providing new feedback in terms of new observations and rewards at each time-step. The environment keeps running until reaching a terminal state which is defined, either, as a specified step count or completion of the design task. When an episode ends, whether through task completion or truncation, the environment is reset to prepare for the next run as shown in the following Fig. 3.8.

Stable-baselines3 complements Gymnasium by providing a collection of modular components designed to integrate with the Gymnasium API. It offers an open-source implementation of seven commonly used model-free deep RL algorithms in PyTorch, simplifying the process of training, saving, loading, and inferring actions from learned policies. Additionally, Weights and Biases (WandB) includes scripts to log the agent's training results, perform sweeps, and visualize outcomes. Together, these tools streamline the development workflow, from training RL agents to efficiently managing and analyzing their performance.

3.2 SYNTHESIS

The selection of RL algorithm for this task is model-free, as the state-transition probabilities are unknown beforehand. A value-based learning approach, such as DQN, is chosen to examine the suitability of off-policy methods in deriving unexpected string-action maneuvers, with an emphasis on exploring design spaces. This decision is further supported by the abundance of literature available on DQN implementation.

To combine the available tools, the quad-mesh grammar must be successfully translated into the various components of RL. The iterative nature of string-based

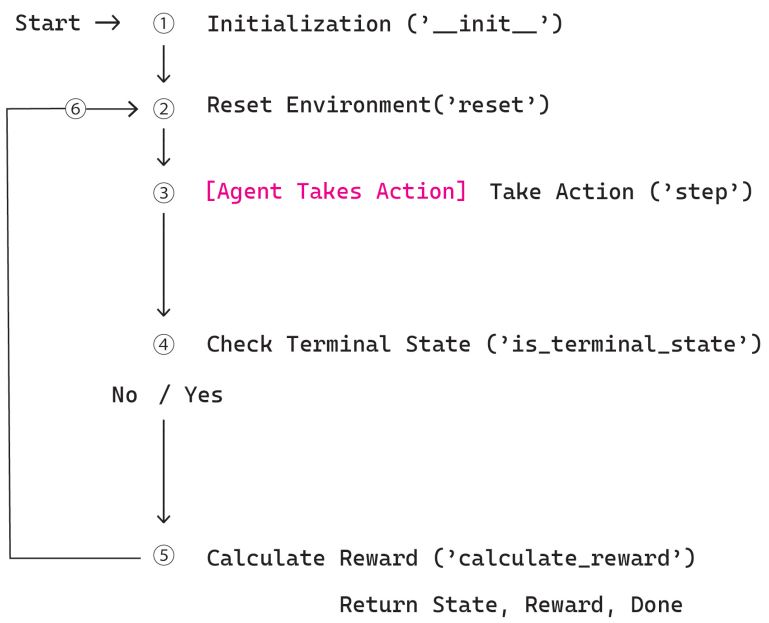


Figure 3.8: Gymnasium Environment Workflow.

operations makes RL a more intuitive selection over other AI approaches, such as genetic algorithms; however, challenges remain in translating key concepts before initiating the learning process for a policy that constructs diverse and efficient meshes.

The following sections outline the workflow for adapting the quad-mesh grammar to an MDP framework, ensuring the design problem of this thesis is appropriately programmed for the agent to learn effective strategies. The feasibility of the thesis is assessed by verifying compatibility between the selected Python libraries. A simplified methodology, which reduces model complexity, is implemented by initially removing the focus from geometric and structural variables, allowing the primary objective to be the successful integration of compas_quad into the RL process.

3.2.1 SIMPLIFIED MODEL

An additional goal of the preliminary model is to determine whether the RL agent can effectively explore the design space by using compas_quad to modify an initial simple mesh topology, s_0, in order to reach a specified terminal state, s_T, as shown in Fig. 3.9. The terminal state is characterized by both a different mesh topology and its corresponding action string, which serves as the ground truth during the training process. A custom mesh environment has been created with a focus on fine-tuning the reward function and hyperparameters to guide

the RL agent toward the desired outcome. The output of the model can be post processed with a circular boundary in equilibrium only under homogeneous internal forces for visual purposes.

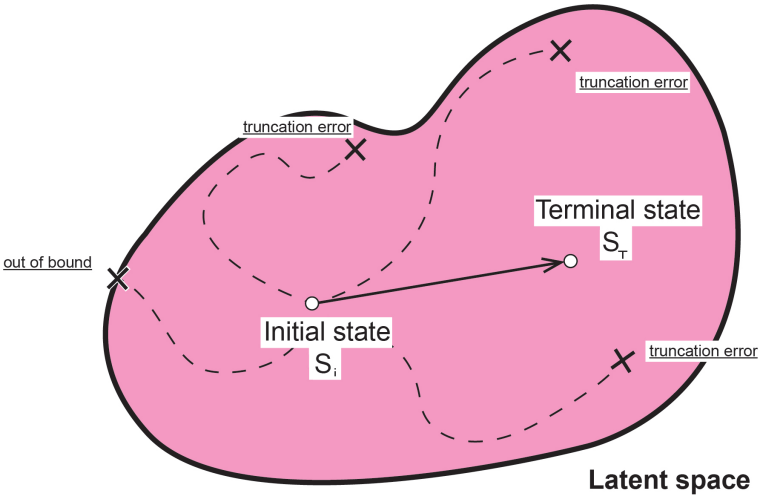


Figure 3.9: Latent space representation.

ACTION SPACE, $A(t)$

In this model, discrete action corresponds to the specific letters used in the quad-mesh grammar. Each time step represents a successive stage of decision-making event, where the agent executes a string-based operation. For simplicity, only the 'A', 'T', and 'P' maneuvers are implemented, while the deletion action is omitted.

Although the string encoding can generate an infinite combination of movement and editing operations, the compas_quad algorithm itself cannot process arbitrary heuristic combinations due to the computational rules discussed in Section 3.1.1. To address this, truncation conditions for the learning process have been defined to notify the agent that action sequences prone to failure are inconsequential. With this approach, the agent is implicitly discouraged from attempting such action combinations. For instance, this occurs for addition sequences that conclude without any movement commands, such as 'AA', or during strip addition with purely pivot commands, such as 'APA' and 'APPPA'. Both produce a KeyError when tracking the marker position, returning 'None' instead of an integer that should represent an existing node in the matrix of the mesh.

It is noted here that extended operations during strip addition will be avoided to prevent exponential growth in computational load. The extent of exploration

will be controlled by specifying the number of design steps that can be taken during each design episode.

STATE SPACE, $S(t)$

The state represents any relevant information that aids the RL agent in learning the decision-making process. The agent's performance is dictated by its observations; without this input, the agent would act blindly, unable to adapt to or influence the environment. Hence, an appropriately structured observation space is critical for reaching the terminal state or any goal in its designated task. Although the mesh is constructed within a Cartesian coordinate system, in the simplified model, details regarding the coordinates of the vertices, edge indices, edge attributes, and face data required to fully describe the mesh have not been introduced. Instead, two key parameters, organized in dictionary format, define the observation space: the histogram of all nodes and the position of the marker through which the agent can navigate and modify the topology of the mesh.

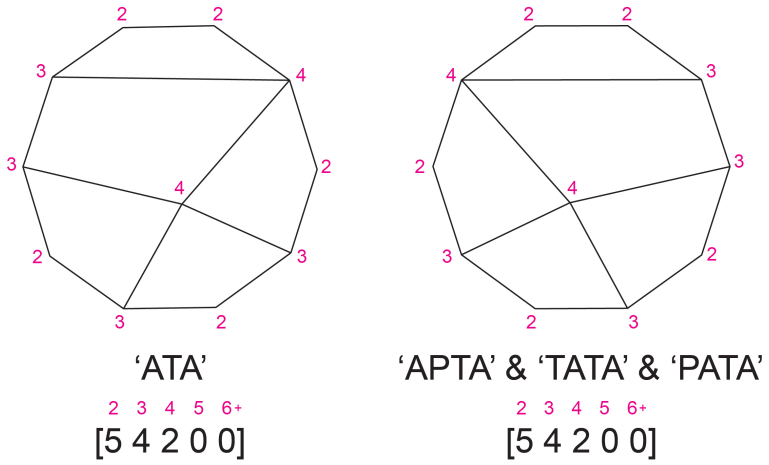


Figure 3.10: Singularity histogram.

At each time step, the agent receives an updated set of topological and positional data based on the string action it performs. The topological data includes a histogram of all the vertices in the mesh, classified by the number of edges they are connected to. This allows for the identification and comparison of both regular nodes and singularities between the current and terminal mesh layout. When the histogram of the current mesh topology matches that of the terminal configuration, the episode is terminated as the specified end goal has been reached. If the histograms do not match, the agent continues to execute different actions until the maximum number of design steps specified for that run is reached. Fig. 3.10 also illustrates multiple action-string sequences that pro-

duce the same histogram and mesh layout, highlighting the lack of isomorphism between strings and meshes. This non-isomorphic relationship can influence the RL agent's ability to interpolate and effectively explore the design space of strings and meshes.

The position of the marker, hereafter termed "lizard" based on the original implementation of compas_quad, relays the associations between the design spaces of the grammar and the mesh. The position information is structured as the placement of the head, body, and tail of the lizard, which is used to identify the trajectory along which the marker can operate when executing the action commands, as shown in Fig. 3.11. For example, the 'T' and 'P' commands change the referenced node locations, as opposed to no change when activating the sequence of strip addition through the 'A' command. These two observation parameters enrich the observation space from which the RL agent must learn.

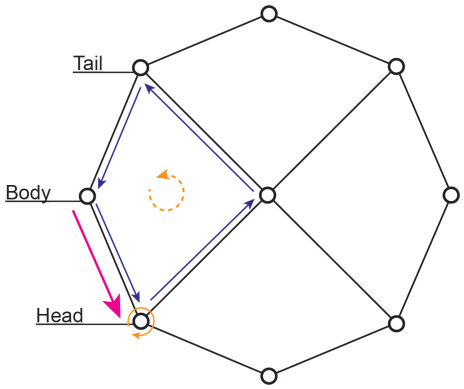


Figure 3.11: Lizard position.

REWARD, $R(t)$

For the quad-mesh environment, the reward function has been designed to incentivize exploration and prevent stagnation through a combination of shaped and sparse rewards based on the parameters used to describe the observation space. Three distinct reward mechanisms will be independently investigated to determine the most suitable methodology for guiding the RL agent to reach the terminal state. These mechanisms include different definitions of a distance metric that illustrate the closeness or proximity between the current iteration and final mesh layouts in various design spaces, namely the histogram distance, string distance, and mesh distance, illustrated in Fig. 3.12.

As previously mentioned, the histogram distance compares the current and terminal mesh layouts relying on the topological information. This distance metric can return an aggregated value or probability distribution describing the

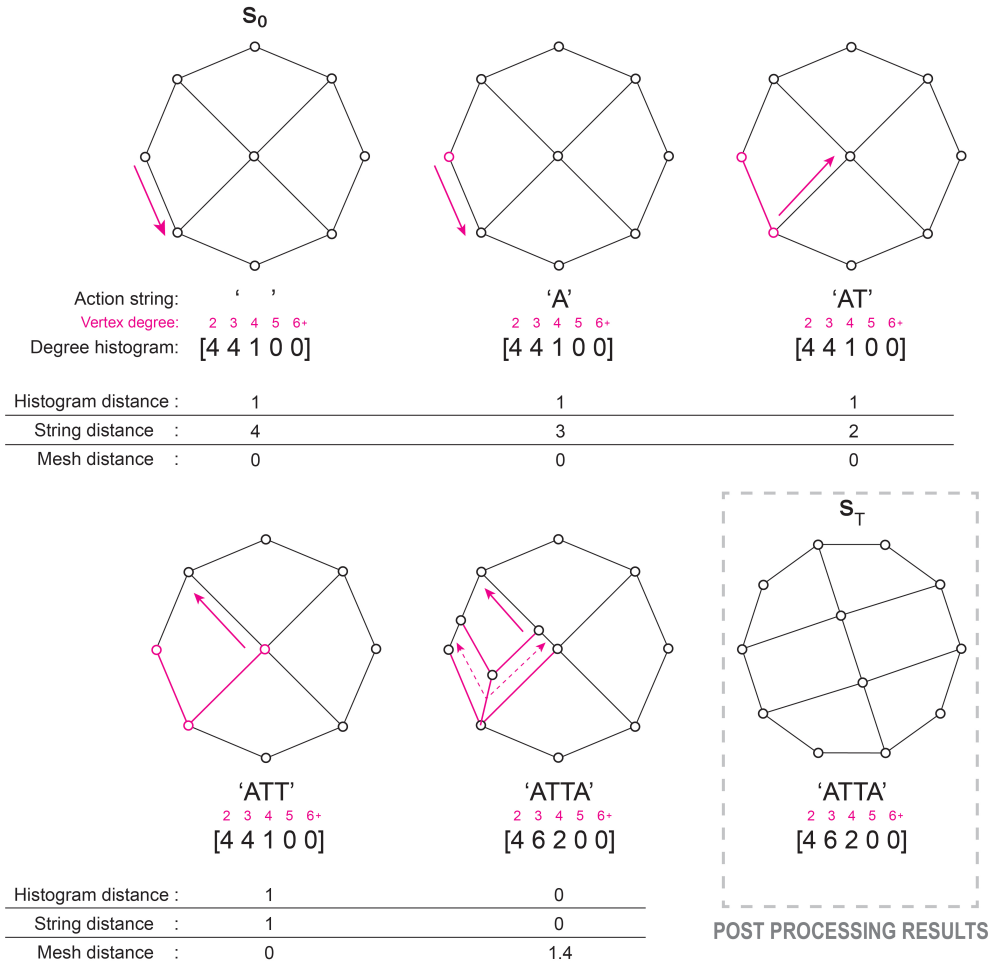


Figure 3.12: Example depiction of A to B operation.

likelihood of singularity degrees between two states. Hence, the primary driver for the agent in exploring different actions will be to reduce the distance between the topological states of the initial and terminal conditions. Regardless of the selected distance metrics, the histogram information used in the observation space will be employed to allocate a large positive sparse reward when the histogram of all singularities in the current and terminal states match, indicating successful completion. For the simplified model, the difference calculated between two histograms will be investigated using mean-squared error (MSE) . This analysis will be conducted only for the the histogram distance because the values returned by the other two methods are integers, which do not provide a granular basis for comparison.

The two other measures of distances are associated with the terms "genotype" and "phenotype," which Oval uses as synonyms to describe the string encoding and mesh layout, respectively. These terms are used to discuss the lack of string-to-mesh isomorphism in the computational methodology of the quad-mesh grammar, a concept briefly mentioned earlier. The infinite nature of recombination of alphabetic characters results in multiple distinct combinations of movement and editing operations that can potentially result in the same quad-mesh topology. This redundancy implies the lack of injection from the genotype to phenotype, leading to "[a lack of] bijection… between the space of strings and the space of quad mesh topologies" [36]. The string distance is a distance parameter based on Levenshtein's method for calculating the difference between two string sequences, measured by the minimum number of single-character edits required to transform one string into another. This metric, widely used in computer science and information theory (e.g. text comparison tasks and DNA sequence analysis), effectively describes the genotype attribute of the mesh. For example, between the strings, 'kitten' and 'sitting', the distance is three, as 'kitten' can be transformed into 'sitting' by substituting 'k' with 's', replacing 'e' with 'i', and appending 'g'.

[36] Oval. 2019. *Topology Finding of Patterns for Structural Design.*

Finally, the mesh distance measures the difference in the number of strips or faces appended as a result of topology modification from the string actions. While the number of additional faces embedded to the mesh layout can depend on the resolution of the mesh input, each strip addition maneuver incorporates an additional strip between the starting and ending points of the lizard, thus relaying an additional layer of information that is neither at the space of the strings, nor completely overlapping with the information gained through topological data alone. Hence, the term mesh distance, as it provides feedback on the mesh-strip connectivity during intervention.

Negative shaped rewards are also applied at every time step to encourage exploration of the design space. Without these penalties, the agent can become idle, accumulating no rewards and leading to stagnation. The shaped rewards prevent the RL agent from exploiting sequences of inconsequential actions and ensure that both modification and movement commands are actively pursued throughout the learning process. Additionally, at the end of each design episode, the topological histogram of the mesh structure will be compared to the terminal mesh configuration to determine whether the agent has successfully implemented any modifications. Any original sequence that introduce deviations from the original histogram will be rewarded, while identical configurations are penalized using a negative sparse reward to notify that the agent has not been successful at its task. The application of the three distance metrics in Fig. 3.12 and their specific values are explained in the next chapter.

IMPLEMENTATION

The DQN of the simplified model will be constructed using a fully connected NN with two layers consisting of 64 neurons each, and a ReLU activation function to determine the q-value for the three discrete actions ('A', 'T', P'). The stable-baselines3 API stores the replay data of the RL agent using a replay buffer, in which the estimates are derived following a greedy policy. Stable-baselines3 provides a comprehensive list of variables that can be adjusted to modify the DQN parameters. For the numerical experiments, fine-tuning hyperparameters is essential to optimize the model for the respective reward function set up. A set of curated variables, beyond the elements mentioned in the previous section, will be adjusted to achieve optimal performance.

The following pseudo code illustrates the implementation of the training setup using stable-baselines3, which incorporates the custom quad-mesh environment created with the Gymnasium API. The source code can be accessed at: https://github.com/tohmakobayashi1016/RL-StringOp. For non-commercial uses only.

Algorithm 1 Training execution

Require: Initial/terminal coarse mesh, S_B, and maximum step count, n_{max}
Ensure: Quad-mesh data structure and lizard position

Initialize custom environment, $\mathcal{E}$, and step counter, n
Hyperparameters: DQN:(x,x,x...)
while Training not finished **do**
    Modify initial mesh copy, S_A
    if n < n_{max} **then**
        Perform mesh modification
        Update to current mesh
        truncated: $n \geq \mathrm{n}_{max}$
        terminated: $S_A = S_B$
    else if *ValueError* or *TypeError* **then** ▷ Robustness of the lizard
        truncated = True
    end if
    Calculate reward
    Reset $\mathcal{E}$
end while

4

Numerical Experiments

This chapter evaluates the performance of the constructed simplified model, as detailed in Section 3.2, by analyzing the influence of the chosen RL algorithm reward function configuration, and hyperparameter on overall model efficacy. The investigation begins with a set of simple initial and terminal states, examined through the sequential application of the quad-mesh grammar. This preliminary exploration is followed by a similar setup, where the defined terminal state is generalized to evaluate the RL agent's ability to handle complex string-action sequences of varying lengths. Finally, the outcomes of the study are extended by subjecting a mesh topology generated by the RL agent to a form-finding workflow, which reintroduces structural engineering parameters to demonstrate the potential of an extended, integrated workflow.

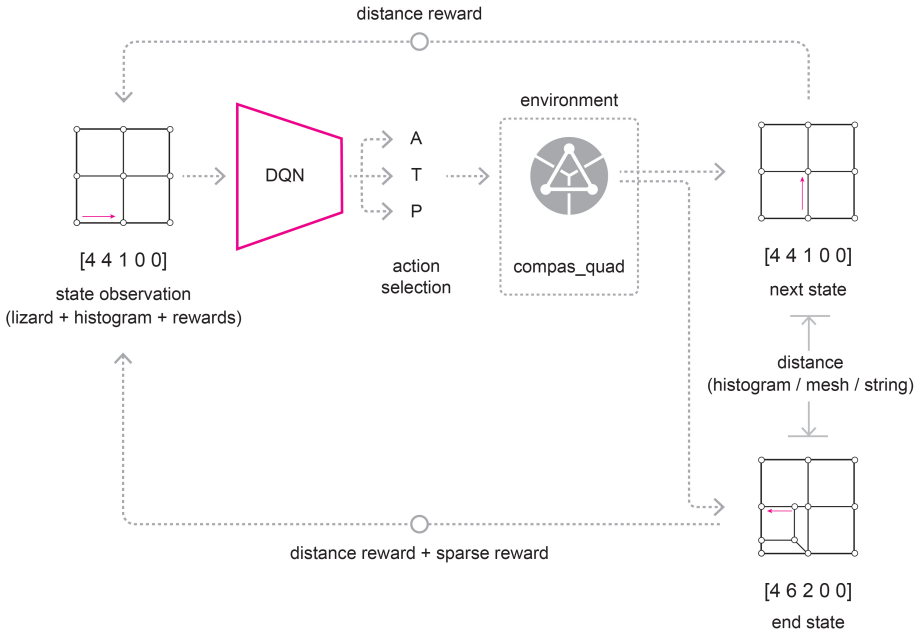


Figure 4.1: Interaction of the RL and environment components.

4.1 SEQUENTIAL EXPLORATION

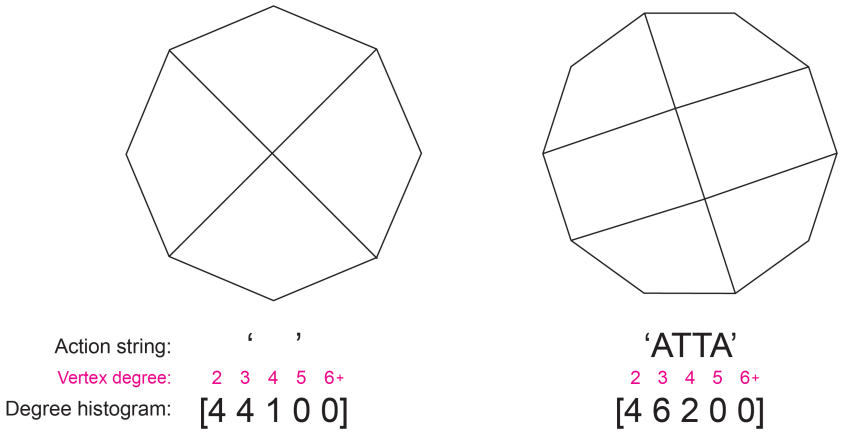


Figure 4.2: Initial and terminal states

The sequential exploration employs a straightforward four-step action sequence, ‘ATTA’, serving as the ground truth for the RL model. The objective of this initial investigation is to assess the RL agent’s proficiency in executing quad mesh grammar commands to achieve the target action sequence and its corresponding topological structure. This is done through an independent analysis of different reward function configurations, utilizing the DQN algorithm.

4.1.1 FORWARD & BACKWARD APPROACHES

The four step action sequence, ‘ATTA’, is segmented into four distinct steps, with performance assessed using both a forward and backward approach. In the forward approach, the initial state begins with an empty action string and a basic coarse quad mesh, characterized by an input mesh refinement level of two -indicating the subdivision degree of the boundary edge. The initial state is kept constant, and the target sequence is then progressively constructed and defined: ‘A’, ‘AT’, ‘ATT’, ‘ATTA’. At each stage, the RL model must execute the correct actions corresponding to the target action sequence and its associated topological histogram.

In the backward approach, the four step action sequence is deconstructed in reverse: ‘ATT’, AT’, ‘A’, ‘’. In this method, the goal is to reach the ground truth sequence in reverse order, with the terminal state kept constant while the initial environment state is modified. Consequently, the RL agent must develop an understanding that an addition action is required to achieve the target terminal state in the first deconstructed sequence. In the last step, the RL agent will start with an empty action string to reach the terminal state in both the forward and

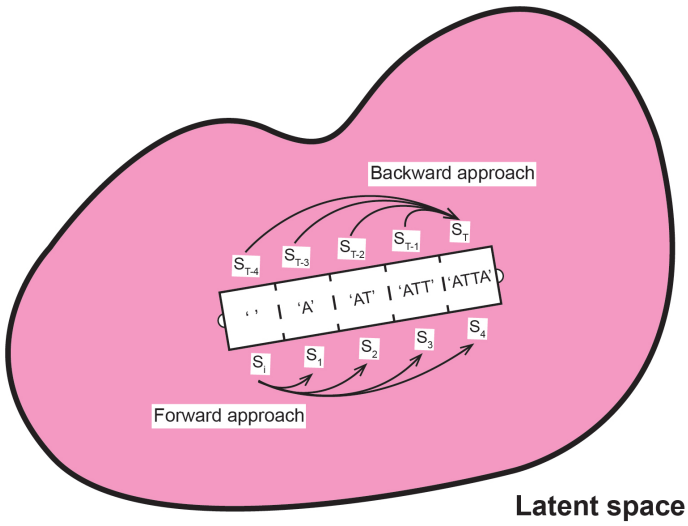


Figure 4.3: Sequential exploration between two specified states.

backward approaches to reach the four-step action sequence and the results will be used evaluate the model's adaptability in executing the correct actions to transition between any two specified states.

4.1.2 HYPERPARAMETER TUNING

Each training process involves executing 1,000 design episodes for each deconstructed step, which has been observed to allow the rewards of the simplified model to converge to a satisfactory level while maintaining reasonable computational efficiency. With each additional stage, the total number of design steps increases proportionally, as the RL agent must perform a greater number of actions within a single design episode. For hyperparameter tuning, 30 training batches will be evaluated using Weights and Biases, aiming to determine the optimal model configuration for each reward function setup at each stage. Given that four distinct stages are analyzed across three different reward function configurations in two directions, a total of 21 parameter sweeps will be conducted. The selected hyperparameters for each problem configuration and outputs from Weights and Biases are documented in the annex. For the DQN implementation, the sweep objective was to minimize the Q-loss using a Bayesian optimization approach.

The results presented are averaged over 10 runs, except for the one-step scenario, which is based on 5 runs due to its straightforward nature that does not necessitate complex decision-making skills. A Gaussian smoothing operation with a moderate window size of 30 points was applied to obtain a representative

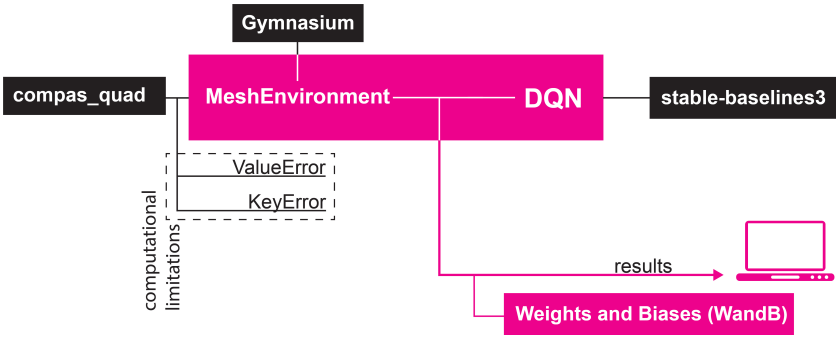


Figure 4.4: Python package dependencies

mean for the derivation of rewards and q-values in the figures and tables included in this chapter. For clarity purposes, all additional graphs have been moved to the annex.

4.1.3 ONE-STEP

The learning agent demonstrates a clear preference for selecting action 'A' over actions 'T' and 'P'. This pattern is consistent across both the forward and backward approaches, as the terminal condition for the one-step scenario is to reach specific action strings: 'A' and 'ATTA'. This outcome aligns with the task requirements to reach the target states.

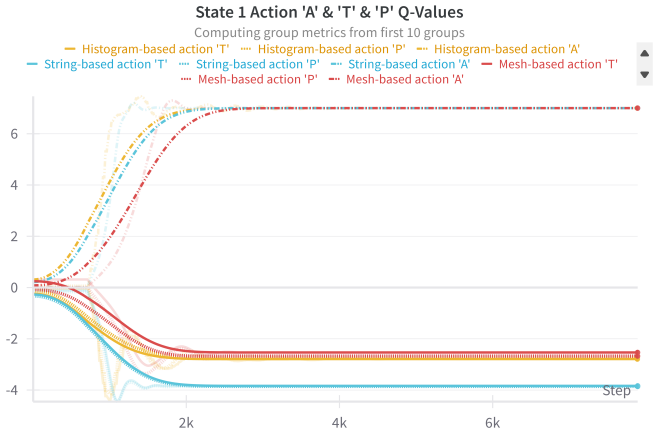


Figure 4.5: Forward one-step: state 1 action q-values

The Q-network loss rapidly decreases during the initial training phase. This early stage reduction is expected, as the model quickly learns and refines its

value estimates for a single state. The loss subsequently stabilizes at a low level, indicating that the DQN has converged, with limited further improvements in the agent's performance. The convergence to a low level suggests that the agent has successfully learned that action 'A' is the only viable option for reaching the terminal state, indicating no need for further exploration adjustments.

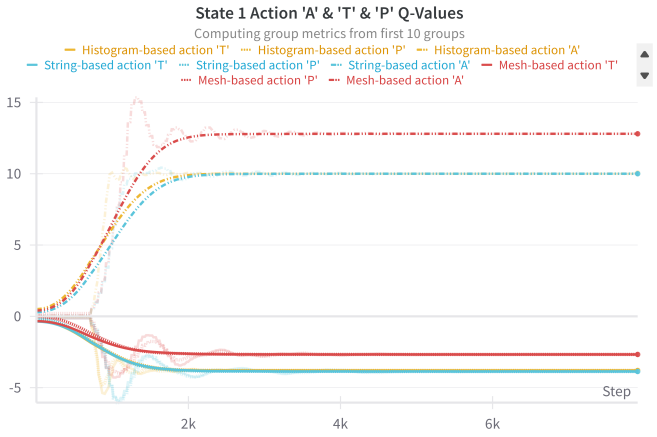


Figure 4.6: Backward one-step: state 1 action q-values

The agent's preference for action 'A' is also evident in the q-value convergence trends across all three reward function setups and both forward and backward approaches in this one-step scenario. As shown in Fig. 4.5 and Fig. 4.6, the q-values for action 'A' converge positively, while the q-values for actions 'T' and 'P' converge negatively. The three reward mechanisms and their respective actions are distinguished by color and line weights: yellow, red, and blue colors represent the histogram-, mesh-, and string-based approaches, respectively. The solid line represents action 'P', the dash-dot line represents action 'A', and the dashed line represent action 'P'.

In DQN, the q-value represents the combination of immediate and expected cumulative rewards for a particular action in a given state, followed by an optimal policy. Since the one-step approach only includes a single step per design episode, it is intuitive that the q-values converge to the reward associated with that action, as it determines whether the terminal condition is met when choosing among only three possibilities.

The reward function structure has been designed for flexibility, allowing for application to any number of time-steps. The strength of the model is its general purpose which includes a time-step penalty of -1 at each step to discourage stagnation of the learning agent from settling into suboptimal policies. When the agent successfully reaches the specified terminal state, a large positive

sparse reward of 10 is provided, representing successful application of the quad-mesh grammar. For both forward and backward approaches, this reward is only issued when there is an exact match in the singularity histogram between the final state and the target state. Additionally, in the forward approach, an action-string comparison between the current and terminal states is included. Computationally, successful strip addition does not occur until after the third step in the forward approach, as a pair of 'A' actions with at least one turn movement command is required. Since the terminal action strings for the first three steps are 'A', 'AT', and 'ATT', the singularity histogram remains unchanged until the fourth step, warranting an inclusion of an action-string comparison that rewards the agent only when it applies the correct action sequence.

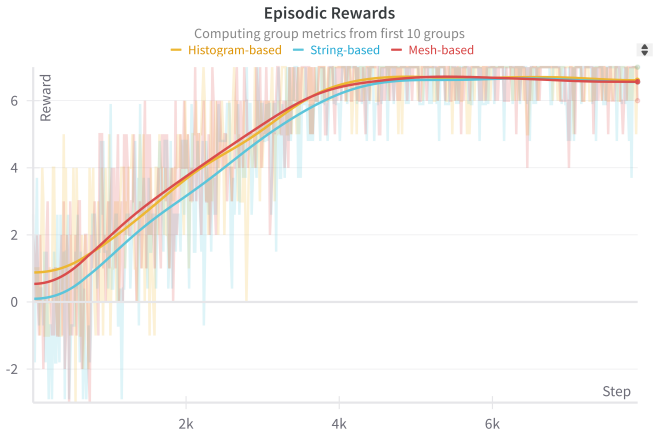


Figure 4.7: Forward one step: episodic rewards

Separately, a supplementary reward system is implemented to encourage alternative solutions introducing minor modifications to the mesh topology. Upon episode termination or truncation, a comparison of the initial and final singularity histogram is conducted. Episodes with no topological changes incur a negative sparse reward of -2, while those with modifications receive positive rewards that offset the time penalty and are proportional to the distance metric used. Both Fig. 4.7 and Fig. 4.8 illustrates a gradual improvement in agent performance over time, indicating progressive optimization of its policy. The plateau suggests that the agent has converged to an optimal policy, relying mainly on a single action.

The exploration rate, ϵ, for all models begins at 1.0 (pure exploration) and linearly deteriorates to 0.01 at a rate of 0.5 as shown in Fig. 4.9. Although the one-step scenario is relatively straightforward, this exploration strategy justifies the gradual improvement in the running reward average observed in

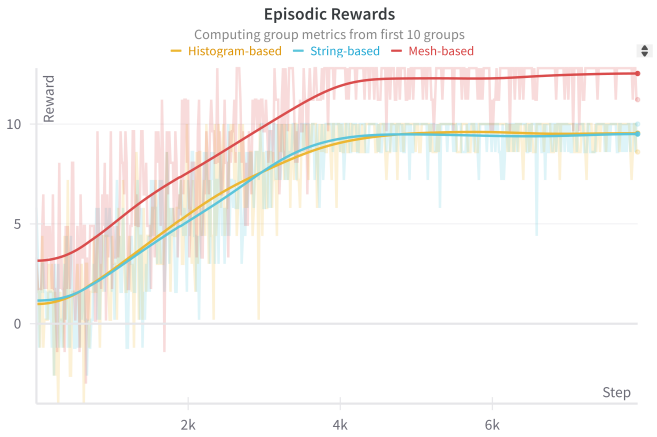


Figure 4.8: Backward one step: episodic rewards

the initial half of the training. The agent is encouraged to select alternative actions throughout a substantial portion of the training to promote exploration across all scenarios. Close inspection of the two reward plots reveals that the running reward converges to three general values: 7, 10, and 12.8.

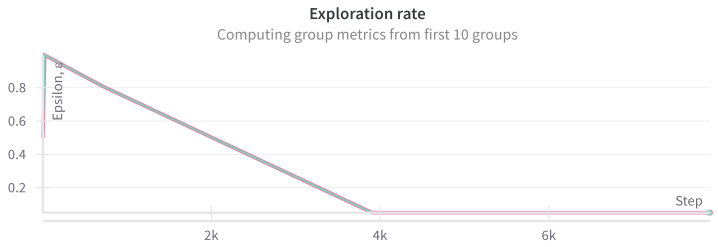


Figure 4.9: Epsilon decay

Across all reward mechanisms, each time step incurs penalty of -1. If no change in the topological data is recorded, an additional penalty of -2 is applied. Upon reaching the terminal state, a larger positive reward of 10 is distributed. The sum of these rewards and penalties accounts for the convergence to a value of 7 for all forward approaches. In the backward method, the reward function structure leads to a convergence close to 12 for the mesh-distance metric, while the string- and histogram-distances converge around 10.

The additional reward component for mesh-distance provides a shaped reward, where successful implementation of the 'ATTA' sequence yields a reward composed of the time penalty, the large positive reward, the mesh distance, and

a reward for topological intervention. Specifically, a successful sequence results in a reward of 12.8, calculated as follows:

$$\begin{aligned} Reward &= p_{time} + r_{terminal} + d_{mesh} + (-p_{time} * n_{max} + d_{mesh} * n_{max}) \\ &= -1 + 10 + 1.4 + (-(-1) * 1 + 1.4 * 1) \\ &= 12.8 \end{aligned} \tag{4.1}$$

For the histogram- and string-distances, successful implementation of the quad-mesh grammar yields a reward of 10, as the avoidance of the time-step penalty suffices, with the distance metrics contributing no additional value due to convergence to zero. The difference between these two metrics and the mesh-distance lies in their incentive structure: the former two encourage minimizing the distance metric, whereas the mesh-distance is formulated as a positive shaped reward, promoting behavior that successfully implements meaningful action sequences at an intermediate level between string and topological spaces, leveraging the grammar-based logic for strip insertion. Despite variations in reward structure, all agent converge to the desired outcome due to the simplicity of the one-step scenario.

4.1.4 TWO-STEPS

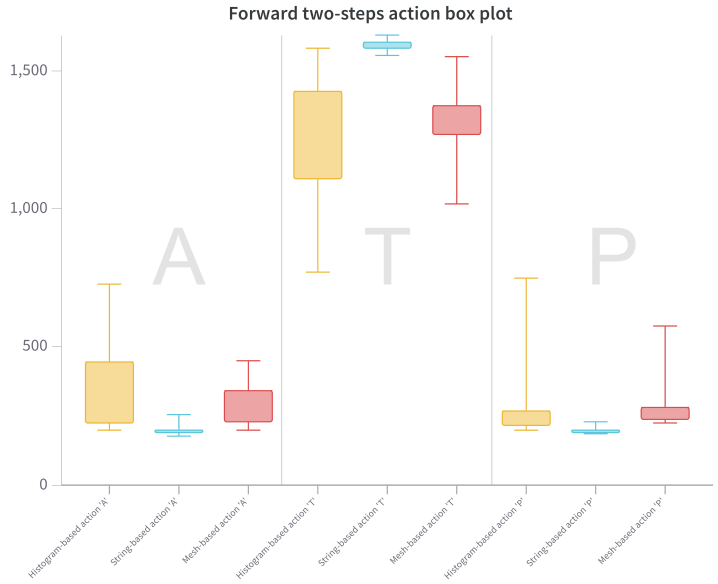


Figure 4.10: Forward two-steps: action box plot

By introducing a second possible action, the complexity of the scenario increases, and the learning agent begins to show early signs of difficulty in achieving the simple design task. The box plots in Fig. 4.10 and Fig. 4.11 summarize the distributions of selected actions separated based on the forward and backward approaches.

In the forward approach, the selection of actions 'A' and P' shows similar trends, with lower occurrences compared to action 'T'. In contrast, in the backward approach, a high occurrence of action 'A' is observed, followed by the selection of actions 'P' and 'T'. This difference in recorded actions can be attributed to the directionality of the two-step scenarios, where the learning agent initiates exploration from two different initial states: an empty state (''), and 'AT'. In the latter, selecting action 'A' leads to strip addition, even though 'ATA', Fig. 4.15, does not perfectly align with the desired target outcome 'ATTA'. However, this alternative sequence positively impacts the distance metrics while avoiding penalties for an unchanged mesh topology.

In the forward approach, all q-values converge negatively in the first state, which is expected since the learning agent receives only time-penalties in the initial step. The action-value functions illustrated in Fig. 4.12 indicate that actions 'A' and 'P' yield similar values across all the reward function setups, suggesting

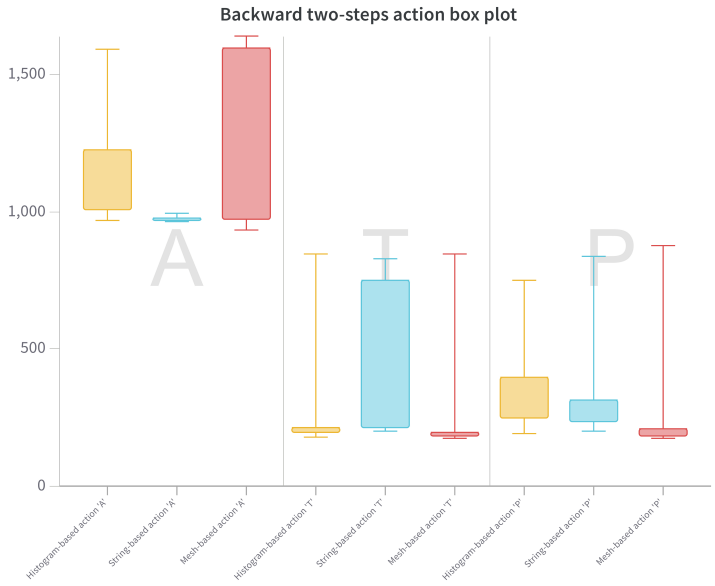


Figure 4.11: Backward two-steps: action box plot

that selecting either action leads to comparable outcomes in terms of immediate and potential future rewards. The convergence of action 'P' shows a more pronounced negative response in the string-distance method, a consequence of the Levenshtein distance (L-distance): selecting 'A' or 'T' in the first step incurs an L-distance of 1 to the terminal state, whereas 'P' results in a distance of 2 due to the additional substitution operation required to reach the target string, as 'P' does not exist in the desired outcome. Fig 4.12 shows the q-value convergence, in the forward approach, for action 'T' as the least negative outcome, indicating that the learning agent has identified this movement command as minimizing immediate and future negative rewards.

In the second state of the forward approach, the action-value functions of each actions exhibit substantial instability, reflecting the agent's difficulty in optimizing its policy based on available observations and reward structures. For the histogram- and mesh-distance metrics, the failure penalty in the final step results in -3, combining the time-step and no-change penalties. Due to the limited length of the action-string meaningful interventions that would affect the mesh topology are prevented. Conversely, the string-based metric has a failure reward range between -4 and -5, as non-target states incur penalties from recombination and substitution operations, in addition to the aforementioned penalties. Upon successfully reaching 'AT', the agent receives a reward of 7, calculated as 10 plus the failure penalty of -3. The discrepancy between the

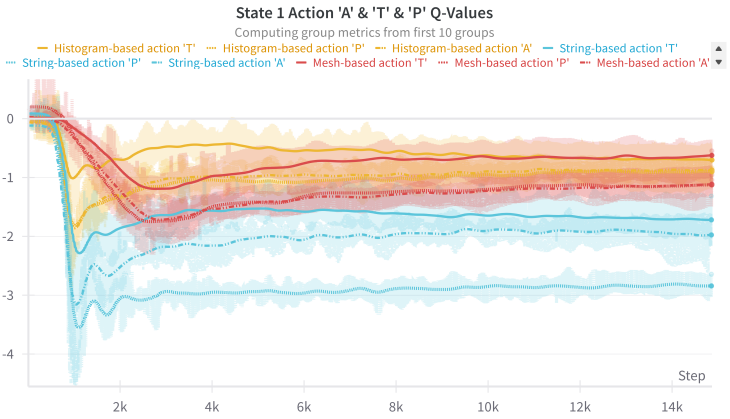


Figure 4.12: Forward two-steps: state 1 action q-values

the recorded q-value convergence and the large positive sparse reward can be explained by the high likelihood of action 'T' to be selected in the first step, denying the potential for better performing reward in the second step.

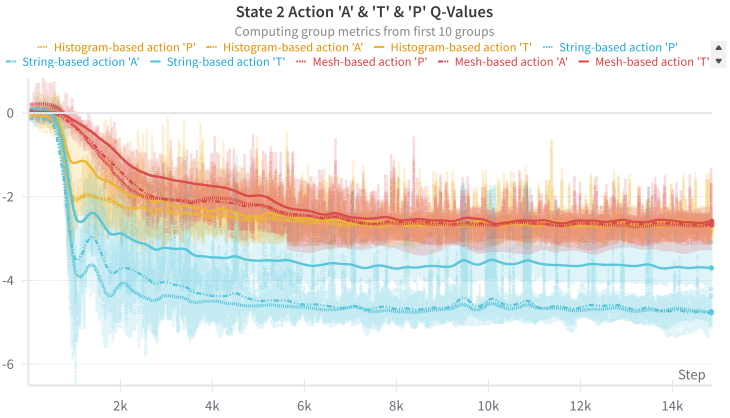


Figure 4.13: Forward two-steps: state 2 actions q-values

Both histogram- and string-distance methods converge to 'TT' by the end of the training period, while the mesh-distance metric retains a closer margin for the q-value across all actions, resulting in alternating choices between 'TA', 'TP', and 'TT'. The number of successful design episode across different reward mechanisms -54, 48, and 52, respectively- indicates that the limited heuristic occurrences of the correct action-sequence did not significantly impact the negative convergence trends. For these action strings, the histogram- and mesh-

based reward mechanisms yield episodic rewards of -2, while the string-based rewards result in -3, as shown in Fig. 4.14.

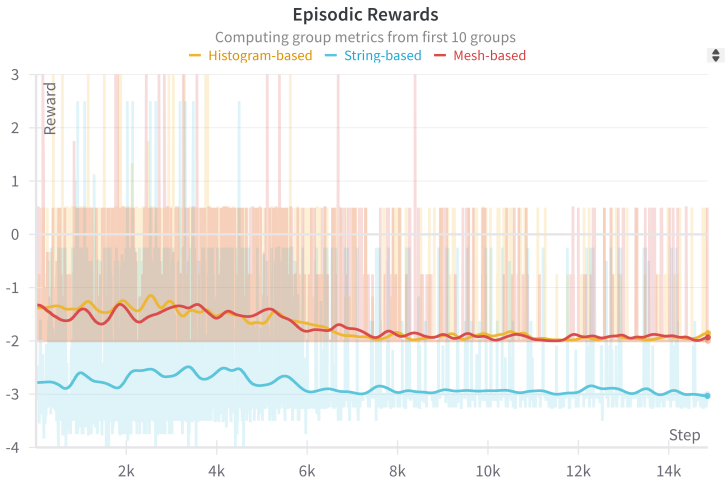


Figure 4.14: Forward two-steps: episodic rewards

In all scenarios, the exploration rate decays uniformly over time. This setup was chosen to encourage the agent to explore a range of randomized heuristic action sequences, potentially leading to innovative solutions or serendipitous discovery of the optimal sequence. However, the DQN system shows limited ability to consistently recognize and reinforce the correct action sequences derived through this exploration. This limitation may be attributed to the impact of sparse positive rewards that are distributed only at the end of the design episode, infrequently generating TD gradients that are potentially too large for the DQN model to effectively handle, thereby complicating its interpretation of action sequences. If the gradient of improvement is too steep, marked by a high TD error, it may dis-incentivize the agent from learning the target policy, favoring training stability over accuracy in replicating the optimal sequence. Although the discount rate is set at 0.99, which should theoretically allow rare instances of positive rewards to be effectively leveraged in the DQN's experience replay, the hyperparameter prioritization of minimizing the loss function between the network and target policies could inhibit the emphasis on isolated positive rewards. Consequently, the agent's policy is balanced based on the accumulated history of experiences, the majority of which are instances of negative penalties. When frequently exposed to negative rewards, the agent initially learns to avoid actions associated with the most adverse outcomes. However, the predominance of negative rewards, may lead the agent to converge on a biased policy that minimizes negative outcomes without actively seeking sparse positive rewards.

One additional factor destabilizing the q-value for action 'A' is the truncation condition triggered by sequences like 'AA'. The computational errors inherent in the compas_quad algorithm impede the learning agent's ability to correctly apply action 'A', resulting in its q-value being ranked low. This misinterpretation is problematic, as action 'A' is essential for the proper functioning of the quad-mesh grammar. DQN estimates an action-value function to approximate a deterministic policy that selects decisions based on the highest estimated q-value at each state. This approach, aiming for convergence to stable q-values for each state-action pair, may not align well with the nuanced, hierarchical action structures present in the quad-mesh grammar, further inhibiting effective learning.

In the backward approach, the q-value for action 'A' shows a slightly positive convergence in the first state compared to its forward counter part, partially due to the possible alternative solution 'ATA'. The string method similarly aligns with the forward approach, Fig. 4.16, where the q-value of action 'P' is lower than that of action 'T' due to the L-distance. As action 'A' shows the least negative or most positive convergence, the plots highlights the reinforcement learning agent's tendency to converge to an unintended policy that constructs the mesh layout illustrated in Fig. 4.15 rather than the targeted state.

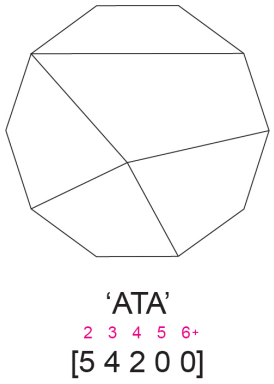


Figure 4.15: Alternative mesh 1: 'ATA'

For both the mesh- and string-based methods the converged q-values slightly resemble the rewards obtained in the first round of decision-making, which can be referenced in Fig. 4.18, a schematic illustration of all possible string combinations in the backwards two-step approach, alongside the final mesh structures and their associated rewards. For the histogram-distance reward function, however, all rewards available in the first state are identical, confirming that the observed differences come from the expected rewards available in the second state. This also applies to the minor deviations between q-values and rewards observed for the other two methods.

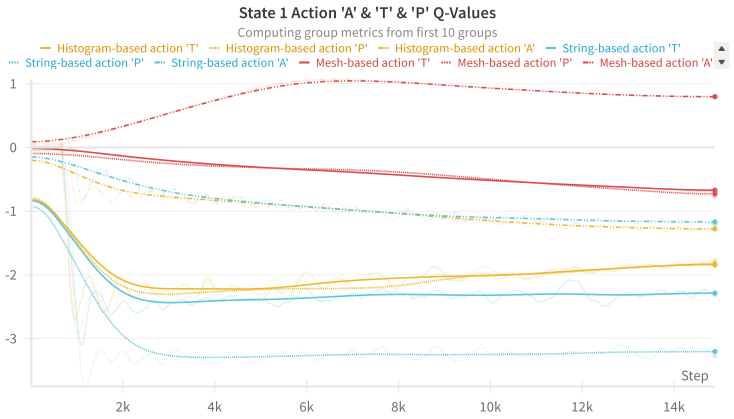


Figure 4.16: Backward two-steps: state 1 actions q-values

In the second state, during the first half of the exploratory training process, q-values for all actions are highly unstable as the DQN model's exploration rate decays. In Fig. 4.17, results show q-values for the mesh-based rewards improve late into the training period, with the highest observation of approximately 4.5 for action 'A'. The same goes for the histogram-based method with a value of 2.5 for action 'A', however, the range of difference across all three actions are negligible. For the string-based method, the value for action 'A' ranks visible lower than the other two actions, suggesting high preference for action 'A' in the previous state, leads to decisions to select a different action to minimize the string-distance in the second step. Despite the selection of action 'T' minimizing the string-distance in the second step, the positive rewards distributed when reaching an alternative mesh layout neutralizes the difference between the selection of actions 'T' and 'P', resulting in the same approximate q-value of 3.4.

The mesh-distance based rewards reveal that the second character in the sequence remains invariant when action 'A' is chosen in the first state. This leads to policy convergence that favors the upper branch of the decision tree, as the agent can consistently obtain positive rewards. Additionally, the mesh-based rewards and q-value convergence for action 'A' in state 1, as the only positive selection across all cases, support this analysis. Although the maximum reward is achievable with the sequence 'ATTA', only 5.8% of the design episodes successfully reached this target state. The mesh-distance model in the two-step sequence predominantly converged on the action string 'ATAA'. Importantly, the final 'AA' in 'ATAA' does not trigger a computational error because the last action functions as an open addition sequence; thus the action string is better interpreted as 'ATA' followed by 'A'.

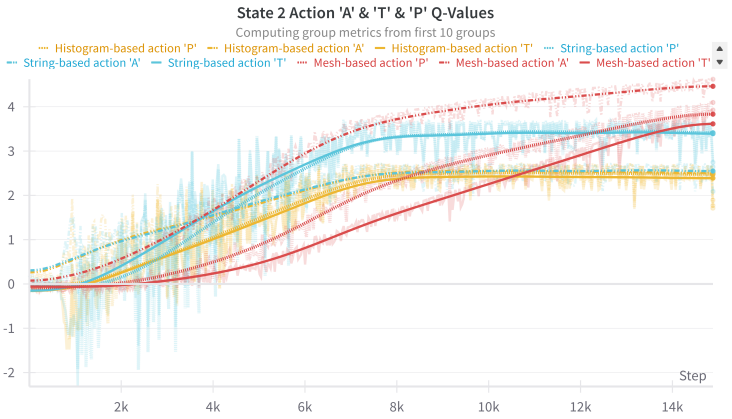


Figure 4.17: Backward two-steps: state 2 actions q-values

The action strings ’ATAT’ and ’ATAP’ emerges as the converged result for the string-based method, providing additional insight into the agent’s behavior. In the first state, actions ’A’ and ’T’ offer equivalent rewards. Selecting ’A’ restricts possible rewards in the upper branch to values of 2 and 3, while, selecting ’T’ allows for rewards in the middle branch to values either 11 or -4. Moreover, the large sparse positive reward is accessible only with a one-in-three probability following the selection of action ’T’ in the first state. Based on these reward distributions and the convergence results for the string-action, the DQN agent can be inferred to favor stable outcomes or higher probabilities of positive rewards, rather incurring negative penalties by attempting the target action sequence. The fluctuation in q-value convergence for actions ’T’ and ’P’ in state 1 may be influenced by the existence of rewards 15.2 and 5.2, which are available at a lower probability compared to those in the top branch of the final state. The same interpretation holds for the histogram-based method, where the string-action converges to ’ATAP’. The reward magnitude are similar for both histogram- and string-based methods, with success rates of 6.1% and 5.8%, respectively. The running reward average for the two steps, Fig. 4.19, for the mesh-, string-, and histogram-based methods are approximately 2.5, 0.5, and 0, respectively, aligning with the average total rewards obtained for each converged string-actions.

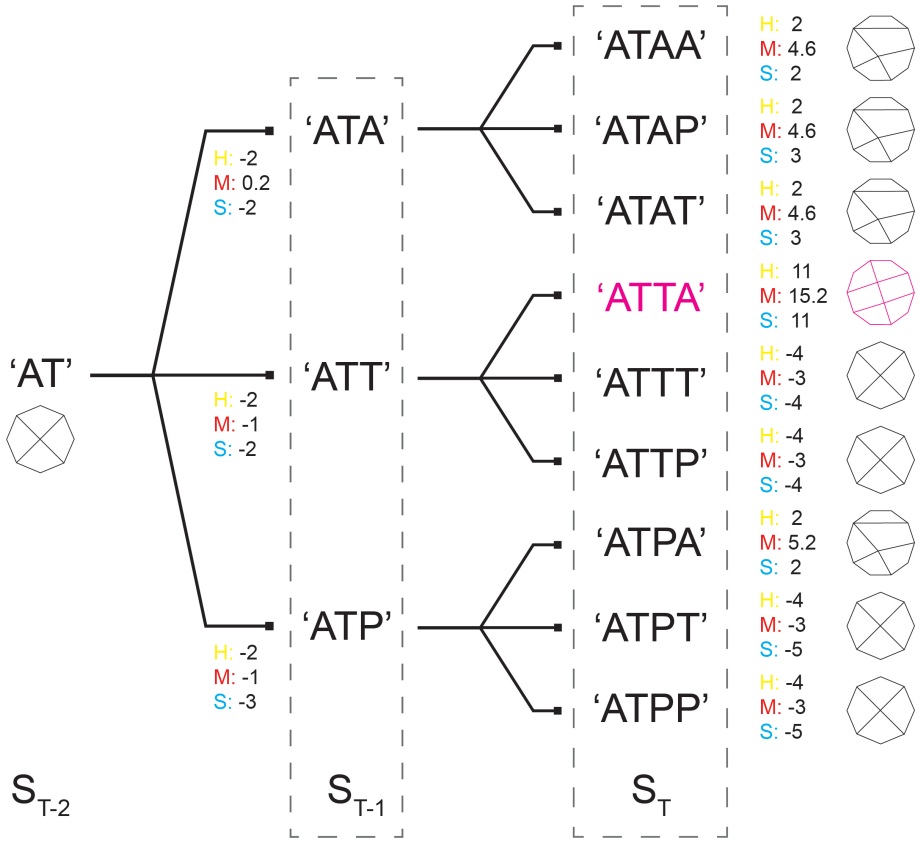


Figure 4.18: Obtainable rewards and mesh structure in the backwards two-steps

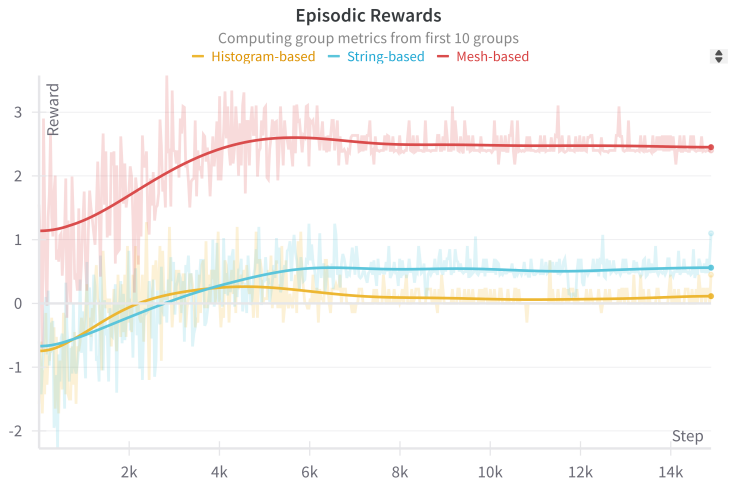


Figure 4.19: Backward two-steps: episodic rewards

4.1.5 THREE-STEPS

In the forward approach, the agent demonstrates a preference for action 'T', showing limited variance across all letters among the three reward structures in Fig. 4.20. A breakdown of all action sequences performed by the learning agent is illustrated in Fig. 4.21. Crossed-out pathways represent sequences that trigger truncation conditions due to computational errors (e.g., 'AA', 'APA')

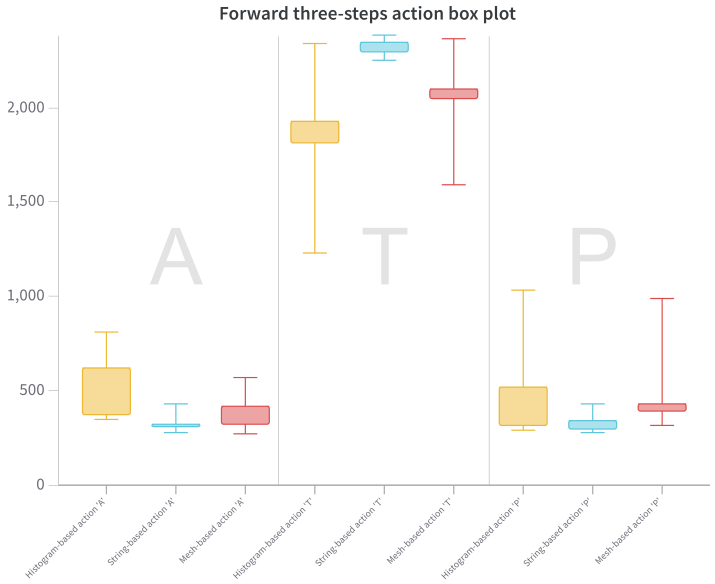


Figure 4.20: Forwards three steps: action box plot

In the first time-step, in Table 4.1, action 'T' exhibits the highest q-value across all reward mechanisms, with histogram- and mesh-based methods placing action 'P' as second, followed by action 'A', which shows the largest negative convergence. The string-based rewards uniquely trend with action 'A' as the second least favorable choice compared to 'P', as seen in previous cases. All action-value functions outputs are negative, reflecting the time-penalty incurred during early steps in the design task. In the three-step scenario, with an increased maximum step count, the observed q-values become more nuanced, incorporating additional states that grow exponentially with each addition to the action string length. This extended sequence length increases the likelihood of accumulating negative rewards and truncation instances through inconsequential action strings, impacting the agent's experience and q-values.

The magnitude of q-values in the first time step is similar for both the histogram- and mesh-based reward functions, as both incur the same penalties.

The histogram-based distance metric is zero, due to the incomplete strip addition sequence in the target state, while the mesh-based reward function cannot distribute intermediate rewards in the forward approach since the action string in the initial state is empty, and a valid strip addition requires a minimum action string length equal to the maximum steps specified for this set-up. However, the L- distance used in the string-based reward function is the reason for its lower convergence trends.

In the second time-step, the histogram- and mesh-based mechanisms show action 'T' as the least negative q-value, indicating a preference over action 'P' and 'A'. Again, since the the current state is intermediary, the q-values reflect the available immediate rewards and the expected future rewards that have converged to a sub-optimal policy. The q-values for action 'T' in the string-based approach shows noticeable in the second state. The learning agent develops an understanding that if actions 'A' or 'T' is selected in the first step, execution of command 'T' is preferable in the second stage since it minimizes the L-distance to the target sequence, 'ATT'.

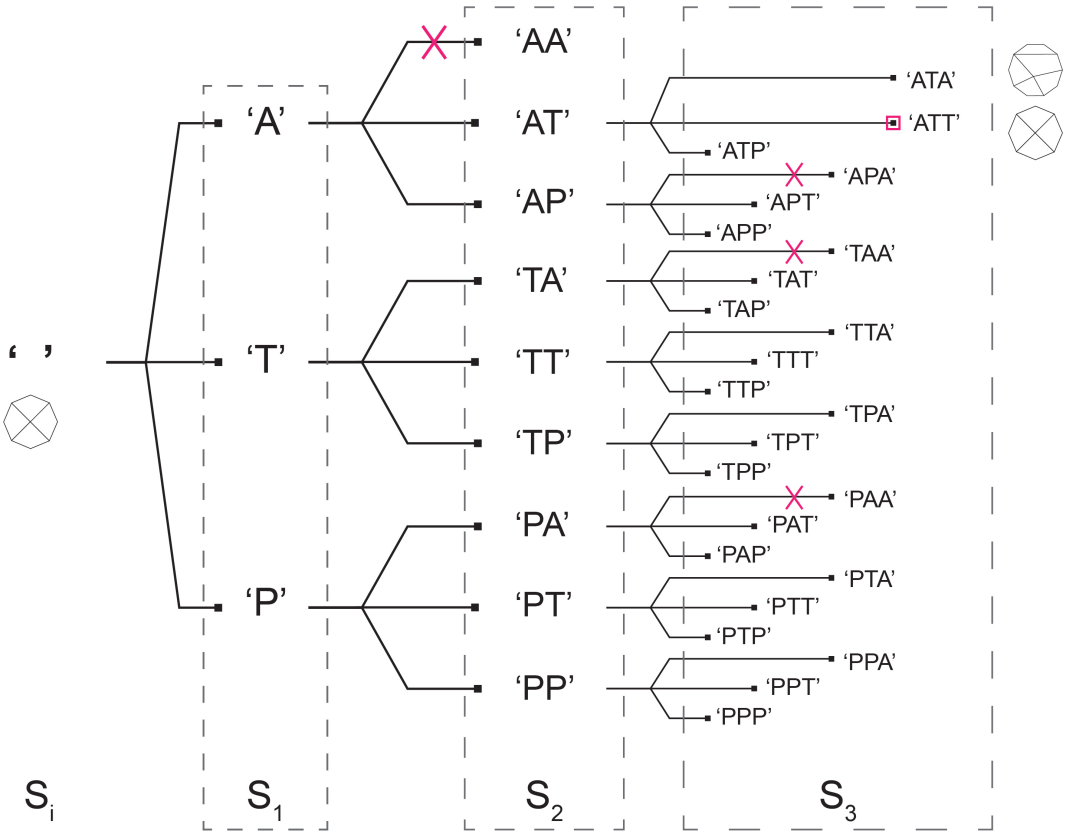


Figure 4.21: Forward three-steps action sequence diagram

In the final time-step, the mesh-based method shows the smallest range of q-values among the three actions, indicating marginal differences in action selection. Across all reward structures, action 'T' remains the most favorable. In general, the learning agent behaves in a manner, that places an emphasis on the letters chosen in earlier steps, as evidenced by the smaller q-value range across all reward methods in the final steps.

The action sequences that converged across all three reward function setups were 'TTT' and 'PPP' or the recombination of actions 'T' and 'P'. The learning agent was not able to boil down the final sequence to a single distinct policy, resulting in different action sequences such as 'TPT', 'PTT', 'TTP', 'TTA', and 'ATT' during the final stages of the training period. The success rate in reaching target state 'ATT' was 3.5%, 2.9%, and 3.1% for the histogram-, mesh-, and string-based rewards, respectively -comparable to the theoretical 4% probability of achieving the desired outcome in 1 out of 25 valid sequences. Notably, the majority of these successful instances occurred in the first half of the training process, when the exploration rate was higher. The probability of reaching the simplest action string that introduced mesh modification, 'ATA', was much lower, with success rates of 1.03%, 1.3%, and 1.1% respectively, indicating a failure to establish a desirable policy for the forward three-step approach.

Table 4.1: Forward three steps: final q-values for initial, intermediate, and final states

Reward	'A'	'T'	'P'
State 1			
Histogram	-0.96	-0.69	-0.91
Mesh	-1.05	-0.61	-1.06
String	-3.1	-2.8	-3.6
State 2			
Histogram	-0.93	-0.68	-0.85
Mesh	-1.4	-0.68	-0.85
String	-3.05	-1.8	-3.4
State 3			
Histogram	-2.6	-2.5	-2.6
Mesh	-2.4	-2.3	-2.4
String	-4.4	-3.6	-4.5

The running reward average, shown in Fig. 4.22, converged to -1.6 and -1.65 for the histogram- and mesh-based rewards, respectively, and -3 for the string-based reward. The first two values reflect the average of rewards accumulated during a single design episode, where -1 and -3 are incurred in the first two steps and final time-step, resulting in an overall mean of -1.67. For the intermediate

and terminal states of the string-based method, the following list defines the range of possible rewards:

- Step 1: A, T: -3; P: -4
- Step 2: AT, TT: -2; AP, TA, TP, PT: -3; PA, PP: -4
- Step 3:
  - ATP, APT, PTT, TTT: -4
  - APP, TAT, TPT, TTA, TTP, PAT, PPT, PTA, PTP: -5
  - TAP, TPA, TPP, PAP, PPA, PPP: -6
  - ATA: 4; ATT: 7

For the converged string action 'TTT', the average of the rewards -3, -2, and -4, for each step, align with the final average reward value, confirming the consistency of the results.

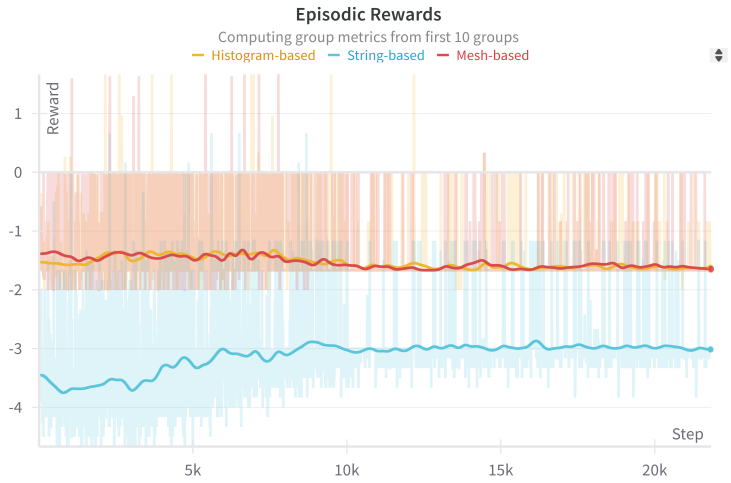


Figure 4.22: Forward three steps: episodic rewards

Similar to the forward approach, the agent shows a preference for actions 'T' and 'P' over 'A' in the backward approach as seen in Fig. 4.23. Action 'T' is the most selected in the string-based method, with a mean of 2359.4 and a low standard deviation of 55.5. On average, it was selected 1327.1 times in the mesh-based method and 1304.9 times in the histogram method, both showing less counts compared to the forward approach. Conversely, 'P' is another frequently chosen in both the mesh- and histogram-based methods with an average count of 1378.4 and 1327.1, respectively, showing significant difference to 294.5 in the

string method. The length of between the lower and upper quartiles also suggest that this trend is consistent among all trials. 'A' appears 346.1 times in the string, 312.6 in the histogram, and 294.5 in the mesh. 'A' is chosen infrequently across all three reward function structures, and there remains higher variance for action 'P' and 'T' in the histogram-based reward function.

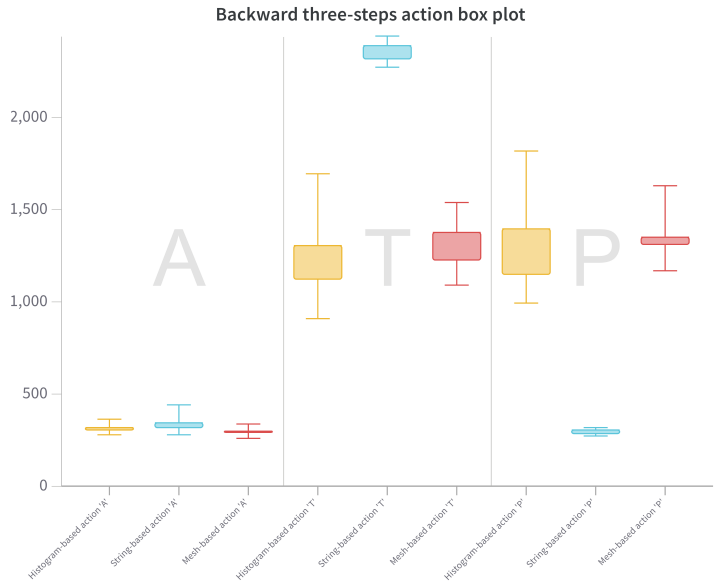


Figure 4.23: Backwards three steps: action box plot

In the first state, the q-values in Table 4.2 vary across reward mechanisms. Actions 'T' and 'P' show the least negative q-values, approximately -1.2, while 'A' has nearly twice this magnitude in the mesh-based method. For the histogram-based method, actions 'T' and 'P' have q-values around -2.6 and -2.8, with -3.8 for action 'A'. For the string-based method, actions 'T' and 'A' are least negative at around -2.8 and -3.2, with action 'P' closer to -3.7. In the backward approach, the initial state of the string action encompasses action 'A', hence, the learning agent identifies that an additional 'A' is suboptimal due to its immediate triggering of the truncation condition. Likewise, 'P' is penalized as an initial action in the string-based method.

While the q-values show high instability throughout training for the second time-step, the average q-values did not change significantly from the previous time-step across all reward mechanisms. The action value functions for 'T' and 'P' remain competitive, explaining why the learning agent frequently selected 'P' in the histogram- and mesh-based methods. Immediate rewards across actions

are similar, largely reflecting only the time-step penalty, as modification checks occur only in the final step of the design episode.

Table 4.2: Backward three steps: final q-values for initial, intermediate, and final states

Reward	'A'	'T'	'P'
State 1			
Histogram	-3.5	-2.4	-2.4
Mesh	-2.05	-1.2	-1.2
String	-4.1	-2.8	-3.6
State 2			
Histogram	-3.5	-2.4	-2.4
Mesh	-2.04	-1.1	-1.2
String	-4.1	-2.7	-3.6
State 3			
Histogram	-3.5	-2.3	-2.3
Mesh	-1.6	-0.79	-0.81
String	-4.04	-2.7	-3.5

It should be noted that q-values in the final step exhibit high instability due to the sparse rewards associated with terminal state checks, compounded by the exploration fraction. Despite the high variance, the final q-values in all reward structures remain consistent from step 1, and action 'T' has the least negative q-value, indicating it is the preferred action.

Across all actions, the agent converges toward the lower bounds of q-values, rather than the upper bounds that represent successful instances of the design episodes. This outcome is due to the scarcity of positive rewards, which are available only at the final time step of each episode. Although the agent encounters these positive rewards during the exploration phase, the discount rate does not sufficiently incentivize prioritization of these rewards. Instead, the Q-network converges on minimizing immediate penalties, which highlights the network's focus on minimizing Q-network loss.

The q-values in Table 4.2 illustrate negative convergence trends suggesting the agent is unable to consistently produce meaningful string actions, and subsequent final rewards of -4 for the histogram-based method, -3 for mesh, and -4, -5, or -6 for the string-based method. The differences in final penalties, due to lack of mesh modifications, align with the varying q-value magnitudes observed across the three reward mechanisms. Furthermore, across all reward structures, the final action sequences are observed to converge to 'ATTT', confirming the argument.

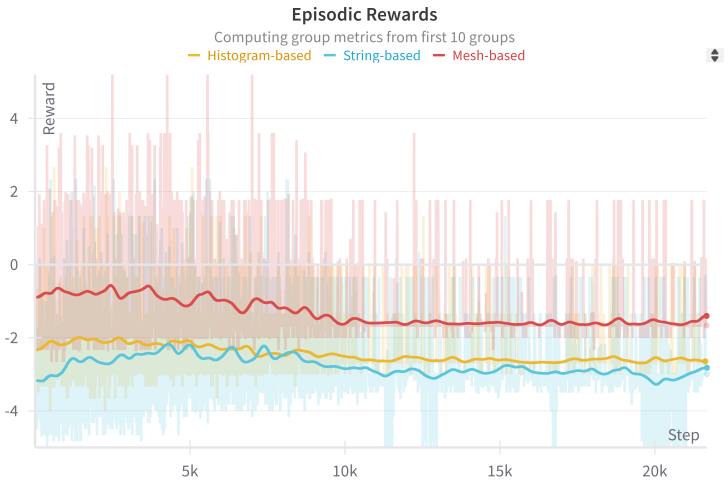


Figure 4.24: Backward three steps: episodic rewards

The histogram-distance metric consistently returns a negative shaping reward in the backward approach, resulting in an intermediate reward state of -2 when combined with the time-step penalty. If no changes to the mesh topology are recorded, the final reward is -4. Conversely, sub-optimal action strings that introduce alternative mesh layouts (e.g. 'ATA') yield a reward of 4, while reaching the specified target state results in a reward of 12. These two positive rewards were achieved in only 7.01% and 2.2% of instances, respectively. The converged action string, however, oscillated between 'ATTT' and 'APPP', both of which produce a reward average of -2.67 ((-2-2-4)/3=-2.67), aligning with the converged value of -2.66 in Fig. 4.24.

The mesh-based reward function also returned consistent values for intermediate (-1) and final states (-3) that did not include valid strip-addition sequences. Of all design episodes, 7.1% successfully introduced alternative mesh states, with a 2.4% success rate in reaching 'ATTA'. Similar to the histogram-based method, the most common action strings during training were 'TTT', 'TTP', and 'PPP' occurring 81.1% of the time. The reward averages for these action strings is calculated as -1.67 ((-1-1-3)/3=-1.67), aligning with the converged value of -1.3.

Finally, the string-based reward structure offers a broader range of rewards, as seen in the forward approach to the three-step scenario. Intermediate rewards range from -2 to -4, based on the L-distance from the target action string 'ATTA'. For alternative mesh solutions, a positive reward is assigned proportional to the distance metric, alongside a penalty offset for the time-step penalty incurred. This approach inadverdently caused valid action strings closer in L-distance to the target sequence receiving lower rewards than those involving more

modification operations to resemble the desired state. Consequently, action strings 'ATAA', 'APTA', and 'ATPA' were rewarded with 4, while 'ATAT' and 'ATAP' received 6. Together, these reward value occurred in 8.9% of instances. The target sequence and mesh structure associated with 'ATTA' was reached 4.3% of the time, while the converged action sequence was, again, 'ATTT'. The reward average for 'ATTT' in the backward string-based reward structure is -3 ((-3-2-4)/3=3), aligning with the converged value of -2.73.

4.1.6 FOUR-STEPS

The final process of the sequential exploration for the action string 'ATTA' is consistent across both the forward and backward approaches. The objective of the learning agent is to reach the terminal state from an empty string, with a total of 67 viable action strings and an approximate 1.5% probability of reaching the target state. The challenge for the learning agent to achieve this outcome further increases, as reflected in the results presented in this section.

Action 'T', in Fig. 4.25, shows the highest preference across all three reward mechanisms. However, substantial variability exists among runs for the histogram-based method, with action 'P' contributing to the observed high outliers.

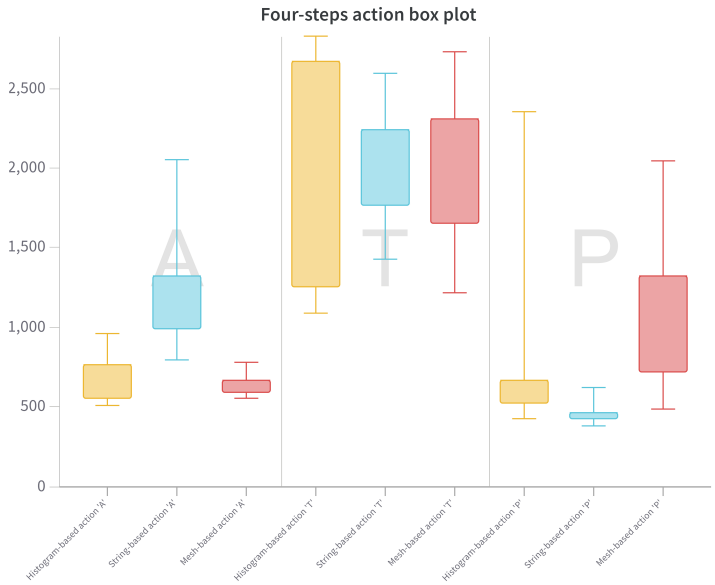


Figure 4.25: Four steps: action box plot

Table 4.3: Step four: Mean and standard deviation of average action history

(μ, σ)	'A'	'T'	'P'
Histogram	700.1, 162.4	2130., 747.0	1170., 802.6
Mesh	655.3, 76.99	2103.9, 510.3	1242.8, 521.4
String	1372., 406.3	2156., 400.5	472.3, 69.59

Table 4.4: Step four: final q-values for initial, intermediate, and final states

Reward	'A'	'T'	'P'
State 1			
Histogram	-2.0	-1.7	-1.8
Mesh	-1.1	-0.95	-0.99
String	-4.0	-3.7	-4.6
State 2			
Histogram	-2.1	-1.7	-2.0
Mesh	-1.2	-0.85	-1.1
String	-3.3	-2.7	-3.8
State 3			
Histogram	-2.2	-2.1	-1.9
Mesh	-1.2	-0.84	-0.99
String	-3.2	-3.1	-3.7
State 4			
Histogram	-3.5	-3.5	-3.5
Mesh	-2.6	-2.5	-2.6
String	-3.4	-3.8	-4.1

Across all reward structures. q-values converge toward immediate penalties incurred in intermediate steps. Similar to previous scenarios, the lack of convergence to positive values -alongside consistent minimization of the Q-network loss for all reward mechanisms- suggests that the agent has learned an optimal policy focused on reward minimization. All figures are included in the annex.

In state 1 and 2, action 'T' is preferred across all reward mechanism. By state 3, the agent using the histogram-based reward structure shows a slight preference for action 'P'. In the final state, preferences vary clearly: differences between actions diminish in the mesh- and histogram-based methods, while action 'A' in the string-based method shows the least negative q-value.

The converged action strings and episodic reward averages are as follows:

- Histogram: 'TTTA', 'TTTT', and 'TTTP'; -2.5
- Mesh: 'TTTT', 'PTTT', and 'TTTP'; -1.5

- String: ’TTTA’ and ’TTAT’; -3.53

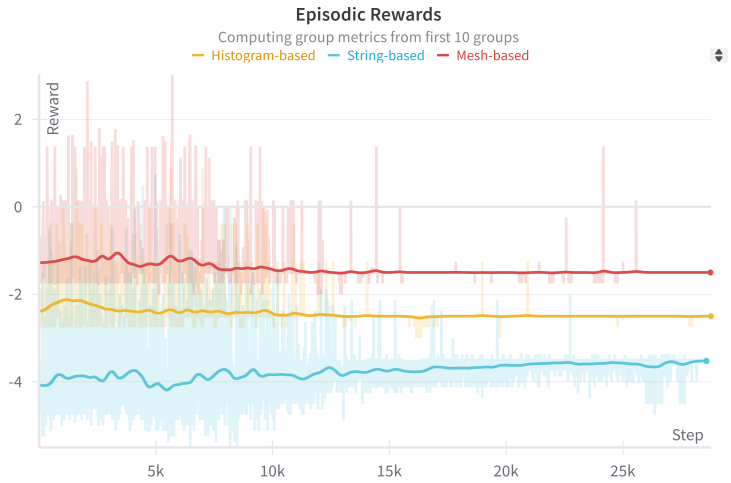


Figure 4.26: Four steps: episodic rewards

The histogram- and mesh-based reward functions do not effectively assist the agent in navigating the design space. The absence of convergent strings indicative of alternative mesh modifications, as well as limited execution of strip additions, shows that these reward structures have minimal influence on the learned action sequence. The action string ’ATTA’ was selected as a simplified target to:

- Verify successful integration of the compas_quad and reinforcement learning algorithms, and
- Provide a straightforward test of the agent’s ability to transition between specified states within the design space of quad-mesh grammars

Additionally, the reward structures for the application of quad-mesh grammar were initially constructed with an emphasis on generalization to avoid overly restrictive rewards that might lead to over-fitting to the specific target ’ATTA’ string.

However, the simplicity of the design problem, combined with these generalized reward structures, may have adversely affected the four-step sequential exploration due to the short length of the action string. This constraint significantly impacted q-value and final action sequence convergence, particularly in the histogram- and mesh-based methods in the final steps. As explained earlier, the histogram-based approach uses MSE to calculate a distance metric

based on differences between the current and target singularity histograms. For the short action string 'ATTA', with a histogram distance of 1 to a basic mesh starting mesh, the incentive to minimize this value may have been insufficient. Furthermore, because a minimum of three actions is required to execute a successful strip addition sequence, the four-step limit may have prevented the mesh-distance reward function from adequately issuing positive intermediate rewards to guide the agent in the correct direction.

4.2 GENERALIZED TARGET STATE

Fig. 4.27 illustrates the exploration of various target states beyond 'ATTA'. This experiment builds on the results of the four-step sequential exploration and aims to address the previously mentioned shortcomings. The same three reward function structures will be used to reach target states 'ATPTA', 'ATAATA', and 'ATPTTPTA' from an initial blank state, as shown in Fig. **??**. Additionally, a hybrid model that combines the three reward functions will be evaluated to determine whether synthesizing the different parameters can assist the learning agent in making more informed decisions. For the following experiments, the number of designated episodes for training has been increased to 2000 to accommodate convergence of model parameters due to increased lengths of action steps per design episode.

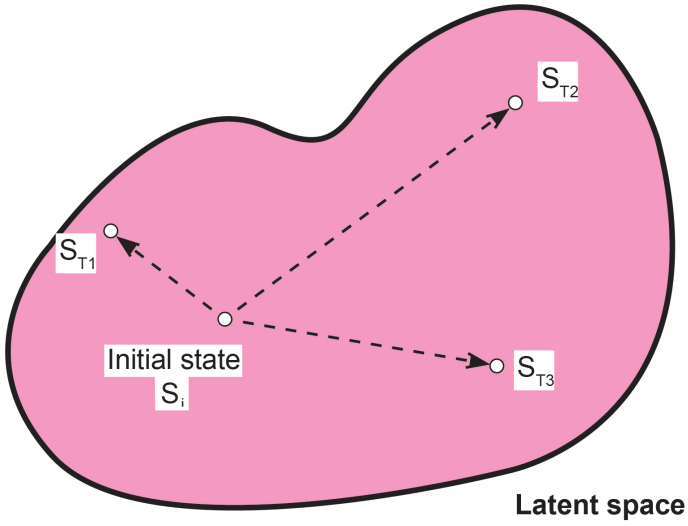


Figure 4.27: Generalized exploration between two steps

4.2.1 'APTPA'

For the five step sequence, Fig. 4.28 and Table 4.5 reveal a general trend of action 'T' being the frequently selected option, followed by action 'A' or 'P', depending on the reward structure. Specifically, in the histogram-based training, the learning agent exhibited a preference for action 'P', albeit with significant variability. In contrast, the mesh- and string-based reward structures showed a higher frequency of selecting action 'A' over 'P' on average.

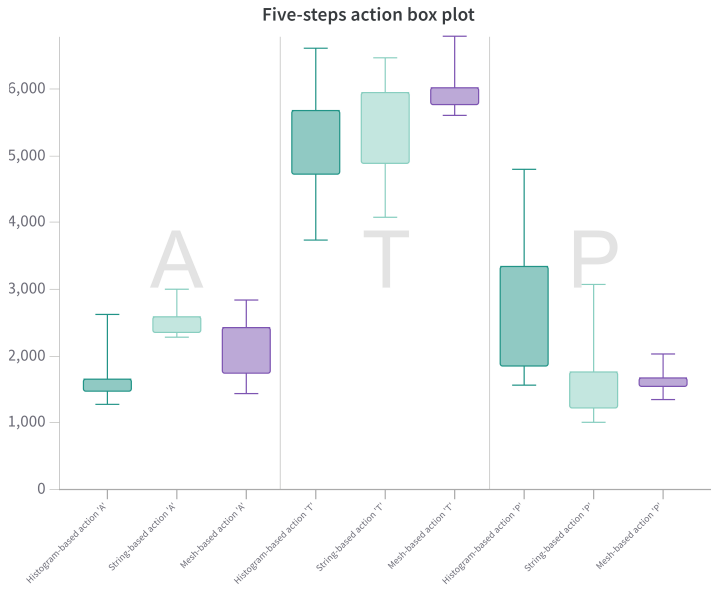


Figure 4.28: Five steps: action box plot

Table 4.5: Step five: Mean and standard deviation of average action history

(μ, σ)	'A'	'T'	'P'
Histogram	1792., 474.4	5324., 974.0	2887., 1180.
Mesh	2289., 529.2	6026., 374.4	1687., 220.2
String	2569., 236.4	5623., 883.9	1809., 718.7

Across the first four states, the final converged q-values indicate that action 'T' dominates across all reward functions. The moderate range of q-values suggests that the learning agent encounters challenges in developing an optimal policy that leads to the correct action sequence of the target state. While the mesh-based method also favors action 'T' in the final state, the histogram- and string-based

Table 4.6: Step five: final q-values for initial, intermediate, and final states

Reward	'A'	'T'	'P'	'A'	'T'	'P'
	State 1			State 2		
Histogram	-4.9	-4.3	-4.5	-4.9	-4.5	-4.6
Mesh	-2.2	-1.9	-2.1	-2.5	-2.1	-2.4
String	-13.4	-12.0	-13.0	-11.6	-10.3	-11.2
	State 3			State 4		
Histogram	-4.9	-4.4	-4.8	-5.7	-5.3	-5.6
Mesh	-2.5	-2.2	-2.6	-3.2	-3.0	-3.4
String	-9.5	-7.5	-9.8	-8.9	-7.9	-9.6

State 5			
Reward	'A'	'T'	'P'
Histogram	-6.7	-6.7	-6.2
Mesh	-4.2	-3.0	-4.2
String	-9.0	-10.4	-10.6

reward structures exhibit a notable decline in the q-value of action 'T', resulting in a shift towards favoring actions 'P' and 'A', respectively. It is important to note that the neural network architecture in the DQN model functions as a blackbox, where the internal weights adjustments made during training are generally not easily interpretable compared to traditional tabular methods, especially for complex decision-making processes. As the length of the design episode increases, the complexity of tracing the changes in updated q-values back to specific sequences of state-action pairs grows due to the compounded learning from various transitions and experiences stored in the replay buffer. Although the q-values presented in Table 4.6 may not precisely represent the immediate and expected future rewards as comprehensible to humans, the converged action sequences indicate the following outcomes across all three reward structures:

- Histogram: 'TTTTT' and 'PPPPP'; -2.5
- Mesh: 'TTTT', 'PTTT', and 'TTTP'; -1.5
- String: 'TTTA' and 'TTAT'; -3.53

All reward structures exhibit convergence towards non-essential action strings primarily composed of actions 'T' and 'P', which fail to produce significant topological transformations in the mesh layout. While the learning agent demonstrates instances of successful attempts, particularly during the initial training phase, the final convergence indicates that these sporadic developments of alternative strategies do not suffice for the emergence of a novel policy. Despite

the lack of success in learning an effective policy, the learning agent generates interesting alternative mesh topologies that, while different from the target state, are both visually and topologically unique, as illustrated in Fig. 4.29. Furthermore, isomorphic states are accurately identified, and rewarded during training, as shown in Fig. 4.30.

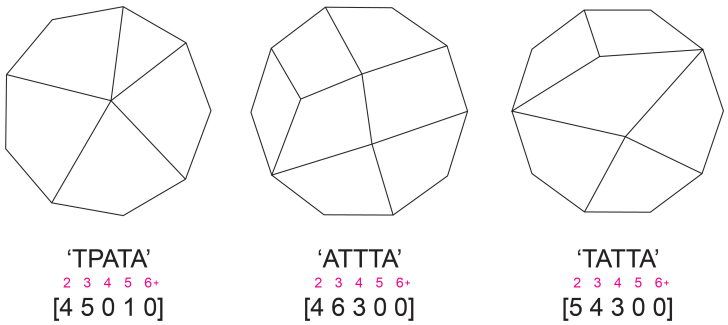


Figure 4.29: Step five: alternative mesh

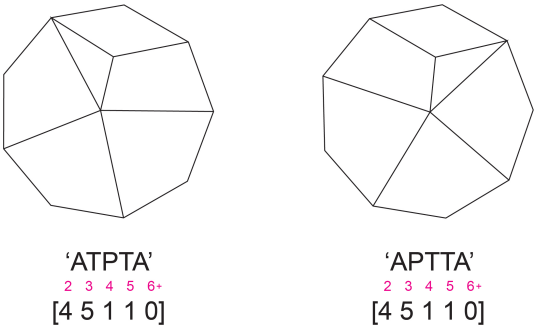


Figure 4.30: ATPTA isomorphism

4.2.2 'ATAATA'

In the six step sequence, a more pronounced divergence in behavior among the three reward mechanisms is observed. Fig. 4.31 illustrates that, under the string-based reward structure, there is an unusual preference for action 'A' over action 'P', a pattern not previously recorded. As detailed in Table 4.7, action 'A' was selected, on average, more than twice as often as action 'P', with minimal variance across the ten runs, in contrast to the results from both the histogram- and the mesh-based methods.

As the string length increases and convergence trends persist, only the episodic rewards for the string-based show a decline, highlighting the magnitude of

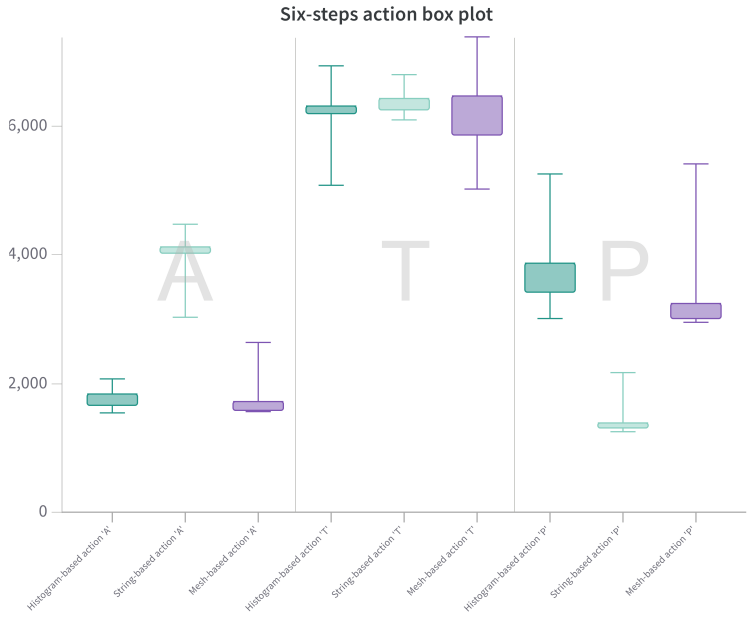


Figure 4.31: Six steps: action box plot

Table 4.7: Step six: Mean and standard deviation of average action history

(μ, σ)	'A'	'T'	'P'
Histogram	1805., 168.7	6355., 550.1	3839., 597.9
Mesh	1859., 357.4	6410., 799.0	3731., 901.2
String	4077., 414.7	6448., 232.7	1475., 270.7

Table 4.8: Step six: final q-values for initial, intermediate, and final states

	'A'	'T'	'P'	'A'	'T'	'P'	'A'	'T'	'P'
Reward	State 1			State 2			State 3		
Histogram	-4.5	-4.0	-4.5	-4.4	-4.0	-4.3	-4.6	-4.2	-4.7
Mesh	-5.0	-4.1	-5.0	-4.4	-3.9	-4.5	-4.7	-3.9	-4.6
String	-11.5	-10.6	-12.2	-10.1	-9.6	-10.8	-9.8	-9.6	-10.2
Reward	State 4			State 5			State 6		
Histogram	-5.1	-4.5	-5.5	-5.3	-6.3	-4.3	-3.8	-5.9	-5.8
Mesh	-5.8	-4.9	-5.7	-6.3	-7.3	-5.2	-6.1	-6.8	-5.7
String	-9.8	-9.6	-10.1	-9.9	-10.0	-10.2	-10.3	-10.3	-10.5

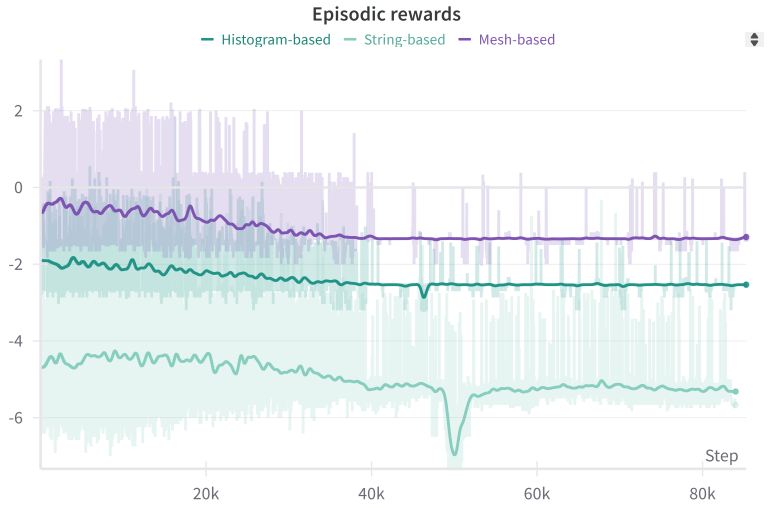


Figure 4.32: Six steps: episodic rewards

intermediate rewards obtained. The final episodic reward averages illustrated in Fig. 4.32 for the string-based method reveals a sharp decline midway through the training period. Analysis of the observation record indicates an extended phase of truncated episodes, during which the agent frequently encounters computational errors when executing the empty strip addition pair, 'AA', at the end of its six-step action sequence. Consequently, these episode fail to conclude successfully, resulting in higher averages of negative rewards and a subsequent dip in performance. The final converged action strings yield the following outcomes:

- Histogram: 'TTTTPP', 'TTTPPA', and 'TPPPPP'; -2.53
- Mesh: 'TTTTPP' and 'TTTTPA'; -1.29
- String: 'ATTTTT' and 'PTTTTT'; -5.32

Similar to the five step scenario, the q-values for the first four states show a preference by the learning agent for action 'T'. However, in states five and six, the q-values across all reward structures begin to indicate alternative opportunities for action selection. The sub-optimal meshes discovered for the action string 'ATAATA' are depicted in Fig. 4.33. Qualitatively, the action strings 'ATTTTA' and 'TPATTA' results in modifications to the mesh layout that vary significantly, leading to notable visual discrepancies. The addition of two faces into the topology matrix in the former case can influence the form-finding process, which will be explored further in Section 4.3.

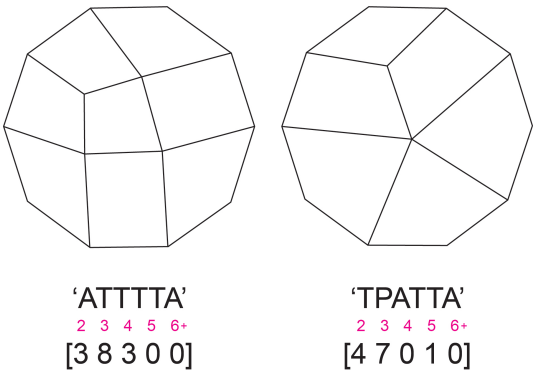


Figure 4.33: Step six: alternative mesh

4.2.3 ’ATPTTPTA’

In the final eight-step sequence, the selection of action ’P’ within the histogram-based reward structure yields competitive results compared to the favorability of action ’T’ as shown in Fig. 4.34. Although the mean selection of action ’A’ selection for the string-based method is comparable to that of action ’P’, the pronounced differences observed during the six-step sequence are no longer evident.

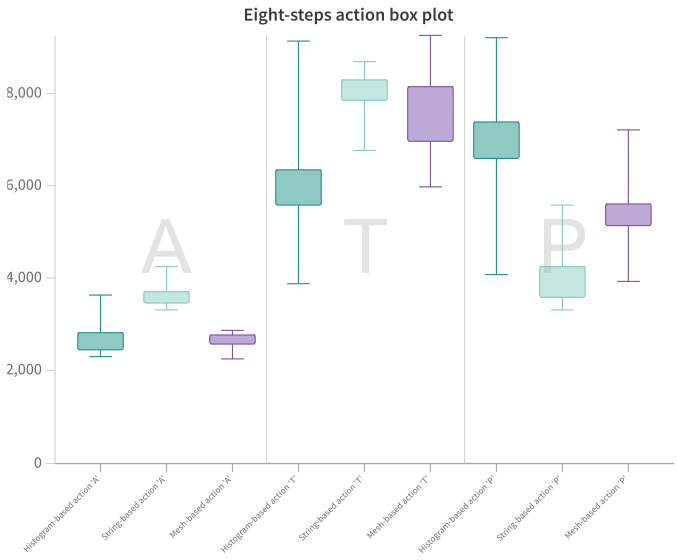


Figure 4.34: Eight steps: action box plot

The q-values presented in Table 4.9 indicate a preference for action 'T' up to state 4, after which a more varied selection of actions is observed across the three reward structures. Beyond six steps, the episodic reward averages demonstrate that as the design episodes become longer, the significant positive sparse rewards distributed at the final state becomes increasingly diluted. This phenomenon is believed to cause the range of all q-values across the eight states to be minimal when the policy converges to an inconsequential action. The penalty for failing to modify the mesh layout is outweighed by the accumulation of intermediate rewards and penalties, which are primarily influenced by the initial inputs and the respective distance values defined by each reward metric.

Table 4.9: Step eight: final q-values for initial, intermediate, and final states

	'A'	'T'	'P'	'A'	'T'	'P'	'A'	'T'	'P'	'A'	'T'	'P'
Reward	State 1			State 2			State 3			State 4		
Histogram	-10.1	-9.5	-9.5	-10.0	-9.1	-9.4	-9.7	-8.9	-9.7	-9.4	- 8.7	-8.9
Mesh	-5.5	-4.7	-5.4	-5.2	-4.6	-5.2	-5.05	-4.4	-5.1	-4.8	-4.2	-5.0
String	-21.2	-19.5	-20.0	-18.6	-17.2	-17.8	-16.3	-15.2	-15.5	-14.9	-14.0	-14.4
Reward	State 5			State 6			State 7			State 8		
Histogram	-9.1	-8.7	-8.7	-9.5	-9.1	-9.0	-9.6	-9.2	-9.1	-9.6	-9.2	-9.2
Mesh	-4.4	-4.2	-4.0	-4.1	-3.8	-4.2	-4.5	-4.9	-4.1	-4.3	-4.2	-4.0
String	-14.2	-13.9	-14.0	-14.1	-14.2	-13.8	-14.2	-14.4	-13.9	-14.5	-14.2	-14.0

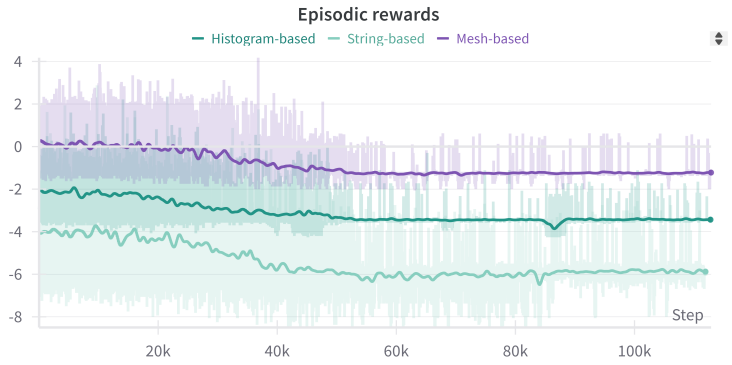


Figure 4.35: Eight steps: episodic rewards

For the string-based approach, as long as the learning agent develops a policy that includes a moderate combination of actions 'T' and 'P' within its action sequence regardless of whether a successful strip addition has occurred or not the string-distance will be minimized. The mesh-based method similarly converges

toward the time-penalty, while the histogram-based method converges toward the mean squared error value of 2.0 between the initial and target state. The final converged action strings and episodic reward averages, depicted in Fig. 4.35, give the following results:

- Histogram: 'TTTTTTTT', 'ATTTTTTT'; -2.53
- Mesh: 'TTTTTTPA', 'TTATTTTT', and 'TPPPPPPT'; -1.29
- String: 'TTTTTPPP', 'TPTPTPPT', and 'TPTPTPTP'; -5.32

As the string-length increases, the combinatorial nature of the quad-mesh grammar becomes amplified, enabling the discovery of alternative sub-optimal meshes that exhibit both qualitative and quantitative (histogram) diversity, as shown in Fig. 4.36. Additionally, two distinct isomorphs, Fig. 4.37, of the same string lengths were identified during the exploration phases of training, providing first signs of success in generating novel mesh layouts. However, it is also observed that in the eight-step scenario, the number of computational errors encountered by the learning agent rises drastically, hindering the exploration process due to interruptions in the algorithm. While the combinatorial nature of the quad-mesh grammar aids in the exploration of a broader range of possibilities, it simultaneously increases the proportion of inconsequential action strings.

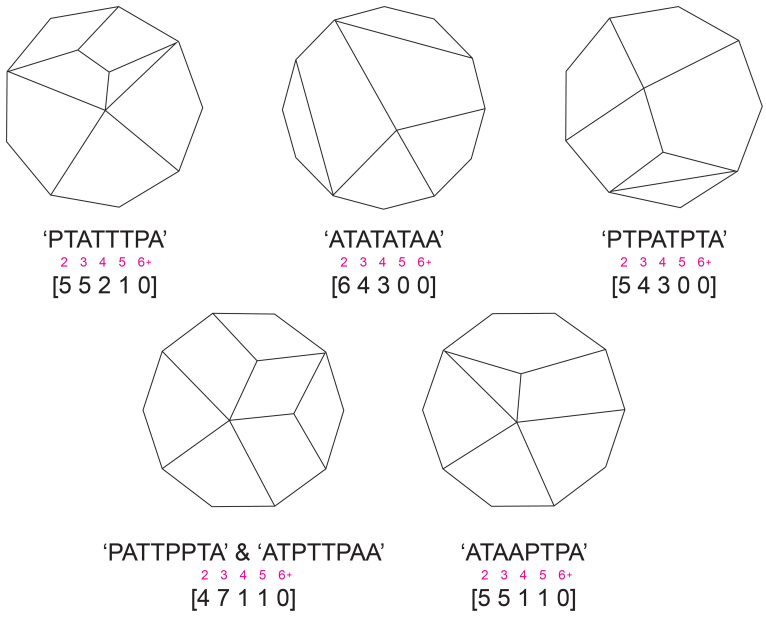


Figure 4.36: Step eight: alternative mesh

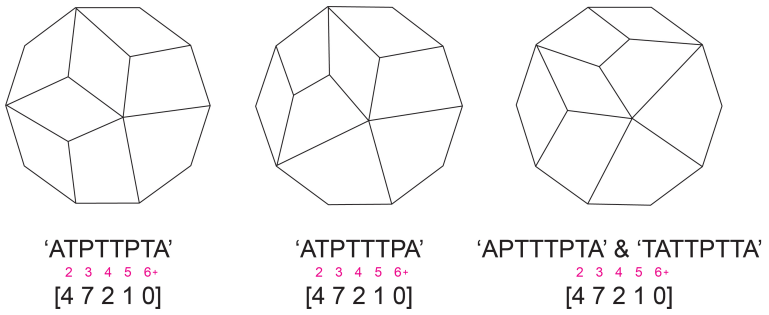


Figure 4.37: ATPTTPTA alternatives

4.2.4 HYBRID REWARD FUNCTION

The numerical experiments conducted for all sequential exploration scenarios indicate that using a single distance metric that captures one aspect of the design task is insufficient for assisting the learning agent in navigating the design space effectively. A notable issue with both the histogram- and mesh-based distance metrics is that they are shaped rewards, only distributed after the strip addition sequence is completed, specifically when the second 'A' action is executed. This results in the agent frequently selecting action 'A' , however, sequences such as 'AA' which lead to computational errors, complicate the learning process and understanding of the consequences of each action. The hierarchy of actions required to effectively utilize the quad-mesh grammar, combined with computational limitations, presents significant challenges in learning an optimal policy. By combining the different distance metrics, the objective is to provide the agent with multiple branches of knowledge. Nonetheless, as the hybrid function aggregates all values, understanding how changes in reward signals affect the learning process becomes increasingly difficult with each additional layer of information.

Following the analysis of generalized terminal states in the five-, six-, and eight-step scenarios, four combination schemes have been developed to integrate the established reward function mechanisms. These schemes are structures as follows:

Table 4.10: Hybrid reward function schemes

	Histogram	Mesh	String
Hybrid 1		✓	✓
Hybrid 2	✓	✓	
Hybrid 3	✓		✓
Hybrid 4	✓	✓	✓

These hybrid reward structures will be applied to the eight-step scenario to facilitate a comparison with the results from the previous section. To address the issue of sparse reward dilution, the magnitude of positive sparse rewards for altering the mesh topology and matching the target state has been significantly increased to ensure that the average episodic rewards remain positive. For hybrid methods 1 to 3, when the mesh topology is successfully modified, a sparse reward of 30 is distributed, from which negative distance metrics are subtracted to better reward action sequences in close proximity to the target. Additionally, upon reaching the target state, a larger sparse reward of 100 is applied. For hybrid method 4, both rewards are further increased with a reward of 80 distributed for successfully modifying the mesh topology and a sparse reward of 200 when the target state is reached. These adjustments aim to investigate the impact of enhanced sparse rewards on improving the average episodic rewards, as these rewards are typically only obtained at the terminal state.

In Fig. 4.38, reward structure Hybrid 4 shows the most consistent results with minimal variance between the lower and upper quartiles for all three actions among its 10 runs.

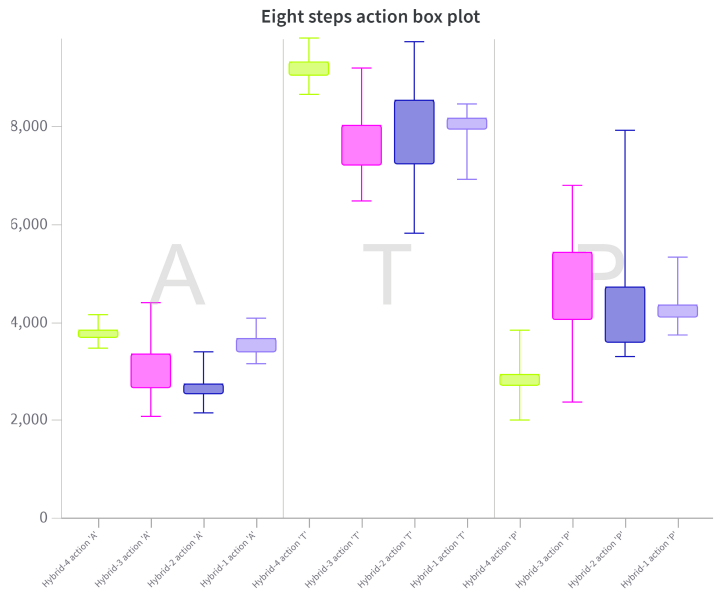


Figure 4.38: Hybrid eight steps: action box plot

The final converged action strings and episodic reward averages, depicted in Fig. 4.39, give the following results:

- Hybrid 1: 'TTPTPTPP', 'TPTPTPTA'; -5.91

Table 4.11: Hybrid step eight: Mean and standard deviation of average action history

(μ, σ)	'A'	'T'	'P'
Hybrid 1	3582., 263.9	7996., 436.5	4421., 505.0
Hybrid 2	2736., 349.3	8213., 1502.	5051., 1633.
Hybrid 3	3243., 762.3	7811., 801.0	4946., 1368.
Hybrid 4	3820., 228.4	9281., 349.3	2898., 524.7

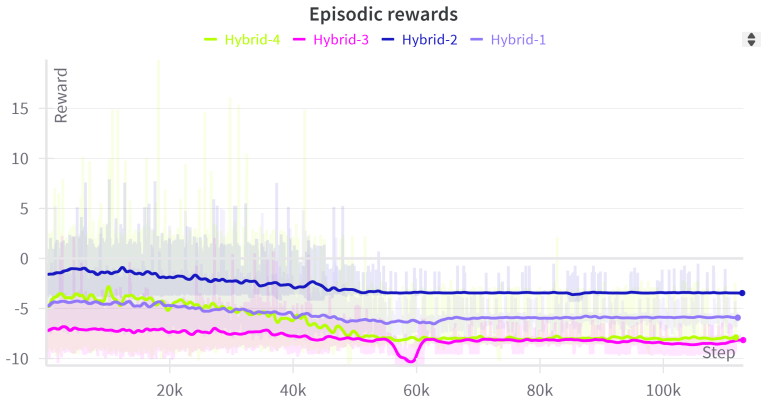


Figure 4.39: Hybrid eight steps: episodic rewards

- Hybrid 2: 'TTTTPTPP', 'TTTTTPPT'; -3.45
- Hybrid 3: 'PTPTPTTP', 'PPPPPPPP', and 'PTTTTPPP'; -8.42
- Hybrid 4: 'ATTTTTTT', 'TTTPPPPP'; -7.93

The following Table 4.12 summarizes the statistics of the successful design episodes during training of the scenarios 5 through 8 using the independent reward function metrics, and the same 8th scenario using the hybrid reward function structures. Success rate is defined as the number of attempts the learning agent was able to successfully modify the topology of the mesh layout, no matter the insignificance of the magnitude at which the change is brought forth.

Results from the eight-step investigation indicate that the two best-performing reward structures are the string-based method and Hybrid 4, which incorporates all three distance metrics. The success rate of all three independent reward mechanisms also increased proportionally with the number of time steps per design episode. However, the performance of Hybrid 1-3 reward structures did

not show an improvement in the final episodic rewards or in the convergence of action strings.

Table 4.12: Training results for steps 5+

Reward	Median	Mean	Std.	Success
5 steps: 'ATPTA'				
Histogram	92.5	92.7	13.99	5%
Mesh	93	93.6	9.28	5%
String	112.5	111.7	11.58	6%
6 steps: 'ATAATA'				
Histogram	115.5	116.8	11.32	6%
Mesh	128	121.5	17.15	6%
String	170	168.9	11.85	8%
8 steps: 'ATPTTPTA'				
Histogram	141.5	138.4	22.08	7%
Mesh	155.5	157.1	13.52	8%
String	237.5	236.7	16.86	12%
Hybrid 1	164.5	161.7	21.28	8%
Hybrid 2	194	188.9	30.52	9%
Hybrid 3	163.5	171.6	22.16	9%
Hybrid 4	190.5	192.6	11.98	10%

4.3 COMPATABILITY IN A DESIGN WORKFLOW

This final section will demonstrate how the generated meshes from the RL algorithm can be post-processed via densification and form-finding, as illustrated in Fig. 4.40. The forced density method will be applied for form-finding, upon which further illustrations and explanations will be included to showcase potential applicability of the developed model as well as the intended workflow.

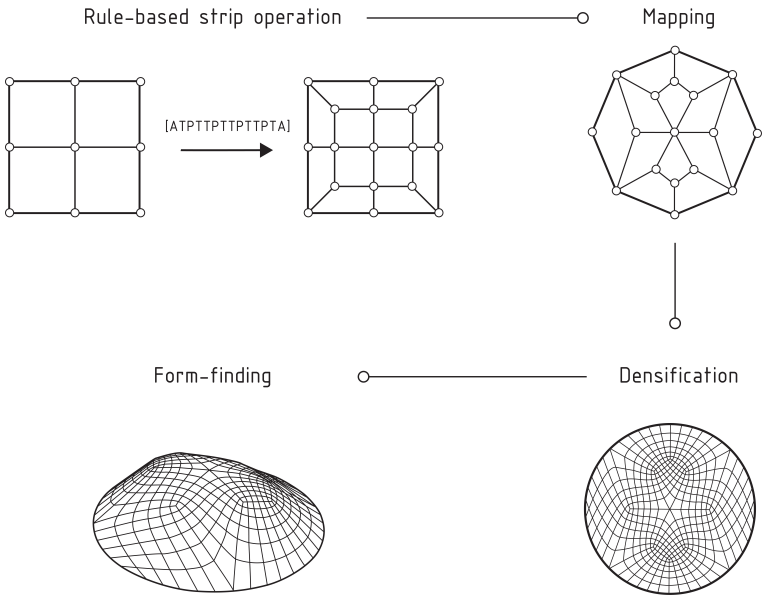


Figure 4.40: Proposed structural workflow

Although current stages of development of the reinforcement learning framework excludes the post processing proceeding application of the rule-based strip operation, idealistically, geometrical and structural factors should be implemented to impart better navigation capabilities to the learning agent. However, the investigation of an extended string operation, 'ATPTTPTTPTTPTA', has revealed that further computational limitations of the compas_quad algorithm exists between the output of the RL model and the post-processor. Strip addition sequnces that includes 'TP' with no additional 'T' after the 'P' would incur KeyErrors when mapping the topology of the coarse quad-mesh to a circular boundary. Furthermore TypeError would also occur when a NoneType object is detected in the data structure of the mesh matrix. Therefore, not all successful mesh modification sequences were able to be utilized during post processing.

Figures 4.41 and 4.42 illustrates form-found results from the developed RL model to a 14-character action string, showcasing the model's capability to generate structured design outputs. Form 20 represents the ground truth string

of 'ATPTTPTTPTTPTA'. However, the final evaluation of compatibility within the design workflow revealed that the alternative design options produced by the model were relatively unremarkable. Furthermore, the workflow in its current state closely resembles a purely randomized approach to mesh generation using the quad-mesh grammar, a functionality already available within the COMPAS framework. This highlights the need for future efforts to integrate domain-specific knowledge into the Markovian framework, enabling the grammar to realize its full potential for informed and purposeful design exploration.

Table4.13 lists the total strain energy of the system obtained by multiplying the respective length and corresponding axial force of the mesh system.

$$\sum_i l_i * p_i \tag{4.2}$$

The summed value represents the work done by the internal forces in the structure, providing insight into how forces are distributed and how efficiently the structure uses material to resist loads. The applied load is a uniformly distributed vertical load applied equally to all vertices in the system, with the total load summing to 3 in the z-direction. The listed values have been normalized to total strain energy of the ground truth form.

Table 4.13: Total strain energy of generated forms

Form	Normalized strain	Form	Normalized strain
1	0.93	11	0.95
2	0.95	12	0.99
3	0.94	13	1.02
4	0.90	14	0.93
5	0.89	15	1.01
6	0.86	16	0.90
7	0.99	17	0.94
8	0.92	18	0.97
9	0.91	19	0.92
10	1.03	20	1.00

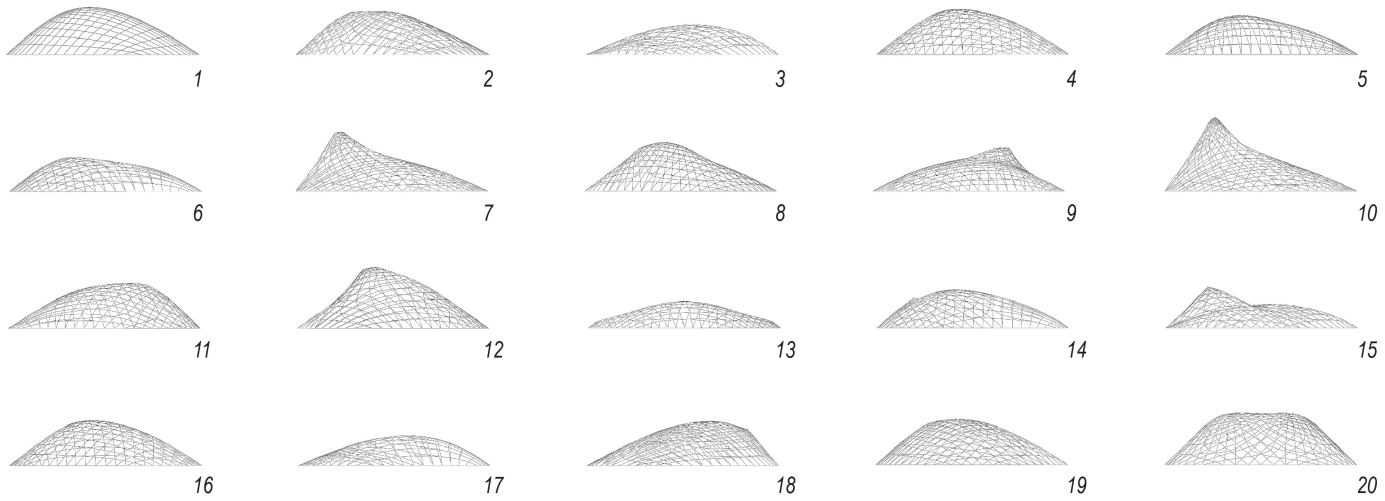


Figure 4.41: Generated forms (front view)

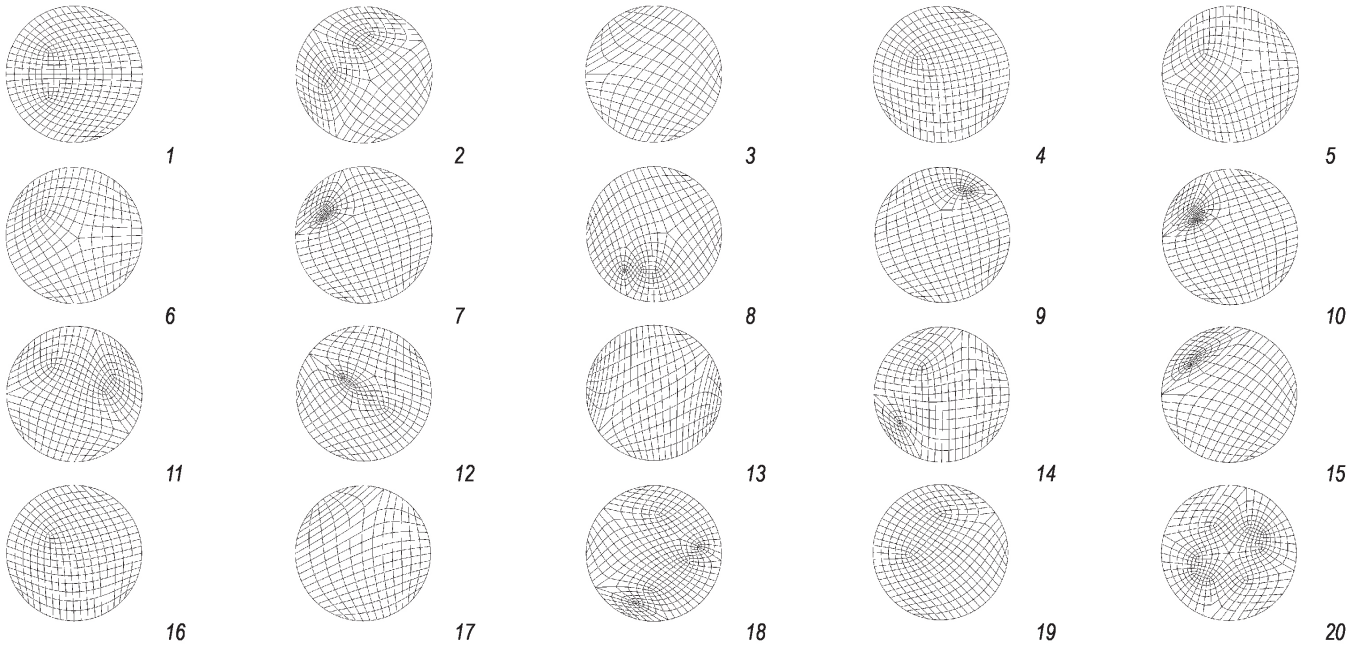


Figure 4.42: Generated forms (top view)

5 Conclusion

This research aimed to develop a computational design algorithm for exploring the design space of quad-mesh patterns for shell-like structures. The methodology combined grammar-based generative approaches, which define the design space, with reinforcement learning to facilitate effective navigation and optimization. This work contributes to bridging the gap between generative and navigational approaches in structural design, offering a foundation for more sophisticated exploration of quad-mesh structures.

5.1 RESEARCH QUESTIONS

How can reinforcement learning be effectively integrated with quad-mesh grammar and what method can be applied to explore diverse structural patterns?

For the integration of quad-mesh grammar into a Markovian decision framework, the observation space was constructed using topological performance metrics and the position of the lizard to provide feedback on how actions within the grammar's design space affect outcomes in the quad-mesh design space. Discrete actions were translated into corresponding characters that defined the movement and modification commands executed to alter the topology of a given mesh layout. Reward functions were designed using only topological information to ensure that the learning agent could effectively use the given actions to execute sequences leading to meaningful outcomes, while maintaining sufficient flexibility to prevent over fitting to specific design tasks. Hyperparameters were adjusted to facilitate the convergence of the DQN model. The training frequency was calibrated to update the online Q-network at optimal intervals, thereby preventing over fitting to recent transitions while maintaining a consistent learning pace. Additionally, the learning rate was set to balance stable updates and convergence speed during back propagation, enabling the agent to adapt efficiently without risking instability. A higher discount rate was applied to encourage long-term planning, essential for tasks involving delayed rewards. The extended exploration phase ensured the agent continued discover alternative actions while preventing early convergence on non-optimal strategies.

The results indicate the need for balanced exploration of the appropriate length of action sequences for the RL agent. In scenarios up to four steps, the explored reward function mechanisms (histogram-, mesh-, and string-based metrics) were insufficient to guide the agent from an initial coarse quad mesh to a target state. This limitation was partly due to the restricted ability of the quad-mesh grammar to modify the mesh topology with only four actions. The distance between the initial and target states in such short sequences was not significant enough to facilitate effective strip modifications, which require a minimum of three actions. As discussed in the analysis of the hybrid reward structures, the trend toward higher success rates at longer design episodes indicates potential for further exploration of RL performance over extended sequences. However, addressing or mitigating computational limitation is crucial, as the combinatorial nature of the grammar approach exponentially increases both mesh diversity and the occurrence of inconsequential action strings.

To enhance the integration of the quad-mesh grammar with RL, further exploration of alternative RL algorithms and techniques is necessary. On-policy algorithms such as PPO could potentially offer benefits by treating the design task as continuous and better handling the hierarchical nature of the actions. Additionally, DQN variants like Hindsight Experience Replay (HER) could be valuable for prioritizing state-action pairs that lead to significant sparse rewards, addressing the tendency of the agent to minimize negative penalties rather than optimize for positive outcomes. The current model was constrained by the use of pre-built RL algorithms designed for general applications; further customization of the computational architecture specific to the design task could significantly enhance performance.

The integration of the compas_quad library with RL algorithm presented here is still incomplete. While the study's use of purely topological and grammatical information in quad-meshes is a step toward a more refined model, uncertainty remains regarding strategies to further optimize the learning process. Potential improvements include incorporating structural and geometric parameters, such as structural performance calculations of the output mesh based on specific boundary and loading conditions. Implementing a surrogate model could alleviate the computational burden associated with complex operations like FEA. Additionally, the current simplified setup excluded the deletion maneuver ('D'), which could significantly enhance the agent's ability to modify the mesh through strip adjustments. The impact of incorporating this additional character on overall model performance remains an open question.

The complexity of integrating quad-mesh grammar with an RL algorithm underscores the intricacies involved in the design process and deepens the understanding of both systems. Applying RL to design tasks poses the unique challenge of guiding the learning agent to perform tasks in alignment with

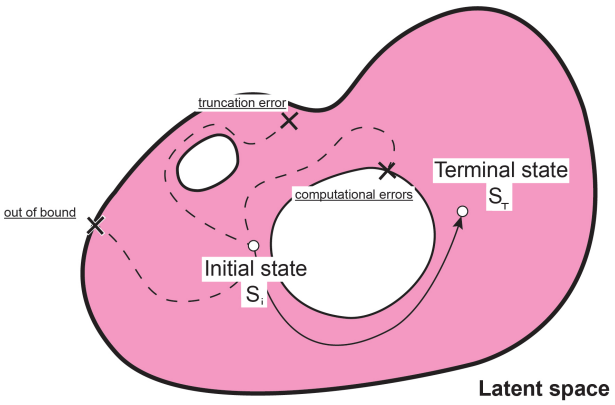


Figure 5.1: Revised design space representation

user expectations. Unlike human designers, learning agents lack preconceived notions of how input data should be utilized without explicit guidance. This dissonance between human and artificial intelligence fosters innovative solutions, allowing the agent to 'break' conventional expectations and leverage partial knowledge in novel ways. Similar to how a child might interact creatively with a new toy, the learning agent's exploratory behavior can subvert the creator's assumptions, providing insights into alternative methods for achieving design goals. Through the examination of the design space via RL, a deeper understanding can be gained of how that space is constructed and how to navigate it effectively, as illustrated in Fig. 5 .1. Without heuristic trial and error, the limitations and opportunities of the quad-mesh grammar might remain unexplored.

Q_1: How can the quad-mesh grammar be used to define the various components that formulate a Markov decision process that promotes the development of an optimal policy?

Summary (Sub-question 1)

The observation space was constructed using topological metrics of the mesh and positional data of the lizard, while discrete actions were translated to string-actions to enable effective navigation and modification capabilities for the learning agent. Various reward functions were defined and attempted to promote successful learning of an optimal policy.

Q_2: What training strategies and parameters are effective for integrating quad-mesh grammar with reinforcement learning and how will their effectiveness be assessed?

Summary (Sub-question 2)

Constraint identification of both the compas_quad and reinforcement learning algorithms lead to the development of three general distance metrics that were used to describe the rewards. Hybrid combinations of the histogram-, mesh-, and string-based mechanisms were addressed to examine the most representative information that facilitated optimal learning. Sweeps were conducted for every experimental set up to ensure model convergence and appropriate conditions for the RL agent to arrive to the desired outcome.

Q_3: Can a generalized computational model be constructed that encourages greater diversity in the solutions generated?

Summary (Sub-question 3)

The integration of quad-mesh grammar and RL facilitated the discovery of suboptimal yet diverse mesh configurations, although incorporating additional features such as deletion maneuvers and structural performance metrics could further enrich solution diversity.

5.2 RECOMMENDATION FOR FUTURE WORK

Future research should address the challenges and limitations identified in this study to further enhance the integration of quad-mesh grammar with reinforcement learning (RL) algorithms. One critical improvement is the incorporation of context-sensitive operations within the grammar. While the current framework effectively encodes and navigates the design space, the lack of contextual awareness limits its applicability to more complex scenarios. Adding such operations, as acknowledged in prior research, could enable more targeted exploration and reduce the likelihood of computational errors caused by the combinatorial nature of the design problem, as shown in Fig. 5.2. Moreover, implementing robust error-handling measures during grammar execution would further improve reliability and scalability.

Another key consideration is the refinement of evaluation criteria for generated meshes. While the overarching goal is to produce diverse and high-performing designs, the definition of what constitutes an optimal mesh remains ambiguous. By incorporating specific case studies involving certain load com-

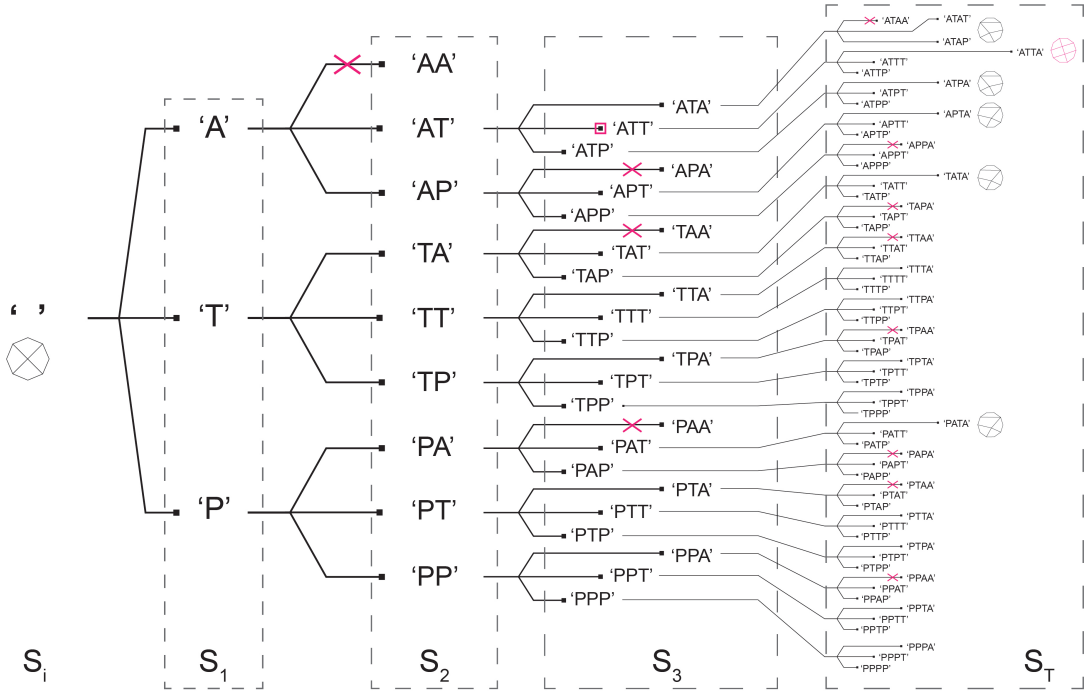


Figure 5.2: Combinatorial explosion

binations and boundary conditions, key geometrical features and structural outputs from FEM analyses can be integrated into the reward signals to provide valuable insights, as shown in Fig. 5.3. However, before integrating such complex parameters, this research has illustrated the importance of establishing a clear balance between maintaining flexible objectives that retain exploratory capacity and incorporating context-specific parameters that aid in optimizing explored solutions. Furthermore, as the application of post-processing operations has revealed, incorporating additional layers of computational management is required to mitigate errors at various points in the workflow. Addressing these challenges will necessitate careful resource optimization and more efficient handling of data structures.

The current framework has demonstrated the potential of combining Markovian structures with quad-mesh grammar for structural design exploration, but further advancements could be achieved by incorporating alternative RL algorithms and techniques. Notably, RL algorithms such as DQN have been successfully applied to a wide range of games, from side-scrolling shooters to three-dimensional car racing simulations. These applications highlight DQN's ability to discover relatively long-term strategies in certain scenarios. However, tasks requiring more temporally extended planning continue to challenge existing RL agents, including DQN [5]. Methods such as Proximal Policy Optimization (PPO) and Hindsight Experience Replay (HER) offer promising avenues for enhancing learning efficiency and adaptability. Additionally, insights from RL applications

[5] Billington et al. 2015. Human-level control through deep reinforcement learning.

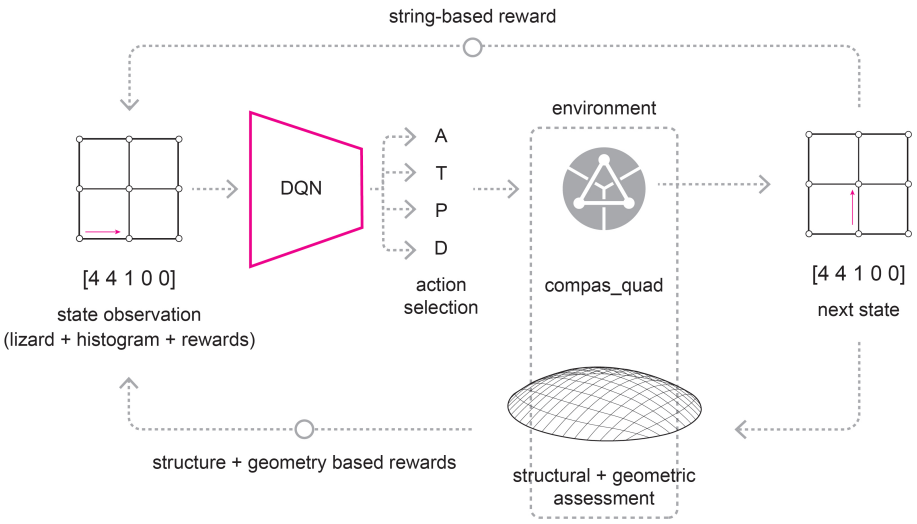


Figure 5.3: Idealized computational workflow

in other fields could inform strategies for optimizing agent performance in this complex design space. The implementation of RL concepts in this project relied on open-source libraries and pre-built generalized classes, which facilitated initial implementation but inherently limited customization. Adapting these algorithms to better align with the specific requirements of the quad-mesh grammar would significantly improve performance. Tailored modifications to RL architectures, including adjustments to state representation, action selection, and reward structures, would likely enhance the agent's capacity to navigate the intricate topological design space more effectively.

Ultimately, advancing the compatibility of the RL framework with broader design workflows requires iterative improvements to both the grammar and RL processes. By refining the criteria for mesh quality, enhancing the integration of contextual operations, and dedicating time and patience to model development, the full potential of the quad-mesh grammar for topological exploration can be unlocked.

5.3 AFTERWORD: APPROACHING DESIGN

Civil engineering, at its core, is more than just laying down concrete and steel – it embodies a deeper ethical commitment to understanding and evaluating complex systems in their entirety. As Fuller described, the behavior of a whole system cannot be predicted by analyzing its individual components alone. This holistic perspective is crucial for civil engineers, who must address the breadth, depth, and coherence of principles and values within the design process. Identifying and

framing a design problem requires a thoughtful evaluation of specific needs and objectives, balancing factors like budget, time, and materials, and the interplay between functionality, aesthetics, and structural integrity. The ability to thread through these constraints depends fundamentally on the depth of engineering judgment and intuition, where experience and insight guide critical decisions – yet, with the advent of machine learning, these human faculties can be further enhanced, preparing us for the complexities of future engineering endeavors.

Just as civil engineering draws upon a deep understanding of complex systems, the development of AI, particularly in DL, demands not only technical expertise but also a nuanced consideration of philosophical principles. The nature-nurture dichotomy often emerges in debates between empiricism and nativism, reflecting the tension between knowledge derived from sensory experience and knowledge based on inherent reasoning structures. DL is typically aligned with an empiricist perspective given its dependence on vast datasets and computational power to model complex behaviors. However, the abstraction required in designing these methods – such as the need for logical frameworks and well defined problems – suggests that DL also incorporates nativist elements, especially in understanding and embedding rationality within AI systems [8]. This intersection is evident in the work of interdisciplinary researchers like Richard Evans from DeepMind, who integrate philosophical principles from Kant's *Critique of Pure Reason*, to augment DL systems with logical reasoning, aiming to balance these two perspectives [11]. The philosophical discourse surrounding DL reveals that while numerical approaches have driven significant advancements, they are not wholly sufficient. Critics such as Richard Sutton, in his *Bitter Lesson*, underscores that an over-reliance on computation may neglect the importance of domain-specific knowledge and abstraction, which are crucial for achieving generalization and human-like intelligence.

[8] Buckner. 2024. *From Deep Learning to Rational Machines.*

[11] Evans. 2020. *Kant's Cognitive Architecture.*

Building on the concinnity between civil engineering's ethical grounding and the philosophical dimensions of AI, it becomes clear that true innovation lies at the intersection of these fields. In analyzing Evan's work, Buckner highlights the risks inherent in interdisciplinary projects, particularly those merging AI and philosophy, warning of "falling between two stools" – neither staying true to philosophical intentions nor contributing meaningfully to AI research. However, Buckner suggests that the solution lies in "discarding the stool altogether" to create "a more accommodating bench," where a diversity of backgrounds and expertise can thrive. This analogy extends to the relationship between structural engineering and architecture, as well as between engineering and AI, emphasizing the urgency to fostering collaborative environments and communities that embrace diversity to advance design methodologies for the built environment. By working together, thinkers and machine can unlock new perspectives

and innovations, making structural design both more dynamic and ethically grounded.

While generating truly novel designs remains challenging, researchers are persistently exploring ways to restructure algorithms, allowing them to navigate design spaces that are not overly constrained. Though the integration of artificial intuition and creativity into structural engineering is still in its early stages, these technologies are not meant to replace the essential human elements; rather, they enhance them, enabling designers to navigate the complexities of modern engineering challenges with greater precision and imagination. As AI pushes the boundaries of what is possible, it will be crucial to ensure that human values and ethics remain central to the design process. By embracing the complexity of whole systems and the role of human creativity, structural engineering can evolve toward more resilient and efficient designs that also align with broader societal values.

6

Appendix

6.1 SWEEP RESULTS FOR HISTOGRAM METHOD

Table 6.1: Selected DQN hyperparameters for histogram-distance method

Hyperparameter	B	L.R.	T.U.I.	T.F.
Step 1 (Forward)	5000	0.01	8000	5
Step 1 (Backward)	5000	0.01	8000	3
Step 2 (Forward)	1000	0.005	5000	5
Step 2 (Backward)	1000	0.01	5500	5
Step 3 (Forward)	1000	0.001	9000	3
Step 3 (Backward)	5000	0.001	8000	5
Step 4	1000	0.0001	9000	1
Step 5	2500	0.001	4000	7
Step 6	1000	0.001	6500	6
Step 8	10,000	0.0001	6000	8

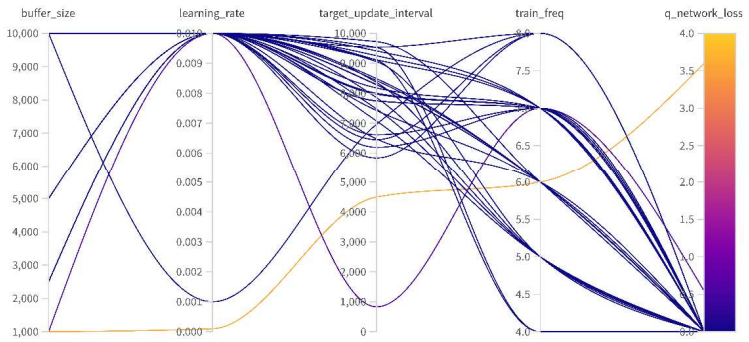


Figure 6.1: DQN histogram method forward approach 1 step

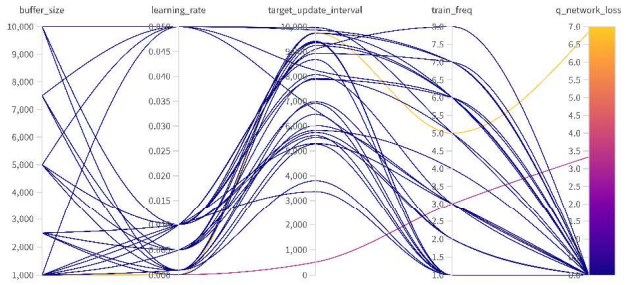


Figure 6.2: DQN histogram method backward approach 1 step

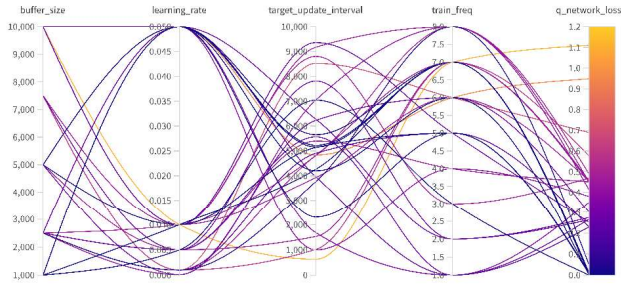


Figure 6.3: DQN histogram method forward approach 2 step

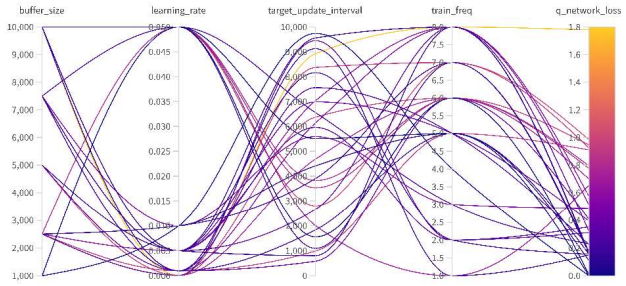


Figure 6.4: DQN histogram method backward approach 2 step

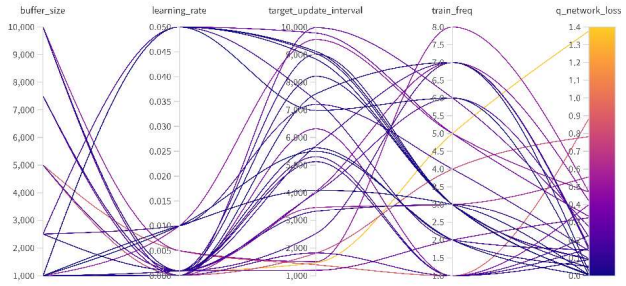


Figure 6.5: DQN histogram method forward approach 3 step

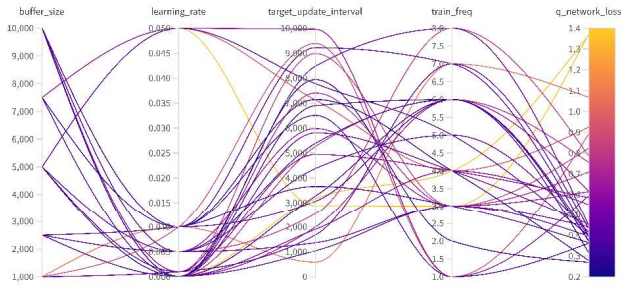


Figure 6.6: DQN histogram method backward approach 3 step

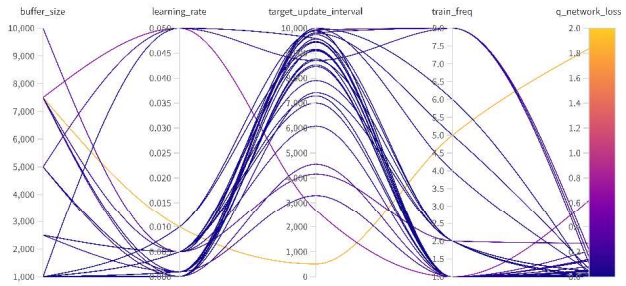


Figure 6.7: DQN histogram method 4 step sweep results

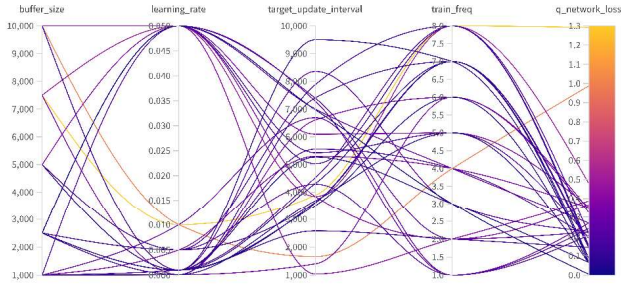


Figure 6.8: DQN histogram method 5 step sweep results

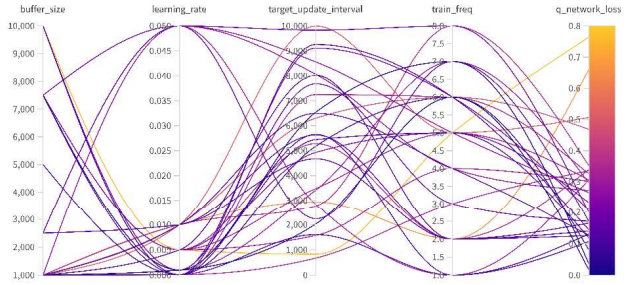


Figure 6.9: DQN histogram method 6 step sweep results

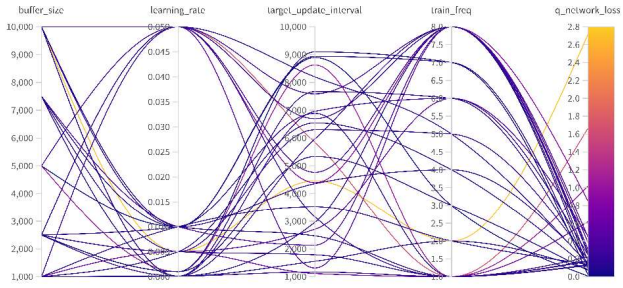


Figure 6.10: DQN histogram method 8 step sweep results

6.2 SWEEP RESULTS FOR MESH METHOD

Table 6.2: Selected DQN hyperparameters for mesh-distance method

Hyperparameter	B	L.R.	T.U.I.	T.F.
Step 1 (Forward)	10,000	0.001	8000	2
Step 1 (Backward)	10,000	0.01	8000	6
Step 2 (Forward)	10,000	0.001	6000	8
Step 2 (Backward)	5000	0.0001	8000	4
Step 3 (Forward)	7500	0.0001	8000	3
Step 3 (Backward)	5000	0.005	6000	3
Step 4	10,000	0.001	4500	3
Step 5	2500	0.001	5000	6
Step 6	2500	0.001	2500	6
Step 8	7500	0.005	3000	7

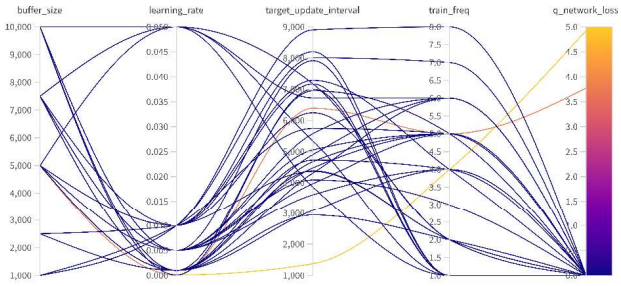


Figure 6.11: DQN mesh method forward approach 1 step sweep

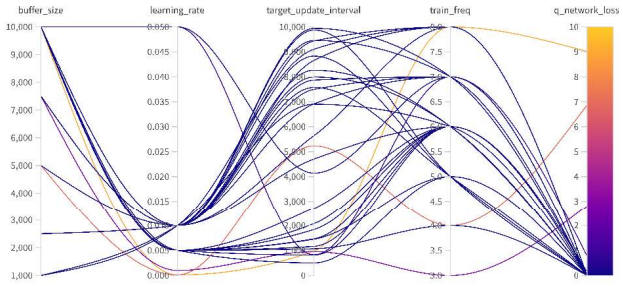


Figure 6.12: DQN mesh method backward approach 1 step sweep

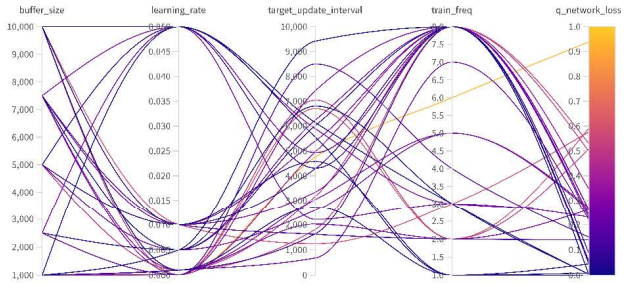


Figure 6.13: DQN mesh method forward approach 2 step sweep

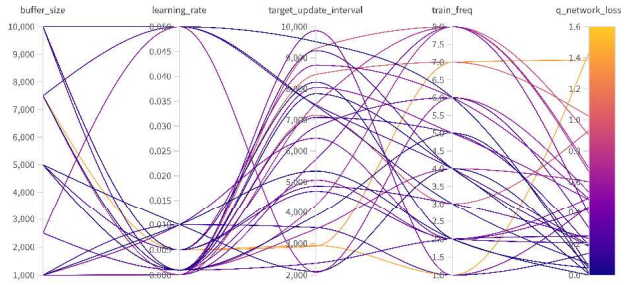


Figure 6.14: DQN mesh method backward approach 2 step sweep

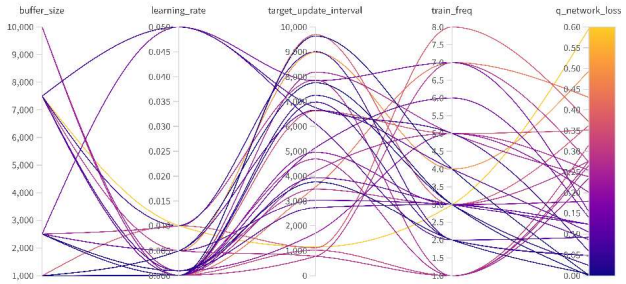


Figure 6.15: DQN mesh method forward approach 3 step sweep

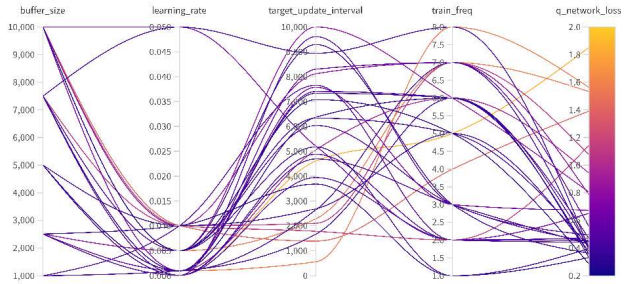


Figure 6.16: DQN mesh method backward approach 3 step sweep

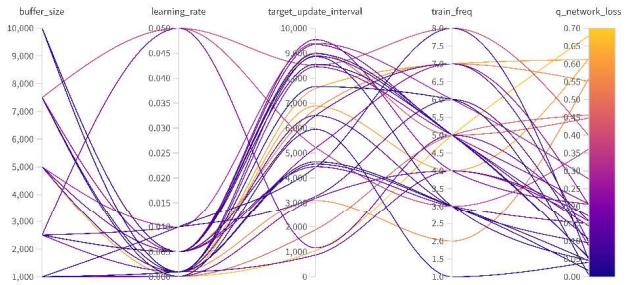


Figure 6.17: DQN mesh method 4 step sweep results

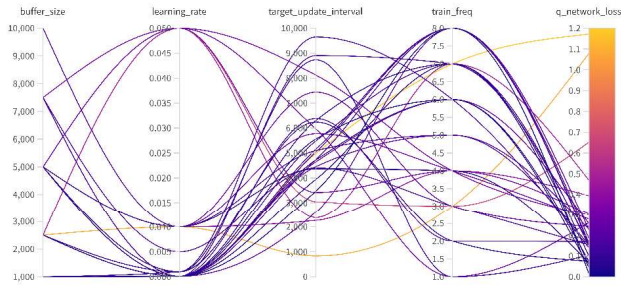


Figure 6.18: DQN mesh method 5 step sweep results

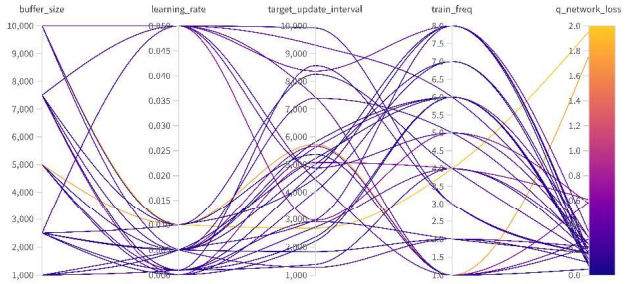


Figure 6.19: DQN mesh method 6 step sweep results

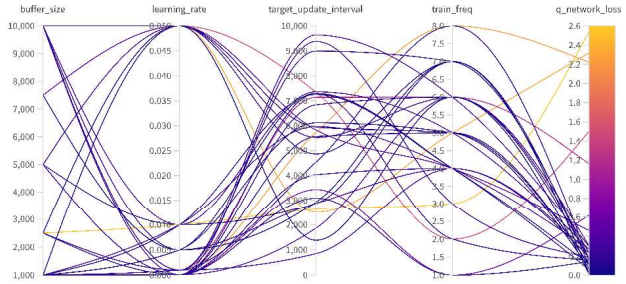


Figure 6.20: DQN mesh method 8 step sweep results

6.3 SWEEP RESULTS FOR STRING METHOD

Table 6.3: Selected DQN hyperparameters for string-distance method

Hyperparameter	B	L.R.	T.U.I.	T.F.
Step 1 (Forward)	5000	0.001	9000	1
Step 1 (Backward)	5000	0.01	6000	6
Step 2 (Forward)	1000	0.005	9000	5
Step 2 (Backward)	5000	0.005	6000	4
Step 3 (Forward)	1000	0.0001	4000	2
Step 3 (Backward)	5000	0.005	5000	6
Step 4	2500	0.0001	4000	4
Step 5	5000	0.001	5000	6
Step 6	1000	0.001	6500	6
Step 8	7500	0.005	6000	8

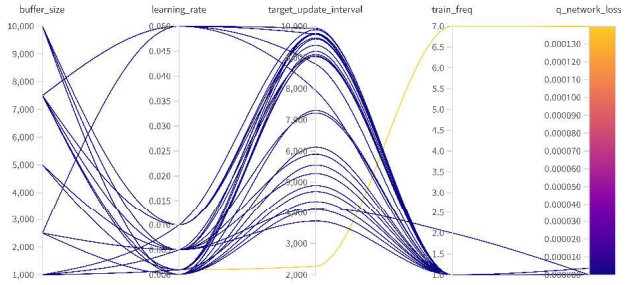


Figure 6.21: DQN string method forward approach 1 step sweep

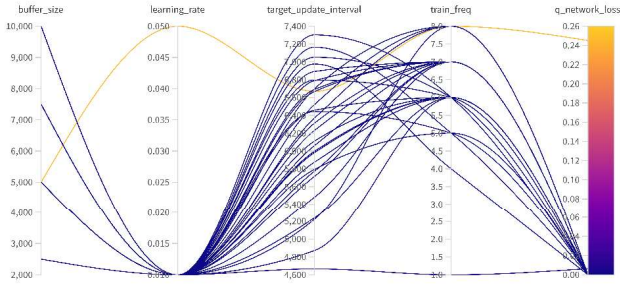


Figure 6.22: DQN string method backward approach 1 step sweep

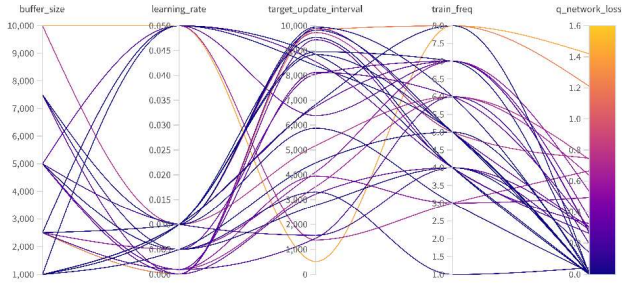


Figure 6.23: DQN string method forward approach 2 step sweep

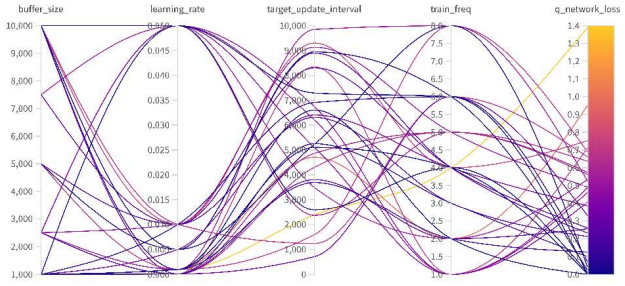


Figure 6.24: DQN string method backward approach 2 step sweep

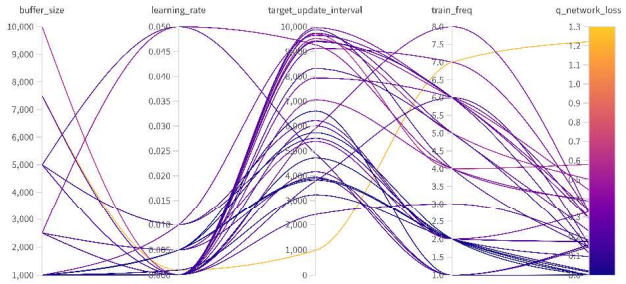


Figure 6.25: DQN string method forward approach 3 step sweep

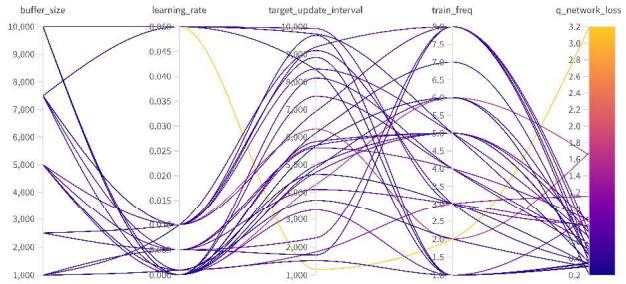


Figure 6.26: DQN string method backward approach 3 step sweep

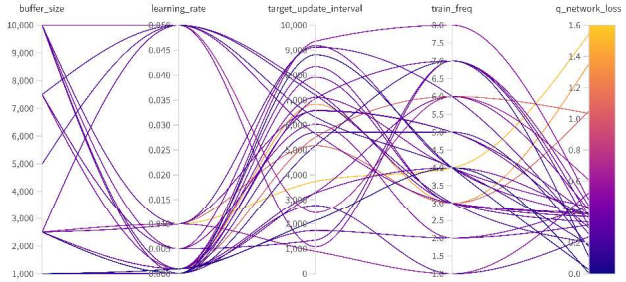


Figure 6.27: DQN string method 4 step sweep results

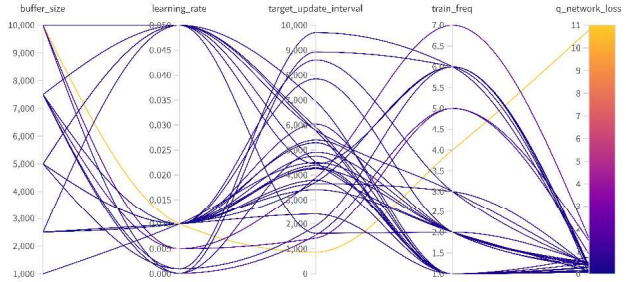


Figure 6.28: DQN string method 5 step sweep results

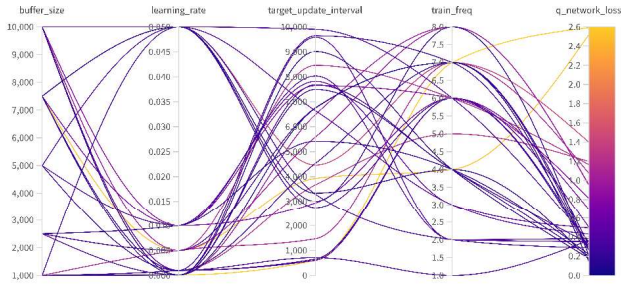


Figure 6.29: DQN string method 6 step sweep results

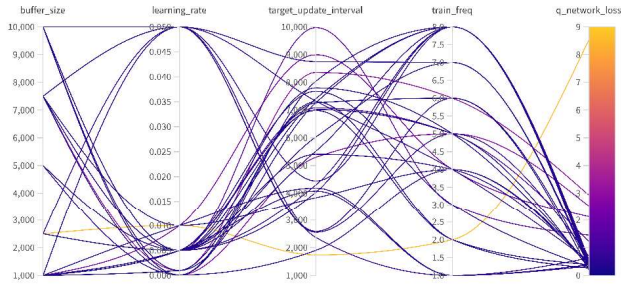


Figure 6.30: DQN string method 8 step sweep results

6.4 SWEEP RESULTS FOR HYBRID METHOD

Table 6.4: Selected DQN hyperparameters for hybrid methods

Hyperparameter	B	L.R.	T.U.I.	T.F.
1: M + S	5000	0.01	8000	5
2: M + H	5000	0.01	8000	3
3: S + H	1000	0.005	5000	5
4: S + M + H	1000	0.01	5500	5

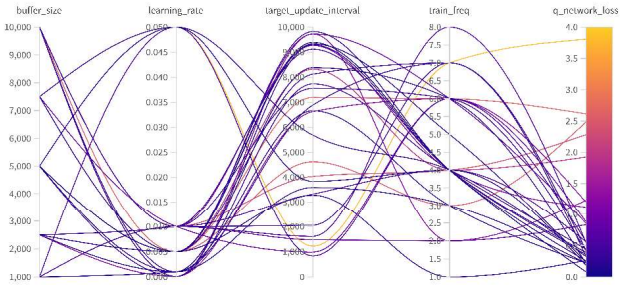


Figure 6.31: DQN hybrid method 1 sweep results

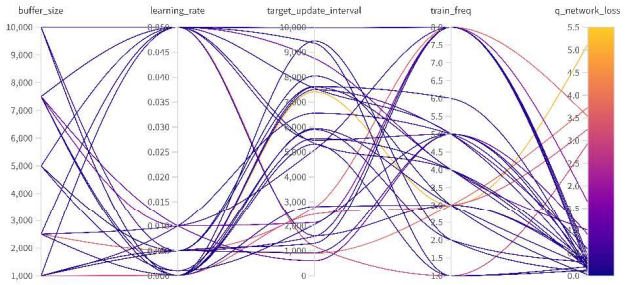


Figure 6.32: DQN hybrid method 2 sweep results

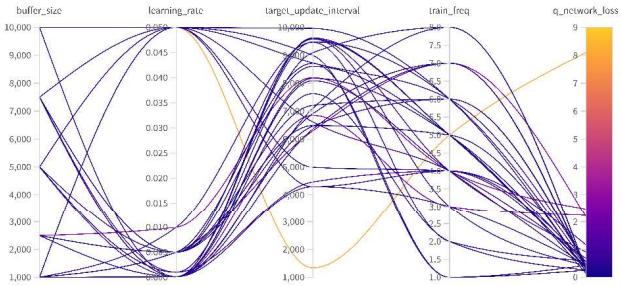


Figure 6.33: DQN hybrid method 3 sweep results

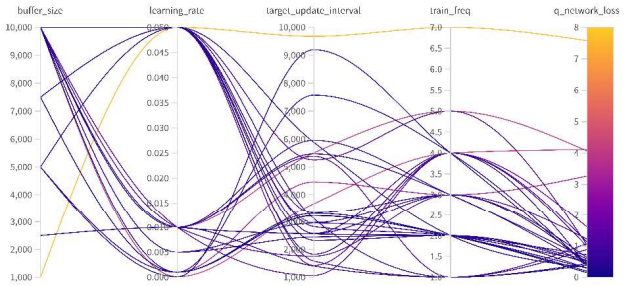


Figure 6.34: DQN hybrid method 4 sweep results

6.5 SEQUENTIAL EXPLORATION

6.5.1 ONE STEP

Figure 6.35: One-step: action box plot

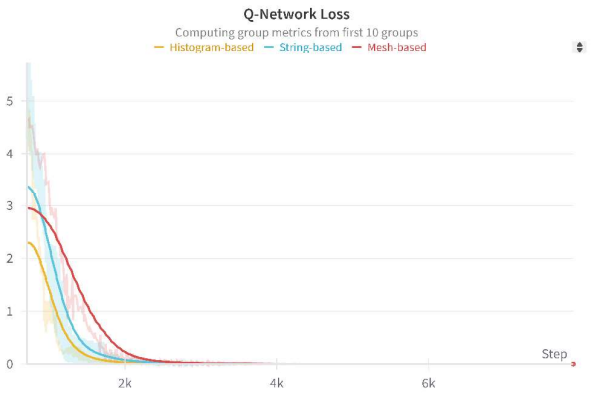


Figure 6.36: One-step: Q-Network loss

6.5.2 TWO STEPS

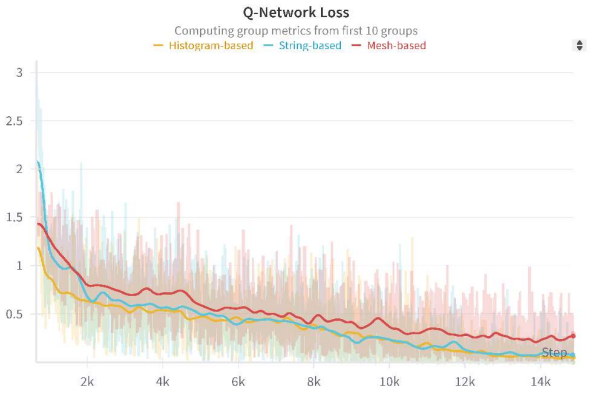


Figure 6.37: Forward two-steps: Q-Network loss

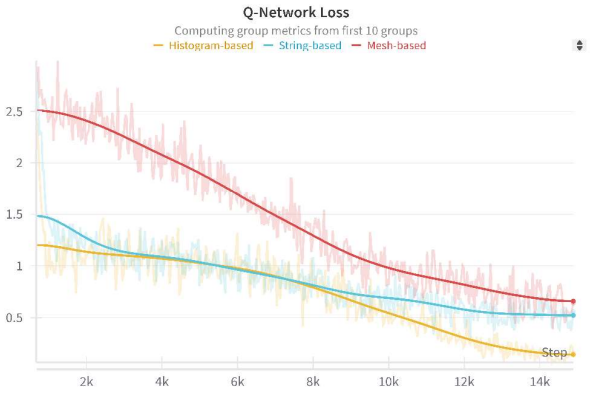


Figure 6.38: Backward two-steps: Q-network loss

6.5.3 THREE STEPS

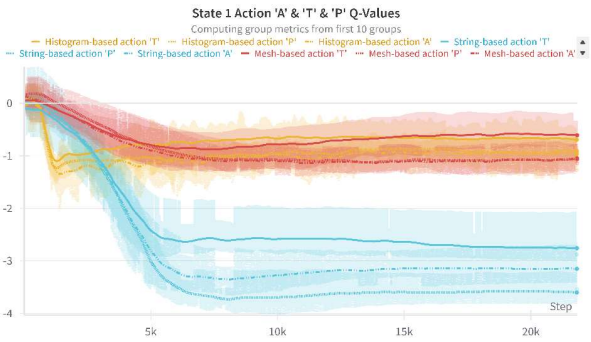


Figure 6.39: Forward three steps: state 1 action q-values

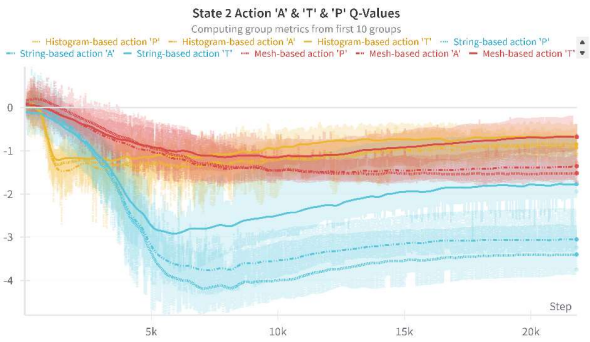


Figure 6.40: Forward three steps: state 2 action q-values

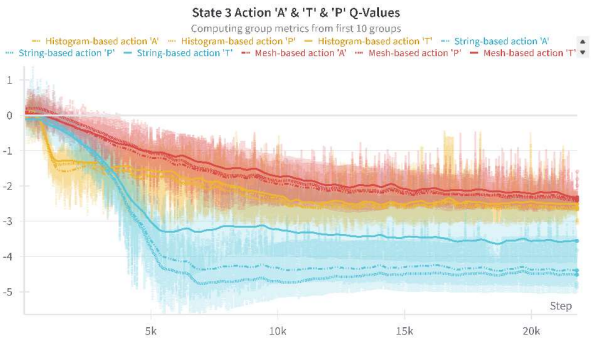


Figure 6.41: Forward three steps: state 3 action q-values

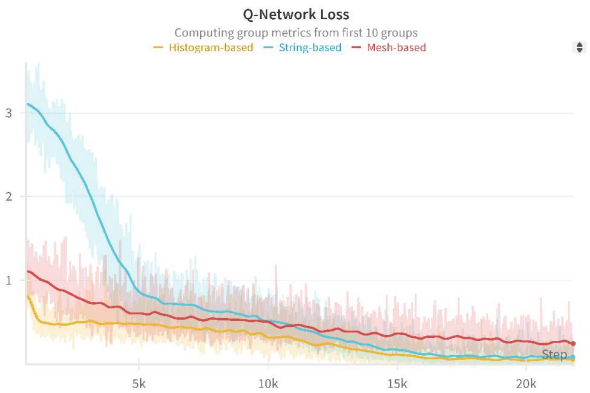


Figure 6.42: Forward three steps: Q-network loss

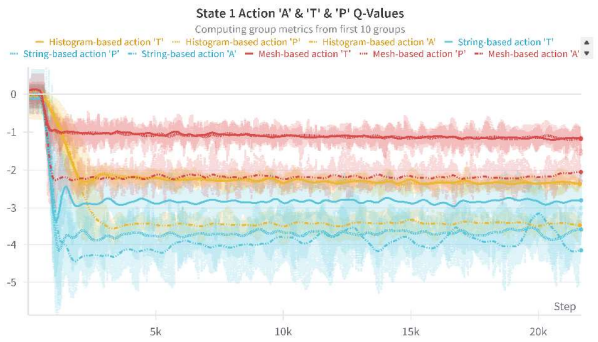


Figure 6.43: Backward three steps: state 1 action q-values

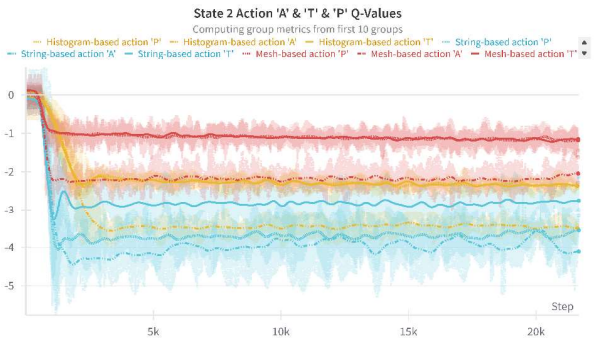


Figure 6.44: Backward three steps: state 2 action q-values

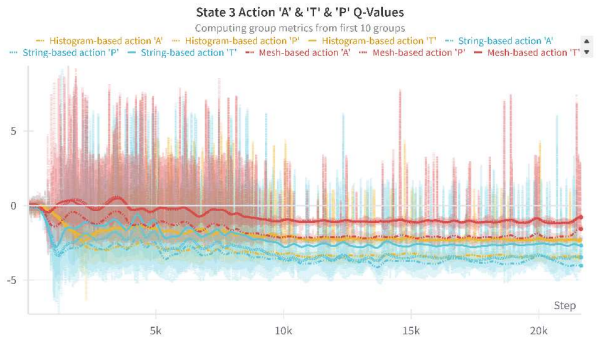


Figure 6.45: Backward three steps: state 3 action q-values

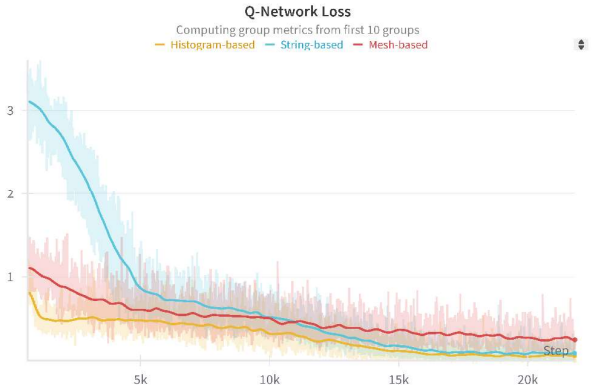


Figure 6.46: Backward three steps: Q-network loss

6.5.4 FOUR STEPS

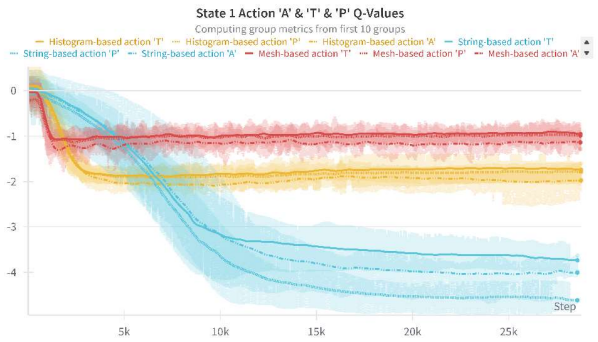


Figure 6.47: Four steps: state 1 action q-values

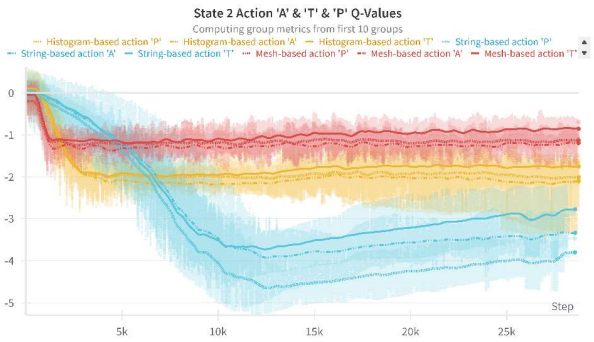


Figure 6.48: Four steps: state 2 action q-values

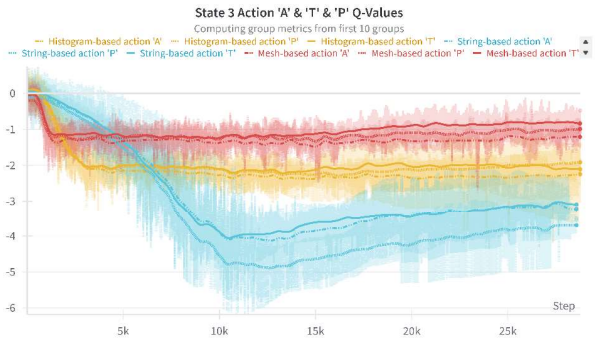


Figure 6.49: Four steps: state 3 action q-values

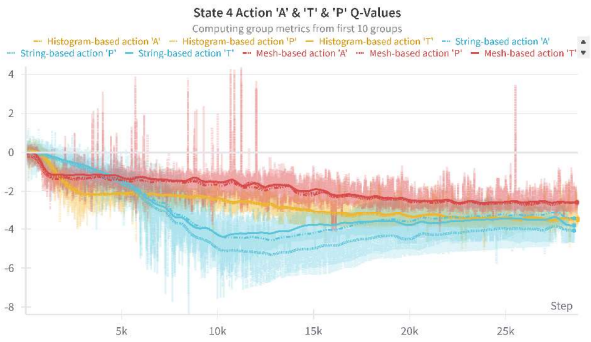


Figure 6.50: Four steps: state 4 action q-values

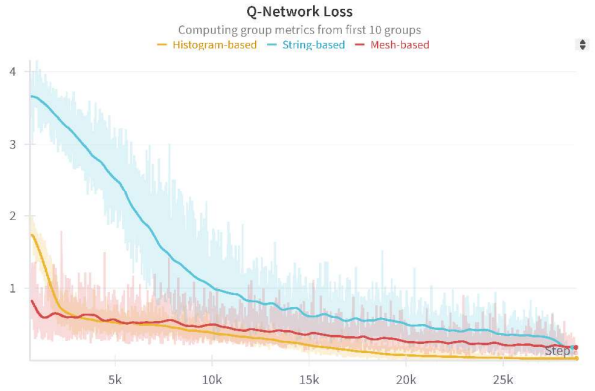


Figure 6.51: Four steps: Q-network loss

6.6 GENERALIZED TERMINAL STATE

6.6.1 FIVE STEPS

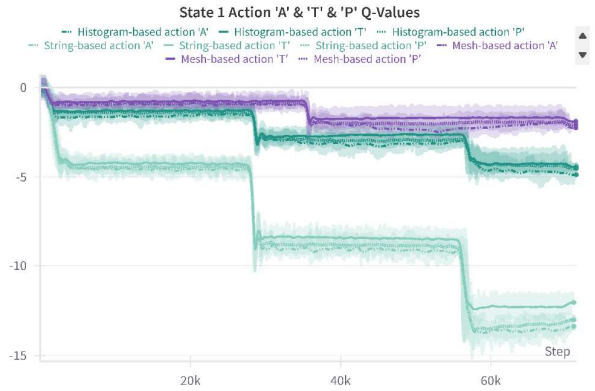


Figure 6.52: Five steps: state 1 action q-values

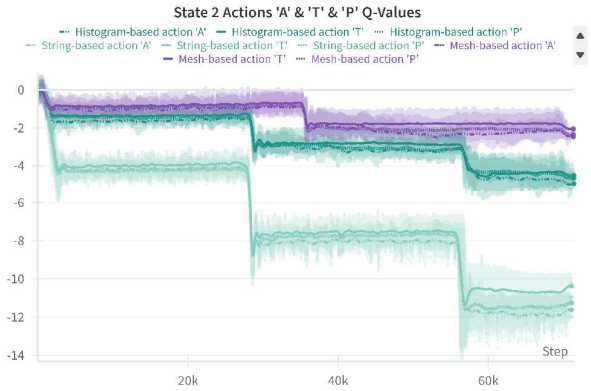


Figure 6.53: Five steps: state 2 action q-values

Figure 6.54: Five steps: state 3 action q-values

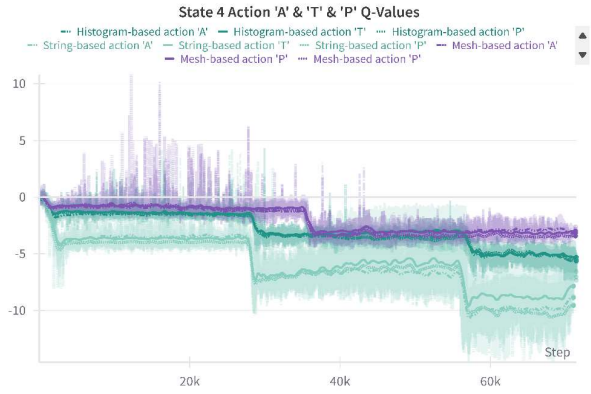


Figure 6.55: Five steps: state 4 action q-values

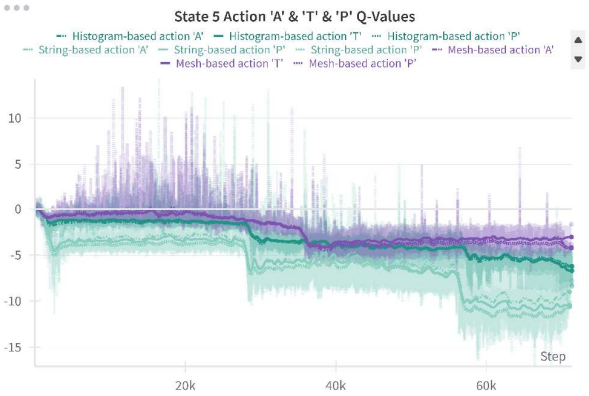


Figure 6.56: Five steps: state 5 action q-values

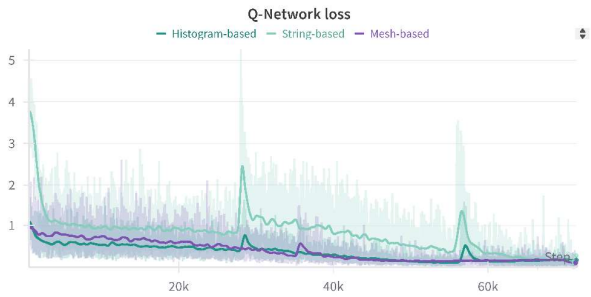


Figure 6.57: Five steps: Q-network loss

6.6.2 SIX STEPS

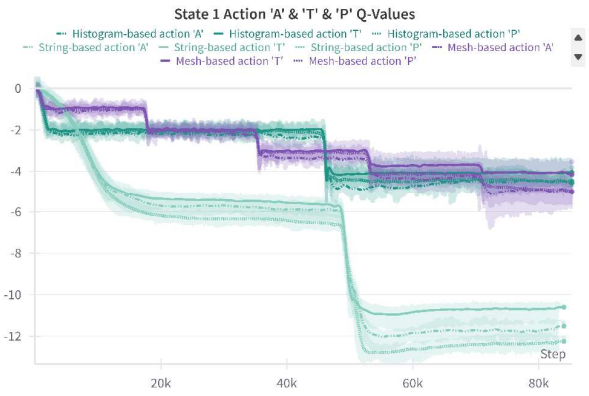


Figure 6.58: Six steps: state 1 action q-values

Figure 6.59: Six steps: state 2 action q-values

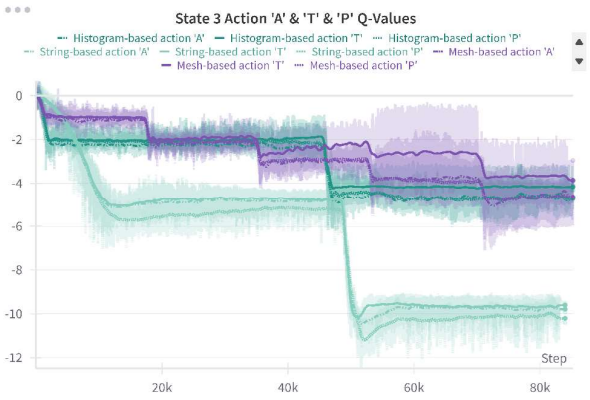


Figure 6.60: Six steps: state 3 action q-values

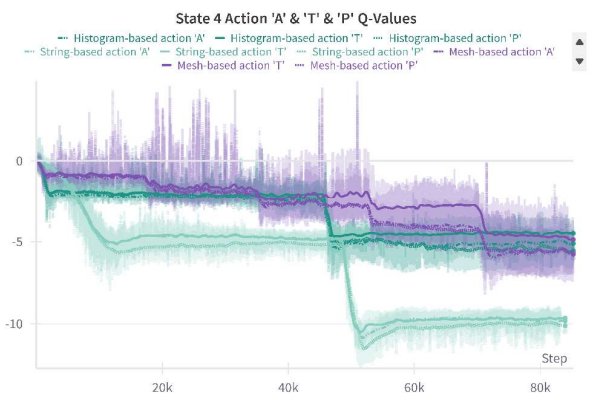


Figure 6.61: Six steps: state 4 action q-values

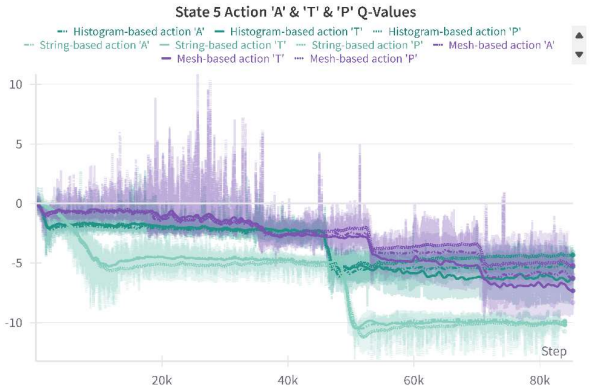


Figure 6.62: Six steps: state 5 action q-values

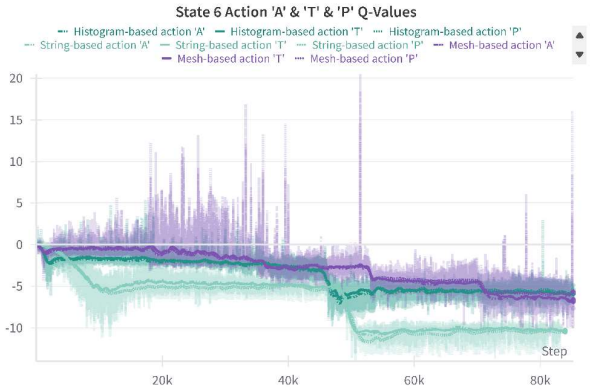


Figure 6.63: Six steps: state 6 action q-values

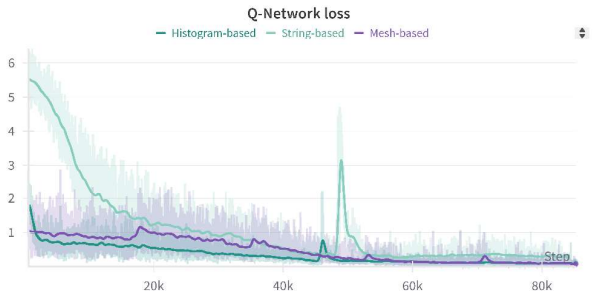


Figure 6.64: Six steps: Q-network loss

6.6.3 EIGHT STEPS

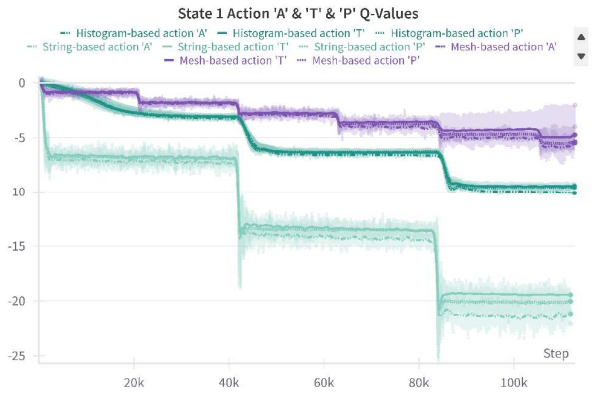


Figure 6.65: Eight steps: state 1 action q-values

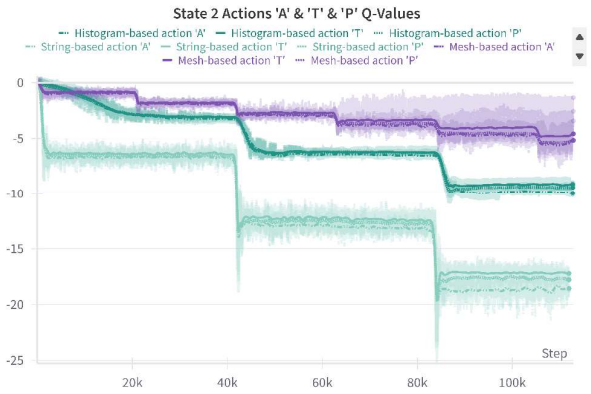


Figure 6.66: Eight steps: state 2 action q-values

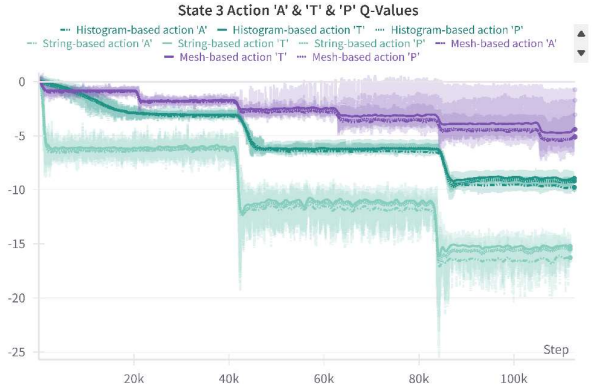


Figure 6.67: Eight steps: state 3 action q-values

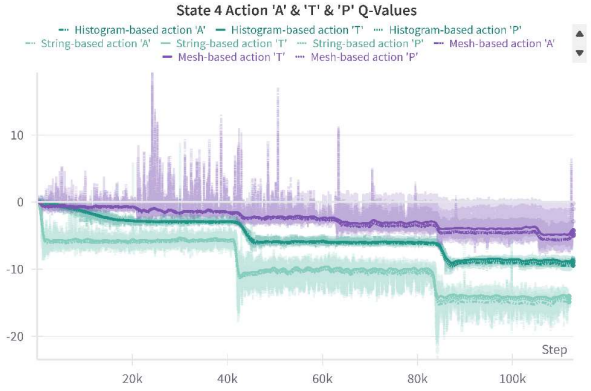


Figure 6.68: Eight steps: state 4 action q-values

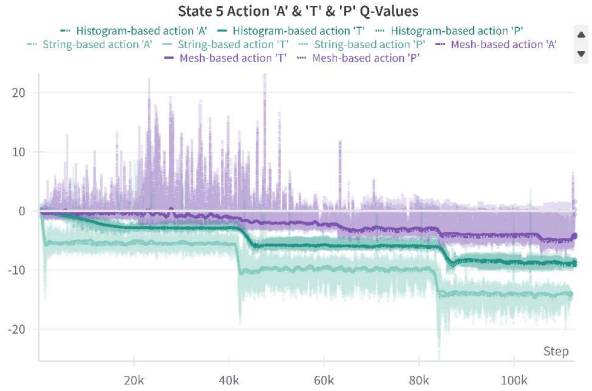


Figure 6.69: Eight steps: state 5 action q-values

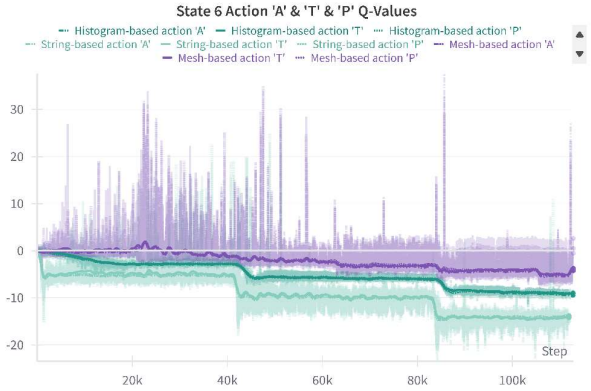


Figure 6.70: Eight steps: state 6 action q-values

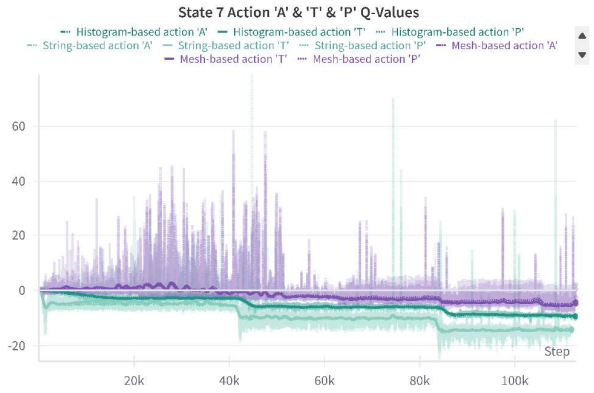


Figure 6.71: Eight steps: state 7 action q-values

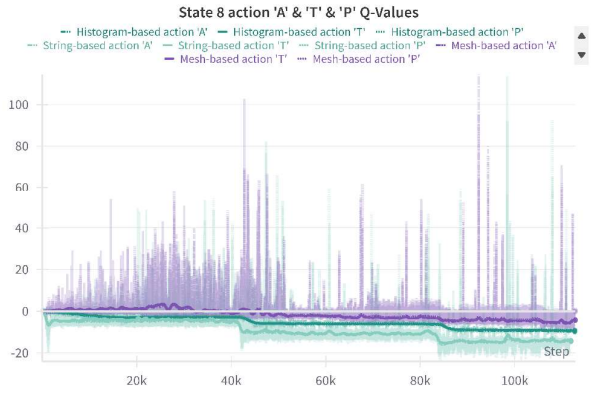


Figure 6.72: Eight steps: state 8 action q-values

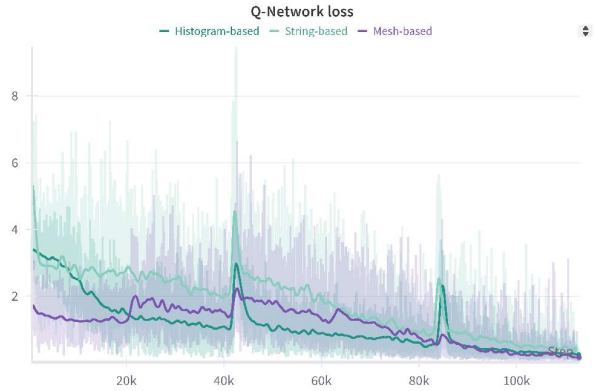


Figure 6.73: Eight steps: Q-Loss

6.6.4 HYBRID

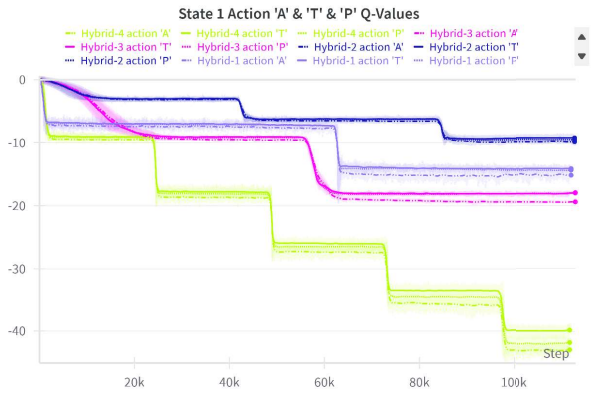


Figure 6.74: Hybrid: state 1 action q-values

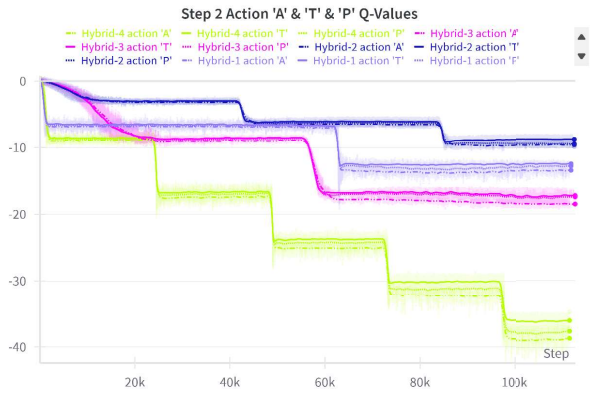


Figure 6.75: Hybrid: state 2 action q-values

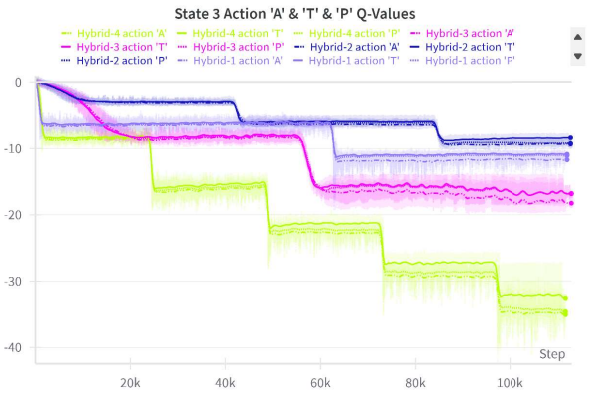


Figure 6.76: Hybrid: state 3 action q-values

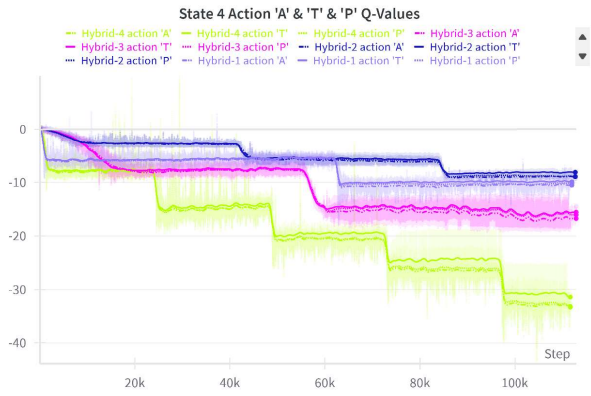


Figure 6.77: Hybrid: state 4 action q-values

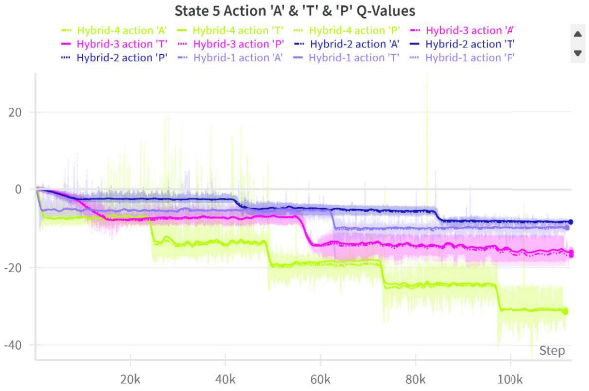


Figure 6.78: Hybrid: state 5 action q-values

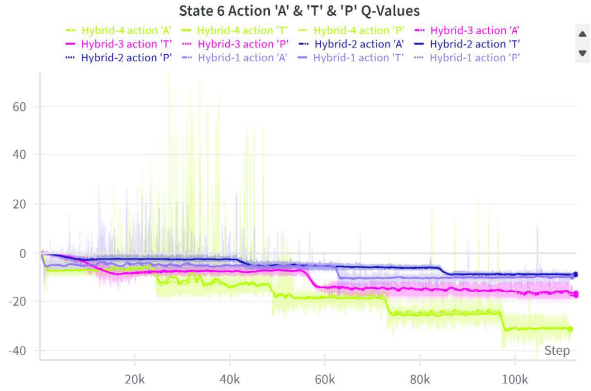


Figure 6.79: Hybrid: state 6 action q-values

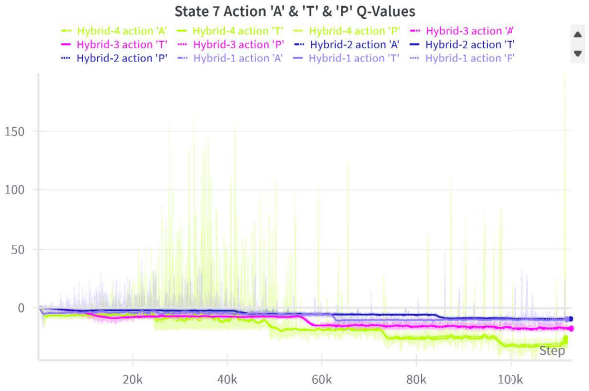


Figure 6.80: Hybrid: state 7 action q-values

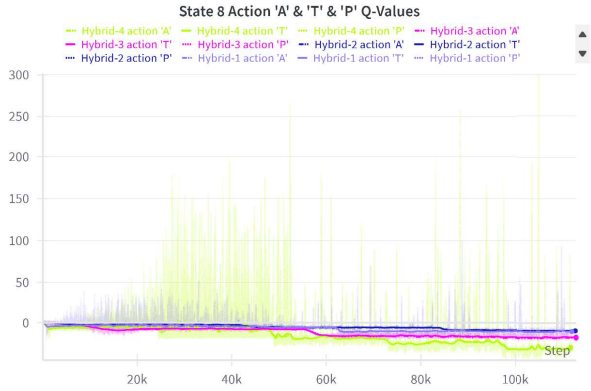


Figure 6.81: Hybrid: state 8 action q-values

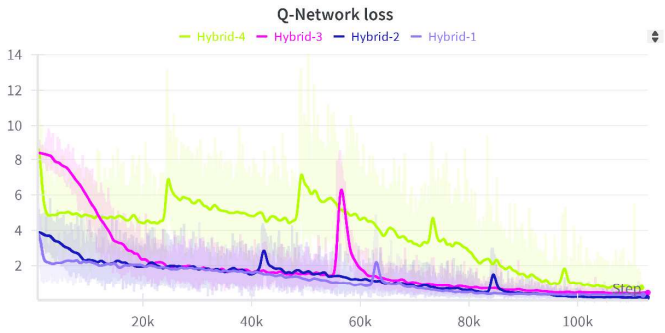


Figure 6.82: Hybrid: Q-Loss

Bibliography

[1] H. Adeli. 1986. Artificial intelligence in structural engineering. *Engineering Analysis*, 3, 3, 154–160. DOI: 10.1016/0264-682X(86)90053-5 (cit. on p. 23).

[2] P. Ayres, M. R. Thomsen, B. Sheil and M. Skavara. 2024. *Fabricate*. UCL Press. ISBN: 978-1-8000863-5-7 (cit. on p. 12).

[3] J. Bennett, L. Creary, R. Englemore and R. Melosh. 1978. SACON: A Knowledge-Based Consultant for Structural Analysis. DOI: 10.5555/892194 (cit. on p. 20).

[4] D. Billington, A. Halpern and S. Adriaenssens. 2013. The Ribbed Floor Slab Systems of Pier Luigi Nervi. *IASS Symposium 2013* (cit. on p. 12).

[5] D. Billington, A. Halpern and S. Adriaenssens. 2015. Human-level control through deep reinforcement learning. *Nature*, 518, 7540, 529–533. DOI: 10.1038/nature14236 (cit. on p. 101).

[6] P. Block, M. Breitenberger, D. I., R. Wüchner and K.-U. Bletzinger. 2016. Integrated design and analysis of structural membranes using the Isogeometric B-Rep Analysis. *Computer Methods in Applied Mechanics and Engineering*, 303, 312–340. DOI: 10.1016/j.cma.2016.02.003 (cit. on pp. 3, 9).

[7] D. Bommes, B. Lévy, N. Pietroni, E. Puppo, C. Silva, M. Tarini and D. Zorin. 2013. Quad-Mesh Generation and Processing: A Survey. *Computer Graphics Forum*, 32, 6, 51–76. DOI: 10.1111/cgf.12014 (cit. on pp. 11, 16, 17).

[8] C. J. Buckner. 2024. *From Deep Learning to Rational Machines*. Oxford University Press. ISBN: 9780197653302 (cit. on p. 103).

[9] N. Chomsky. 1956. Three models for the description of language. *IRE Transactions on information theory*, 2, 3, 113–124 (cit. on p. 24).

[10] H. Edelsbrunner. 2001. *Geometry and Topology for Mesh Generation*. Cambridge University Press. ISBN: 9780521793094 (cit. on p. 11).

[11] R. Evans. 2020. *Kant's Cognitive Architecture*. PhD thesis. Imperial College of London, Department of Computing (cit. on p. 103).

[12] V. Firoiu, W. F. Whitney and J. B. Tenenbaum. 2017. Beating the World's Best at Super Smash Bros. with Deep Reinforcement Learning. (2017). https://arxiv.org/abs/1702.06230 arXiv: 1702.06230 [cs.LG] (cit. on p. 43).

[13] M. S. Floater and K. Hormann. 2005. Surface Parameterization: a Tutorial and Survey. *Mathematics and Visualization*, 157–186. DOI: 10.1007/3-540-26808-1_9 (cit. on p. 9).

[14] J. Gero. 1994. Towards a model of exploration in computer-aided design. *Formal Design methods for Computer-Aided Design*, 315–336 (cit. on p. 21).

[15] M. Gohnert. 2022. *Shell Structures: Theory and Application*. Springer International Publishing. ISBN: 9783030848071 (cit. on p. 13).

[16] B. Grünbaum and G. C. Shephard. 1987. *Tilings and patterns*. W.H. Freeman and Company. ISBN: 9780716711933 (cit. on p. 9).

[17] D. Gunaratnam and J. Gero. 1994. Effect of Representation on the Performance of Neural Networks in Structural Engineering Applications. *Computer aided Civil Engineering*, 9, 2, 97–108. DOI: 10.1111/j.1467-8667.1994.tb00365.x (cit. on p. 21).

[18] M. Ioannis and F. Corentin. 2020. Design space exploration through force-based grammar rule. *archiDOCT*, 8, 1, 50–64 (cit. on p. 22).

[19] V. V. Isaeva, N. V. Kasyanov and E. V. Presnov. 2012. Topological singularities and symmetry breaking in development. *Biosystems*, 109, 3, 280–298. DOI: 10.1016/j.biosystems.2012.05.004 (cit. on p. 12).

[20] M. Ishizuka, K. Fu and J. Yao. 1982. Rule-based damage assessment system for existing structures (cit. on p. 20).

[21] K. Januszkiewicz and M. Banachowicz. 2017. Nonlinear Shaping Architecture Designed with Using Evolutionary Structural Optimization Tools. *Materials Science and Engineering*, 245, 8. DOI: 10.1088/1757-899X/245/8/082042 (cit. on p. 23).

[22] J. Laarman. 2017. *Joris Laarman Lab*. August Editions. ISBN: 978-1947359000 (cit. on p. 23).

[23] J. Lee, C. Mueller and C. Fivet. 2016. Automatic generation of diverse equilibrium structures through shape grammars and graphic statics. *International Journal of Space Structures*, 31, 2, 147–164. DOI: 10.1177/0266351116660798 (cit. on pp. 24, 25).

[24] M. Maher. 1985. HI-RISE and beyond: directions for expert systems in design. *Computer-Aided Design*, 17, 9, 420–427. DOI: 10.1016/0010-4485(85)90289-1 (cit. on p. 20).

[25] C. Málaga-Chuquitaype. 2022. Machine Learning in Structural Design: An Opinionated Review. *Frontiers in Built Environment*, 8. DOI: 10.3389/fbuil.2022.815717 (cit. on p. 26).

[26] G. Mirra. 2023. *Expertise, playfulness, analogical reasoning*. PhD thesis. University of Melbourne, Faculty of Architecture, Building and Planning (cit. on p. 32).

[27] G. Mirra and A. Pugnale. 2023. Enhancing interactivity in structural optimisation through Reinforcement Learning: an application on shell structures. *IASS Symposium* (cit. on p. 3).

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *arXiv*. http://arxiv.org/abs/1312.5602 (cit. on pp. 30, 44).

[29] C. T. Mueller. 2014. *Computational Exploration of the Structural Design Space*. PhD thesis. Massachusetts Institute of Technology, Department of Architecture (cit. on pp. 2, 24, 25).

[30] G. Nordenson. 2008. *Seven Structural Engineers: the Felix Candela Lectures*. New York: Museum of Modern Art. ISBN: 978-0-87070703-2 Check (cit. on pp. 15, 17).

[31] P. Nourian, S. Azadi and R. Oval. 2023. Generative Design in Architecture: From Mathematical Optimization to Grammatical Customization. *Computational Design and Digital Manufacturing*, 1–43. DOI: 10.1007/978-3-031-21167-6_1 (cit. on pp. 18, 22, 23, 30).

[32] M. Okereke and S. Keates. 2018. *Finite Element Applications*. Springer International Publishing. ISBN: 978-3-319-67125-3. DOI: 10.1007/978-3-319-67125-3 (cit. on p. 10).

[33] OpenAI. 2018. Spinning Up: A Taxonomy of RL Algorithms. *Accessed: July, 2024. [website], https://spinningup.openai.com/en/latest/spinningup/rl$_intro2.html$* (cit. on p. 42).

[34] OpenAI et al. 2019. Dota 2 with Large Scale Deep Reinforcement Learning. (2019). https://arxiv.org/abs/1912.06680 arXiv: 1912.06680 [cs.LG] (cit. on p. 43).

[35] M. E. Ororbia and G. P. Warn. 2023. Design Synthesis of Structural Systems as a Markov Decision Process Solved With Deep Reinforcement Learning. *Journal of Mechanical Design*, 145, 6, 1–43. DOI: 10.1115/1.4056693 (cit. on p. 30).

[36] R. Oval. 2019. *Topology Finding of Patterns for Structural Design*. PhD thesis. Universite Paris-ETS (cit. on pp. 1, 5, 9, 16, 52).

[37] R. Oval, R. Mesnil, T. Van Mele, P. Block and O. Baverel. 2023. A vector encoding for topology finding of structured quad-based patterns for surface structures. *International Journal of Space Structures*, 38, 4, 327–342. DOI: 10.1177/09560599231207650 (cit. on pp. 19, 38).

[38] Parametric-Architecture. 2024. 66 - Philippe Block - Computational Design, AI, Compas, Digital Fabrication, 3D-Printing, BRG. *Accessed: October, 2024. [Online video], https://www.youtube.com/watch?v=QfG52cINFqot=1182s* (cit. on p. 41).

[39] S. J. Prince. 2023. *Understanding Deep Learning*. MIT Press. ISBN: 9780262377102 (cit. on p. 26).

[40] P. Prusinkiewicz, J. Hanan, M. Hammel and R. Mech. 1997. L-systems: from the Theory to Visual Models of Plants. *Plants to Ecosystems*, 1–27 (cit. on p. 24).

[41] M. Rippmann, T. V. Mele, M. Popescu, E. Augustynowics, T. M. Echenagucia, C. J. Barentin, U. Frick and P. Block. 2016. The Armadillo Vault: Computational Design and Digital Fabrication of a Freeform Stone Shell. *Advances in Architectural Geometry*, 344–363. ISBN: 978-3-7281-3778-4 (cit. on p. 15).

[42] J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv*. https://arxiv.org/abs/1707.06347 (cit. on p. 43).

[43] K. Shea, J. Cagan and S. Fenves. 1997. A Shape Annealing Approach to Optimal Truss Design With Dynamic Grouping of Members. *Journal of Mechanical Design*, 119, 3, 388–394. DOI: 10.1115/1.2826360 (cit. on p. 24).

[44] e. a. Silver David. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529, 7587, 484–489. DOI: 10.1038/nature16961 (cit. on pp. 4, 42).

[45] T. Smithers. 1985. AI-based versus geometry-based design or Why design cannot be supported by geometry alone. *Computer-Aided Design*, 21, 3, 141–150. DOI: 10.1016/0010-4485(89)90068-7 (cit. on p. 20).

[46] G. Stiny and J. Gips. 1972. Shape Grammars and the Generative Specification of Painting and Sculpture. *Information Processing*, 71, 1460–1465 (cit. on p. 24).

[47] H. Sun, H. Burton and H. Huang. 2021. Machine learning applications for building structural design and performance assessment: State-of-the-art review. *Journal of Building Engineering*, 33. DOI: 10.1016/j.jobe.2020.101816 (cit. on p. 25).

[48] R. Sun and R. Vanluchene. 1990. Neural Networks in Structural Engineering. *Computer aided Civil Engineering*, 5, 3, 207–215. DOI: 10.1111/j.1467-8667.1990.tb00377.x (cit. on p. 21).

[49] R. S. Sutton and A. G. Barto. 2018. *Reinforcement learning: an introduction, Second edition*. The MIT Press. ISBN: 9780262352703 (cit. on pp. 30, 32, 46).

[50] K.-M. M. Tam, D. Kudenko, M. Khosla, T. Van Mele and P. Block. 2022. Performance-informed pattern modification of reticulated equilibrium shell structures using rules-based graphic statics, CW networks and reinforcement learning. *IASS Symposium* (cit. on p. 31).

[51] P. Veloso and R. Krishnamurti. 2020. An Academy of Spatial Agents - Generating spatial configurations with deep reinforcement learning. *eCAADe: Anthropologic*, 191–299. DOI: 10.52842/conf.ecaade.2020.2.191 (cit. on p. 45).

[52] A. Williams and T. Siegmund. 2021. Mechanics of topologically interlocked material systems under point load: Archimedean and Laves tiling. *International Journal of Mechanical Sciences*, 190. DOI: 10.1016/j.ijmecsci.2020.106016 (cit. on p. 9).

[53] H. Zheng. 2019. Form Finding and Evaluating Through Machine Learning: The Prediction of Personal Design Preference in Polyhedral Structures. *DigitalFUTURES*, 169–178. DOI: 10.1007/978-981-13-8153-9_15 (cit. on p. 29).

[54] J. Zhu, F. Wu and J. Zhao. 2021. An Overview of the Action Space for Deep Reinforcement Learning. *4th International Conference on Algorithms, Computing and Artificial Intelligence*, 119, 1–10. DOI: 10.1145/3508546.3508598. (cit. on pp. 31, 46).