Symmetry Heuristics for Stable Reinforcement Learning Design Agents

Akash Agrawal
Carnegie Mellon University

Christopher McComb
Carnegie Mellon University
ccm@cmu.edu

Deep Reinforcement Learning (RL) has emerged as a promising technique for automating configuration design because of its capacity for sequential decision-making. However, it faces challenges in learning stability when complex engineering simulations compose the reward function. This diminishes the practicality of deep RL for configuration design. To address this challenge, this work integrates configuration design heuristics in a deep RL framework to enhance stability and efficiently converge to high performance solutions. Specifically, we shape the reward based on symmetry, a deep-rooted heuristic that is widely applicable and frequently used in engineering design practice. This approach is empirically tested on a truss design problem wherein the RL agent employs a symmetry detection method during the design process. The results reveal that the proposed symmetry-guided approach consistently yields high-performance symmetric configurations, outperforming a naïve approach in terms of stability while also demonstrating an alignment with intuitive human design principles.

1 Introduction

Engineering design often entails navigating complex, non-differentiable, high-dimensional design spaces involving various objectives and

constraints. Deep Reinforcement Learning (RL) [1–3] has emerged as a promising technique within this landscape through its capacity for sequential decision-making, effectively managing design choices over time to achieve a solution [4–9]. It also offers several other advantages like adaptivity and transfer across design tasks [10,11]. However, RL faces significant challenges in ensuring learning stability and convergence [3], particularly when rewards are derived from complex engineering simulations. This work aims to address this gap by integrating heuristic rewards into the RL framework, enhancing the guidance of the learning process. We specifically shape the rewards based on the symmetry of design configurations, demonstrating how this approach not only improves stability and convergence but also aligns with intuitive human design principles.

The design of engineered systems involves the abstract arrangement of components, the selection of specific components, and the assignment of parameter values to the components. In some cases, design also encompasses the synthesis of new components; however, when no new components are synthesized, the design problem reduces to a configuration design problem [12], which is the focus of this work. Graph-based representations and methodologies like graph grammars are pivotal tools for configuration design, offering a structured approach to explore design spaces [13–16]. Specifically, nodes in a graph can represent components or subsystems while edges represent connectivity between them. A graph grammar provides a ruleset for how these graphs can be manipulated in a formalized manner. More recently, Graph Neural Networks (GNNs) [17] have showcased significant effectiveness in handing graph-based design representations [18–22].

In the realm of configuration design, RL emerges as a promising technique for exploring complex design spaces encapsulated within graph-based representations [4,5,8]. The sequential decision-making capability of RL algorithms aligns with the often iterative nature of design tasks wherein the impact of design choices unfolds over a sequence of steps. Specifically, the algorithms can refine the policy to guide the RL agent towards high-performance regions of the design space in a manner which directs each action towards long-term gains. This synergy can be further enhanced through structured exploration facilitated by domain-specific graph grammars [6]. Furthermore, the scalability of RL algorithms is practically achievable using GNNs as function approximators [6,18,19].

While RL offers numerous possibilities for advancing engineering design, its practical implementation is limited by efficiency challenges, encompassing aspects like resource availability, time utilization, learning stability and convergence behavior [3]. There has been an increasing interest in addressing these issues in the context of design. For instance, several multi-fidelity RL frameworks employing human-inspired approaches [23], transfer learning [24], and control variate approaches [25] have shown promise in reducing computational time. While these techniques effectively leverage reward approximations, there exists an opportunity to incorporate a broader spectrum of heuristic rewards, relevant across design problems. For instance, symmetry is a commonly exploited heuristic in mechanical design due to its ability to balance loads, reduce vibrations and improve space utilization [26]. These heuristics can be invaluable for improving efficiency in full configuration design problems with large graph-based design spaces [27].

The contribution of this work lies in enriching a foundational RL framework for configuration design with a symmetry reward and investigating its learning stability in comparison with a naïve approach. The rest of this paper is organized as follows. In Section 2, we brief configuration design and graph-based frameworks, and discuss GNNs and RL in context of design. Further, we discuss the potential of symmetry as a configuration design heuristic including some symmetry detection techniques. Section 3 begins by detailing a graph convolutional neural network-based RL framework for configuration design, setting the stage for the autonomous search of complex design spaces. This is followed by the formulation of a reward component that utilizes a symmetry detection method to bias the agent towards symmetric configurations. In Section 4, a truss design case study is described to demonstrate the effectiveness of the approach. Section 5 presents the results of this study, including a comparison with a naïve RL agent. Section 6 summarizes the contribution of the paper and proposes directions for future work.

2 Background

2.1 The Synergy Between Configuration Design and Graph-Based Frameworks

Configuration design is the systematic process of selecting and arranging predefined components, as well as assigning parameter values to synthesize a functional system that optimizes its objectives under various

constraints [12]. Given an exponentially large design space of possible configurations, the choice of representation for the design space is crucial for effective optimization. In this regard, graph-based representations prove particularly apt [13]. Graphs model the inherent structure within a complex system, where vertices symbolize individual components or subsystems, and edges represent relationships such as spatial connectivity. This intuitive mapping between a design configuration and a graph allows for a formal, structured approach to explore design configurations. Moreover, graphs provide a flexible medium that facilitates the frequent design iterations common in configuration design.

From a mathematical standpoint, a configuration design problem can be formulated as an optimization problem over a graph $G = (N, E)$, where N denotes the set of nodes and E the set of edges. Each node consists of component or subsystem information, including various continuous and discrete variables like the coordinates, orientation, dimensions, and material. For subsystems, it can capture additional holistic attributes such as component enumerations, and operational parameters like pressure and voltage. Each edge encapsulates a connectivity relationship between two components or sub-systems, encompassing continuous and discrete variables involving different connectivity types (like fastening, welding).

The flexibility of graphs is extended by incorporating graph grammars [13–16], which are a set of operations that dictate how the graph can be modified. These operations formalize the steps by which a design can evolve. The simplest types of operations include node addition, node deletion, edge addition, edge deletion and tuning node and edge attributes. These operations can be parameterized to add more context or conditions [28,29]. More complex rules can encapsulate subgraph transformations, wherein a whole cluster of connected vertices and edges can be replaced by another cluster to yield a better configuration [13].

While graph grammar provides a framework for systematically generating a wide array of alternatives in the configuration design space, it presents challenges especially in complex systems [13,29]. Developing an extensive set of rules demands significant expertise, especially for large, diverse design spaces. Further, inadequately defined grammar may restrict design options, potentially resulting in suboptimal outcomes. Addressing these limitations, a grammar derived from GNNs may offer a more effective approach. At the core of GNNs is the ability to capture relational patterns between graph elements [17], which is critical in configuration design where the interplay between components can potentially dictate

further design operations. Specifically, by learning these relationships, GNNs can execute complex combinations of elementary operations across the entire graph for a given design configuration.

2.2 Configuration Design with Graph Neural Network Based Reinforcement Learning Agents

In context of configuration design, RL algorithms [1] have demonstrated their applicability in the sequential decision-making task of searching the design space for high-performance configurations [4–9]. Specifically, the state space of the Markov Decision Process [1] is the set of all possible graphs that represent the design configurations, and the design operations are agent actions. At a specific iteration, the agent state is the graph that represents the design configuration at that design step. The agent actions define the set of design operations to be performed on the graph that transforms it to the next configuration represented by another graph. These operations can range from elementary ones to more complex subgraph transformations. Further, the reward function encapsulates the objectives and constraints of the problem. The transition probability function for configuration design problems is typically deterministic. Lastly, the policy serves as a strategy for design, mapping design configurations to design operations with the goal of optimizing the performance of the configuration over multiple successive design operations.

While the sequential decision-making paradigm of RL aligns with configuration design, the high dimensionality and complexity of problems limits the applicability of traditional RL algorithms. In this regard, the incorporation of deep learning within RL frameworks is emerging as increasingly crucial. Specifically, deep neural networks have proven useful in serving as function approximators for the policy and value functions in RL [2]. In context of the application of RL to configuration design, a GNN based policy autonomously learns to transform design configurations for long-term gains, leading to high-performance configurations. Specifically, the reward feedback from evaluating the configurations in simulations informs updates to the GNN, progressively enhancing the policy and subsequently improving the quality of the designs [6].

While deep RL offers numerous possibilities for advancing engineering design, its practical implementation is limited by challenges in achieving efficiency, encompassing aspects like resource availability (computing systems and data needed), time utilization (speed of learning), stability (consistency of learning across training runs or episodes) and the resultant

convergence behavior [3]. To enhance the learning stability of the RL agent, this work incorporates a broader spectrum of heuristic rewards that are relevant across design problems, essentially shaping the reward function. It aims to guide the agent exploration in large graph-based design spaces more effectively, leading to more stable learning outcomes. Specifically, we enrich a GNN based RL framework with a reward based on symmetry, a deep-rooted heuristic in engineering design practice.

2.3 Heuristic-Guided Reinforcement Learning

Heuristic-guided reinforcement learning is an emerging field that combines the strengths of traditional heuristics with the adaptability and learning capabilities of RL. Heuristics encompass strategies that guide problem-solving processes based on empirical evidence, expert knowledge, and theoretical insights which are critical in managing complex problems like those in design [30,31]. The integration of heuristics in the RL algorithm can steer the learning process towards high performance solutions, potentially enhancing learning stability. One way to do this is to leverage them in the action space to bias the action selection towards more promising directions [32]. Shaping or regularizing the reward or value functions based on heuristics is another avenue to incorporate prior knowledge [33,34]. For instance, Cheng et. al [35] start with short-horizon tasks using heuristic rewards and gradually shift to longer horizons with less reliance on these rewards.

Symmetry is a pervasive and powerful heuristic that transcends disciplines, from biology, mathematics, ML to engineering. In biology, symmetric structures are favored due to their lower informational encoding requirements, which leads to greater mutation resilience [36]. Symmetry is also a fundamental principle in engineering, employed as a heuristic to address several design challenges [37]. For instance, in mechanical and civil engineering, symmetry is essential for balancing loads, reducing vibrations, and optimizing space utilization [26]. Even in seemingly asymmetrical systems, symmetry could still reign on a local scale, thus showcasing the pervasive nature of symmetry [38]. Furthermore, heuristics like modularity, often intertwine with symmetry, exhibiting how these heuristics could be enveloped within the symmetry framework [39]. This common thread of symmetry illustrates its integral role as a manifestation of underlying physical principles, a natural pattern leveraged by evolution, and a strategic design element in human-engineered systems.

Symmetry is mathematically described as an object's attribute of being invariant, when subjected to geometric transformations. It is typically categorized according to the characteristics of the underlying transformations and the resultant functionality, primarily falling into point group or space group symmetry [26]. Point groups like cyclic and dihedral symmetry groups ensure at least one point remains stationary during transformation. This is crucial for applications like braking systems that require rotational or mirror symmetry. In contrast, space group symmetry incorporates translational symmetry and is instrumental in optimizing material use in design layouts.

In addition to exploring the significance and applicability of different symmetry types, there has also been extensive research on the automated detection of symmetry. This is pursued both as a theoretical challenge to analyze intricate geometry and as a tool for applications like model compression [38] and capturing design intent [40]. The methodologies for symmetry detection vary depending on factors like the data type (feature tree [41], projected view/drawings [42], images [43], point clouds [44], voxels [45], graphical models [40]), completeness of data (complete or approximate [46]), type of symmetry transformation (scaling, reflection, rotation, translation or their combinations [47]), scale of symmetry (global [41] and/or local/partial [38]), and precision (exact [41] or approximate [38]). While there is a very diverse range of detection techniques [38,40–49], we briefly discuss one for global symmetry detection in graphical data structures. It is based on the Graph Edit Distance (GED) [50] between a graph and its symmetric transformation. GED is a measure that finds the minimal set of edit operations needed to transform one graph into another and has been commonly used for graph comparison. The editing operations involved in computing GED include insertions, deletions, and substitutions for both nodes and edges. The detection of symmetry using GED involves the minimization of this distance via an optimization algorithm or picking a symmetry element like a reflection axis from a set of candidates for which this distance is the minimum.

3 Methodology

3.1 Foundational RL Framework for Configuration Design

The RL based design agent in this work aims to address the configuration design problem by starting from a seed design graph and iteratively modifying the graph to minimize an objective function, while adhering to

the constraints. The interaction between the agent and the design space, particularly as the agent transitions from state s_t (consisting of the graph G_t) to state s_{t+1} (consisting of the graph G_{t+1}) by performing actions (a_t) in the form of operations on the graph, is detailed hereinafter.

The action space of the agent is grounded in a graph-structured set of design operations that are derived from a graph convolutional neural network, which serves as a policy function approximator for the RL agent. At each iteration t, the policy π outputs an action graph a_t, wherein each node has an attribute vector. This includes one element corresponding to the first operation of *node deletion*. This operation also deletes the edges connected to the deleted node. The next set of elements correspond to the *tuning* of the continuous and integer variables that define the *edge attributes*. Specifically, the tuning values for edges are obtained through aggregation operations (like a summation) on the vector elements corresponding to nodes that defined the edge. Further, the next set of elements govern the *tuning* of the continuous and integer variables that define the *node attributes*. The next element governs the *addition of a new node* to the node via an edge, thereby expanding the graph. Lastly, the next set of elements determine the *attributes of the new node* if it has been added. These operations (denoted by $\oplus$) are applied on G_t to obtain G_{t+1}.

Figure 1 shows an example of the action space for a truss design problem, wherein the action graph consists of seven elements per node. These elements correspond to node deletion (along with the members that it connects), two elements for the editing member type and dimension, two elements to edit the joint co-ordinates, one element to govern joint addition via a member and two elements to determine the co-ordinates of the joint.

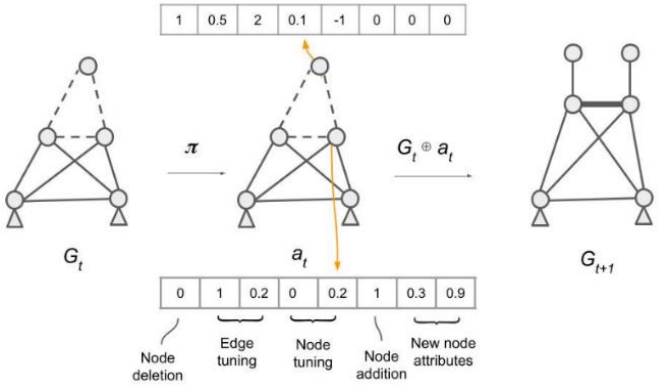

Fig. 1 Action space of RL agent for a truss design problem

Although this example is tailored to trusses, the fundamental nature of these graph-based operations makes them adaptable to any problem that can be modeled with graphs. In addition to the policy function approximator, this work also utilizes a graph convolutional neural network for approximating the value function as illustrated in Figure 2. Specifically, it utilizes an attention pooling mechanism for an adaptive focus on different parts of the graph.

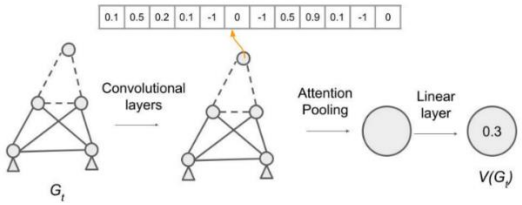

Fig. 2 Value function based on graph convolutional layers and attention pooling

Lastly, the agent reward should reflect the quality of the action (a_t) that transitions the design from state G_t to G_{t+1}. This depends on the amount by which the objective reduces. Further, when the agent is in the infeasible domain, the change in the amounts by which each of the constraints are violated would guide the agent to navigate to feasible regions. The specific formulations of these reward components are adapted from prior work that utilizes deep RL for skeletal design [23]. Lastly, the graphs that result in invalid designs (i.e. objectives or constraints not computable) are assigned a poor objective value to steer the agent towards valid configurations.

3.2 Symmetry Reward Formulation

In addition to the reward components corresponding solution performance, we introduce a novel symmetry reward component in this work. This component employs a symmetry detection technique that finds the minimum GED between the graph representation of the design and symmetric transformations. These transformations are obtained by mirroring the graph across candidate axes passing through its centroid. Specifically, we compute the GED corresponding to each axis by preparing a cost matrix of the pairwise distance between nodes and solving a linear sum assignment problem [51]. The goal of this problem is to assign each node in the original graph to a node in the transformed graph such that the sum of all costs is minimized. Based on the solutions of these problems, we define the degree of symmetry (η) as follows:

$$\eta = -\min_{T_i \in \mathcal{T}} GED(G, T_i(G)) \qquad (1)$$

where, $\mathcal{T}$ is the set of candidate transformations. Further, we clip the degree of symmetry values below a threshold to ensure that the agent does not need to learn from overly complex patterns amongst highly asymmetric outliers. Figure 3 shows an illustrative example with node attributes as the co-ordinates of truss joints. The GED is evaluated between the graph and its mirror symmetry counterpart about four candidate symmetry axes, the minimum of which determines the degree of symmetry.

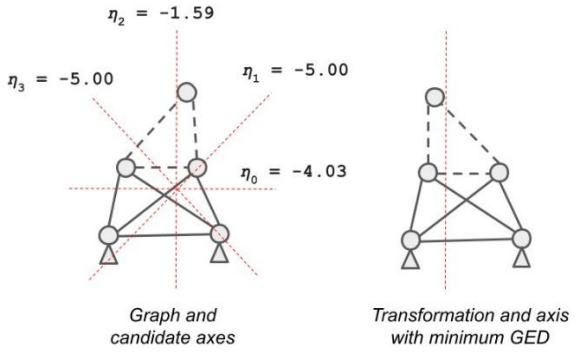

Fig. 3 Degree of symmetry metric rooted in graph edit distance

The degree of symmetry can be further utilized to define a reward component to extend the foundation RL framework in Section 3.1 This reward component denoted by R_η is defined as the change in the degree of symmetry when the graph transition from G_t to G_{t+1}:

$$(R_\eta)_{t+1} = w_\eta \times (\eta(G_{t+1}) - \eta(G_t)) \qquad (2)$$

where, w_η is the weight associated with this symmetry reward component. Like the other reward components, invalid configurations are assigned a poor value to steer the agent towards valid configurations.

4 CASE STUDY

4.1 Truss Design Problem

In this work, a truss design problem serves as a testbed for evaluating the effectiveness of incorporating symmetry in RL-based configuration design. A truss is a structure which is designed to bear loads in buildings and

bridges. The optimization objective in this problem is to minimize the mass of the truss while adhering to a target Factor of Safety (FOS), a measure of the load-bearing capacity of a structure beyond the expected loads. The specific 2D truss design problem tackled in this work consists of two fixed supports at the bottom corners and a vertical load at a node midway between them. A target FOS value of 1.7 is set for the problem. This problem permits a huge number of potential solutions – up to 10^{75} unique configurations of trusses with up to 40 joints, without yet accounting for the positioning of joints in a continuous space. This enormity underscores the complexity of the problem and the challenge it poses for design algorithms.

To set the stage for the symmetry-guided RL framework, the truss configurations are represented as graphs, wherein the edges represent the members and nodes represent the joints. Further, the node attributes include the joint co-ordinates, and the edge attributes consists of a scalar size variable. This graph representation defines the agent state. Actions include removing a node, editing node and edge attributes and adding a new node. Further, the actions that modify the nodes corresponding to the fixed joints and loads are masked during the transition to the new state. For the symmetry-guided agent, the reward is computed based on the mass objective, the FOS equality constraint, and the symmetry reward with their weights tuned to yield satisficing designs. For the naïve agent, the same reward function is used without the symmetry component.

4.2 Training and Evaluating the RL agents

An online one-step actor-critic [1] algorithm with graph convolutional neural networks as function approximators is used for training the policies of the naïve and symmetry-guided agent. The utilization of an online algorithm improves the memory requirements by only having to store the most recent design graph, while also opening the potential for faster learning due to frequent updates. Several random graphs that may or may not be feasible trusses are used as seed designs in the training episodes, each with a maximum length of 20 iterations. To bound the design space and ensure stable learning, the episodes are terminated whenever a configuration with less than 4 or more than 40 nodes is encountered. A common set of seed designs, initial neural networks and hyperparameters are utilized for training the agents. To understand the learning stability of the agents, the policies were periodically frozen during training for further evaluation, the results of which are discussed hereinafter.

5 RESULTS AND DISCUSSION

The naïve and symmetry guided policies are periodically frozen at intervals of 200 episodes. After completion of training at 3000 episodes, each of these policies were evaluated to assess its performance in terms of mass, FOS, and the degree of symmetry. This section provides a detailed account of this evaluation data including an analysis of the distribution of solutions and the stability of the learning process across episodes.

Firstly, we investigate the distributions of the final solution qualities and visualize a few representative truss designs from this distribution. This quality is measured by the weighted sum of the mass objective and the FOS equality constraint penalty with the same weights that were utilized in the reward formulation. Figure 4 shows these distributions for the naïve and symmetry-guided agent. Both the distributions are multimodal in nature, which implies that the agents find several local minima in the design space. However, the locations and densities of the peaks are significantly different for both agents. Figure 5 shows some representative trusses corresponding to the labelled peaks in Figure 4.

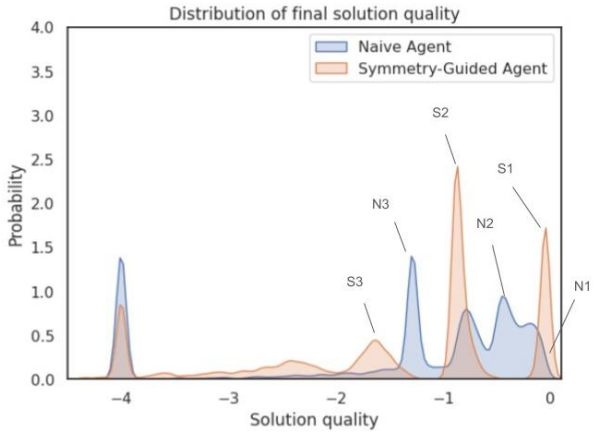

Fig. 4 Distribution of final solution quality

We observe that the symmetry-guided agent has a distinct sharp peak of the highest quality (S1), which showcases its ability to find high performance solutions. The corresponding truss in Figure 5 meets the FOS constraint with a moderate mass value. Moreover, this design is symmetric in nature, much like what is observed in trusses designed by human designers [52]. The highest quality design corresponding to the naïve

agent's peak (N1) also meets the FOS constraint but has a higher mass than the design S1. Moreover, it exhibits a lower degree of symmetry compared to design S1. The next two peaks for the symmetry-guided agent (S2 and S3) are also highly symmetric. However, they do not meet the FOS constraint. While these are not immediately usable configurations, their symmetric structure may still be of value to a human designer for further exploration.

Further, for the naïve agent, we observe that the designs N2 and N3 also do not meet the FOS constraint. However, they have higher qualities compared to solutions S2 and S3 respectively, albeit with varying densities. On one hand the symmetry-guided agent is biased towards symmetric solutions, while on the other hand the naïve agent potentially needs to explore a larger portion of the design space to find high performance solutions. Lastly, both agents yield about 20% of infeasible trusses as reflected by the peaks on the far left with a quality of -4.0. The symmetry-guided agent has a lower density of infeasible solutions that further showcases its superiority in exploring the design space.

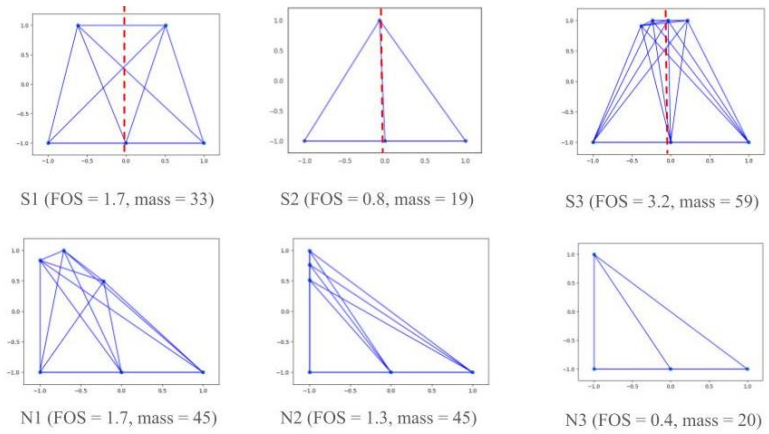

Fig. 5 Representative truss design solutions (red line is detected symmetry axis)

To understand the stability of the learning process, we freeze and evaluate the policies periodically. The distribution of mass, FOS and the degree of symmetry of this evaluation data is visualized in Figures 6, 7 and 8 respectively. In Figure 6, the invalid trusses are assigned a mass value of 0 in the plot. On the onset of training, a relatively high number of trusses are invalid for both the agents. As learning progresses, the mass values start to converge to distinct modes for both the agents, much like the peaks that were observed in the overall quality distributions. However, we observe

minor inconsistencies across episodes for the naïve agent. For instance, the lowest mode of mass abruptly shifts a cell upwards during the last period of evaluation. This is indicative of instability in the learning process.

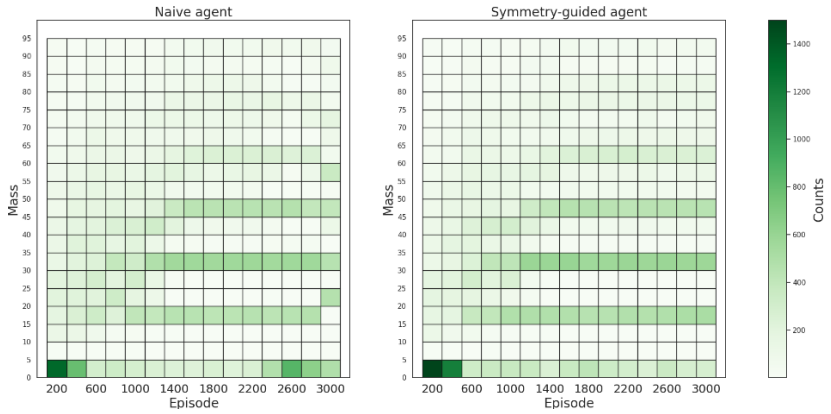

Fig. 6 Evolution of mass across evaluation of frozen policies

Figure 7 showcases the trend for the FOS. Like the previous plot, invalid trusses are assigned a value of 0.0. As learning progresses, the values start to converge to distinct modes for both the agents, much like evolution of mass. A fairly high proportion of configurations meet the constraint of 1.7 for both the agents around 2000 episodes. However, the behavior of the agents varies significantly as learning progresses. While the symmetry-guided agent effectively converges with a high density of feasible solutions, the naive agent exhibits high instability in this period.

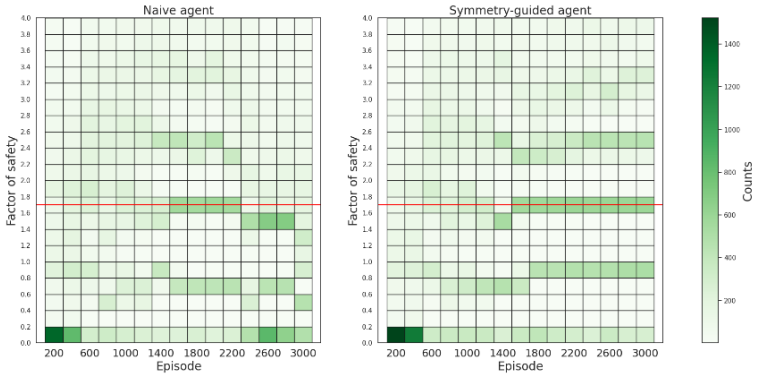

Fig. 7 Evolution of FOS across evaluation of frozen policies
(red horizontal line is target value)

Figure 8 showcases the evolution of the degree of symmetry for both the agents. On the top end, a value of 0.0 corresponds to perfect symmetry, while on the other end a value of -5.0 corresponds to invalid trusses as well as trusses with a degree of symmetry lower than or equal to -5.0. Both the agents exhibit increasing symmetry in the middle of the training around 2000 episodes. This corresponds to the high density of solutions that met the FOS constraint for both the agents. Further, the symmetry-guided agent shows a very high proportion of symmetric designs until the end of learning. However, there is a rapid drop in the case of the naïve agent as it is not driven by the symmetry reward. This shows that the symmetry reward indeed results in a consistent set of high-performance solutions.

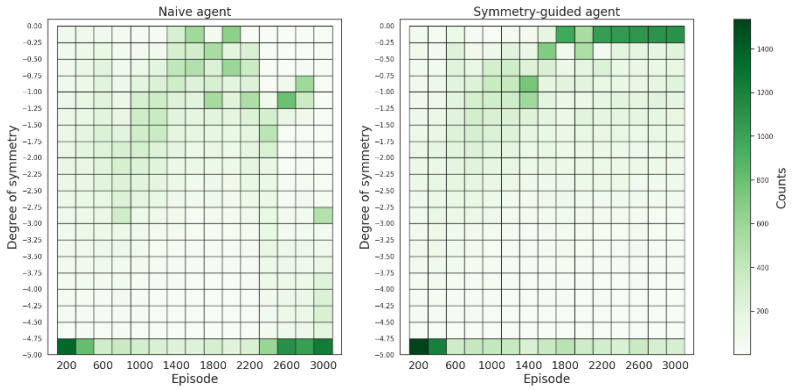

Fig. 8 Evolution of degree of symmetry across evaluation of frozen policies

Overall, the evolution of mass, FOS, and symmetry of both the agents highlight the superior stability and convergence behavior of the symmetry-guided agent in comparison to the naïve agent. While it may be valuable to stop the training of the naïve agent early, it is impractical to continually evaluate an RL agent and identify this performance drop while training. Rather, the symmetry-guided technique provides a more stable approach with a tendency to converge without intensive monitoring.

6 CONCLUSION AND FUTURE WORK

This research proposes a novel paradigm in the application of deep RL to configuration design, particularly involving the use of symmetry heuristics in a framework based on graph convolutional neural networks. The integration of a reward based on the degree of symmetry of configurations has showcased enhanced stability and convergence in the learning process.

Specifically, an empirical investigation is conducted on a truss design problem to substantiate the efficacy of the symmetry-guided RL approach. The results demonstrate that the symmetry-guided agent not only outperforms its naïve counterpart in terms of stability and convergence but also resonates with the natural design inclinations of human experts.

While the proposed symmetry-guided approach has proven effective for a symmetrical truss design problem, engineering problems frequently exhibit only partial symmetry [38]. Future work should focus on investigating the effectiveness of the approach in such scenarios. Learning the symmetry characteristics as an integral part of the design configuration, rather than relying on separate detection algorithms, could enhance the framework's adaptability. Furthermore, the computational cost of calculating the GED can be prohibitively high. A potential solution lies in actively learning a surrogate model [53] that can be deployed in a variable fidelity framework [23]. A more extensive exploration of efficiency metrics like time utilization with such algorithmic enhancements remains a valuable pursuit for subsequent studies.

Future work could also delve into utilizing the symmetry heuristic within the action space rather than the reward function of the agent [54]. Exploring generalization capabilities of the symmetry-guided approach across a spectrum of truss problems is another promising avenue, potentially incorporating conditional factors for boundary and load-bearing nodes. Beyond trusses, the applicability of this approach to diverse domains, such as the design of drones, which often involve heterogeneous graph representations, could significantly broaden the impact of this research in the field of automated configuration design.

References

1. Sutton, R.S.; Barto, A.G. *Reinforcement Learning : An Introduction*; 2018; ISBN 9780262039246.
2. Li, Y. Deep Reinforcement Learning. **2018**.
3. Dong, H.; Ding, Z.; Zhang, S.; Fundamentals, E.; Research, A. *Deep Reinforcement Learning*;
4. Ororbia, M.E.; Warn, G.P. Design Synthesis of Structural Systems as a Markov Decision Process Solved With Deep Reinforcement Learning. *Journal of Mechanical Design* **2023**, *145*, doi:10.1115/1.4056693.
5. Liao, H.; Zhang, W.; Dong, X.; Poczos, B.; Shimada, K.; Burak Kara, L. A Deep Reinforcement Learning Approach for Global

Routing. *Journal of Mechanical Design* **2020**, *142*, doi:10.1115/1.4045044.
6. Fogelson, M.B.; Tucker, C.; Cagan, J. GCP-HOLO: Generating High-Order Linkage Graphs for Path Synthesis. *Journal of Mechanical Design* **2023**, *145*, doi:10.1115/1.4062147.
7. Raina, A.; Cagan, J.; McComb, C. Learning to Design Without Prior Data: Discovering Generalizable Design Strategies Using Deep Learning and Tree Search. *Journal of Mechanical Design* **2023**, *145*, doi:10.1115/1.4056221.
8. Whitman, J.; Bhirangi, R.; Travers, M.; Choset, H. *Modular Robot Design Synthesis with Deep Reinforcement Learning*;
9. Jang, S.; Yoo, S.; Kang, N. *Generative Design by Reinforcement Learning: Enhancing the Diversity of Topology Optimization Designs*;
10. Lee, X.Y.; Balu, A.; Stoecklein, D.; Ganapathysubramanian, B.; Sarkar, S. A Case Study of Deep Reinforcement Learning for Engineering Design: Application to Microfluidic Devices for Flow Sculpting. *Journal of Mechanical Design, Transactions of the ASME* **2019**, *141*, doi:10.1115/1.4044397.
11. Yonekura, K.; Hattori, H. Framework for Design Optimization Using Deep Reinforcement Learning. *Structural and Multidisciplinary Optimization* 2019, *60*, 1709–1713.
12. Wielinga, B.; Schreiber, G. Configuration-Design Problem Solving. *IEEE expert* **1997**, *12*, 49–56.
13. Voss, C.; Petzold, F.; Rudolph, S. Graph Transformation in Engineering Design: An Overview of the Last Decade. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM* 2023, *37*.
14. Zhao, A.; Xu, J.; Konaković-Luković, M.; Hughes, J.; Spielberg, A.; Rus, D.; Matusik, W. RoboGrammar: Graph Grammar for Terrain-Optimized Robot Design. *ACM Trans Graph* **2020**, *39*, doi:10.1145/3414685.3417831.
15. Puentes, L.; Cagan, J.; McComb, C. Heuristic-Guided Solution Search through a Two-Tiered Design Grammar. *J Comput Inf Sci Eng* **2020**, *20*, doi:10.1115/1.4044694.
16. Oberhauser, M.; Sartorius, S.; Gmeiner, T.; Shea, K. Computational Design Synthesis of Aircraft Configurations with Shape Grammars. In *Design Computing and Cognition '14*; Springer International Publishing, 2015; pp. 21–39.
17. Kipf, T.N.; Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. **2016**.

18. Almasan, P.; Suárez-Varela, J.; Rusek, K.; Barlet-Ros, P.; Cabellos-Aparicio, A. Deep Reinforcement Learning Meets Graph Neural Networks: Exploring a Routing Optimization Use Case. *Comput Commun* **2022**, *196*, 184–194, doi:10.1016/j.comcom.2022.09.029.
19. Mirhoseini, A.; Goldie, A.; Yazgan, M.; Jiang, J.W.; Songhori, E.; Wang, S.; Lee, Y.J.; Johnson, E.; Pathak, O.; Nazi, A.; et al. A Graph Placement Methodology for Fast Chip Design. *Nature* **2021**, *594*, 207–212, doi:10.1038/s41586-021-03544-w.
20. Song, B.; McComb, C.; Ahmed, F. Assessing Machine Learnability of Image and Graph Representations for Drone Performance Prediction. In Proceedings of the Proceedings of the Design Society; Cambridge University Press, May 1 2022; Vol. 2, pp. 1777–1786.
21. Zhao, P.; Liao, W.; Huang, Y.; Lu, X. Intelligent Beam Layout Design for Frame Structure Based on Graph Neural Networks. *Journal of Building Engineering* **2023**, *63*, doi:10.1016/j.jobe.2022.105499.
22. Su, X.; Wu, C.; Gao, W.; Huang, W. Interior Layout Generation Based on Scene Graph and Graph Generation Model. In *Design Computing and Cognition'20*; Springer International Publishing, 2022; pp. 267–282.
23. Agrawal, A.; McComb, C. Reinforcement Learning for Efficient Design Space Exploration With Variable Fidelity Analysis Models. *J Comput Inf Sci Eng* **2023**, *23*, doi:10.1115/1.4056297.
24. Bhola, S.; Pawar, S.; Balaprakash, P.; Maulik, R. Multi-Fidelity Reinforcement Learning Framework for Shape Optimization. *J Comput Phys* **2023**, *482*, doi:10.1016/j.jcp.2023.112018.
25. Khairy, S.; Balaprakash, P. Multifidelity Reinforcement Learning with Control Variates. **2022**.
26. Qiu, Q.; Chen, X.; Yang, C.; Feng, P. Classification and Effects of Symmetry of Mechanical Structure and Its Application in Design. *Symmetry (Basel)* **2021**, *13*, doi:10.3390/sym13040683.
27. McComb, C.; Cagan, J.; Kotovsky, K. Mining Process Heuristics from Designer Action Data Via Hidden Markov Models. *Journal of Mechanical Design* **2017**, *139*, doi:10.1115/1.4037308.
28. Vogel, S.; Arnold, P. Towards a More Complete Object-Orientation in Graph-Based Design Languages. *SN Appl Sci* **2020**, *2*, doi:10.1007/s42452-020-2959-x.
29. Kolbeck, L.; Vilgertshofer, S.; Abualdenien, J.; Borrmann, A. Graph Rewriting Techniques in Engineering Design. *Front Built Environ* 2022, *7*.

30. Daly, S.R.; Yilmaz, S.; Christian, J.L.; Seifert, C.M.; Gonzalez, R. Design Heuristics in Engineering Concept Generation. *Journal of Engineering Education* **2012**, *101*, 601–629, doi:10.1002/j.2168-9830.2012.tb01121.x.
31. Yilmaz, S.; Daly, S.R.; Seifert, C.M.; Gonzalez, R. Evidence-Based Design Heuristics for Idea Generation. *Des Stud* **2016**, *46*, 95–124, doi:10.1016/j.destud.2016.05.001.
32. IEEE Computational Intelligence Society; Institute of Electrical and Electronics Engineers *IEEE Conference on Games 2020 : Online @cog2020.Onmicrosoft.Com (Previously Kindai University, Osaka, Japan) : August 24-27, 2020*; ISBN 9781728145334.
33. Bianchi, R.A.C.; Ribeiro, C.H.C.; Costa, A.H.R. Accelerating Autonomous Learning by Using Heuristic Selection of Actions. *Journal of Heuristics* **2008**, *14*, 135–168, doi:10.1007/s10732-007-9031-5.
34. Hulse, D.; Tumer, K.; Hoyle, C.; Tumer, I. Modeling Collaboration in Parameter Design Using Multiagent Learning. In *Design Computing and Cognition '18*; Springer International Publishing, 2019; pp. 577–593.
35. Cheng, C.-A.; Kolobov, A.; Swaminathan, A. Heuristic-Guided Reinforcement Learning. **2021**.
36. Johnston, I.G.; Dingle, K.; Greenbury, S.F.; Camargo, C.Q.; Doye, J.P.K.; Ahnert, S.E.; Louis, A.A.; Designed, A.A.L.; Performed, A.A.L. EVOLUTION BIOPHYSICS AND COMPUTATIONAL BIOLOGY Symmetry and Simplicity Spontaneously Emerge from the Algorithmic Nature of Evolution. **2022**, doi:10.1073/pnas.2113883119/-/DCSupplemental.
37. Montoya, F.G.; Baños, R.; Alcayde, A.; Manzano-Agugliaro, F. Symmetry in Engineering Sciences II. *Symmetry (Basel)* 2020, *12*.
38. Mitra, N.J.; Guibas, L.J.; Pauly, M.; Zürich, E. *Partial and Approximate Symmetry Detection for 3D Geometry*;
39. Modrak, V.; Soltysova, Z. Exploration of the Optimal Modularity in Assembly Line Design. *Sci Rep* **2022**, *12*, doi:10.1038/s41598-022-24972-2.
40. Li, M.; Langbein, F.C.; Martin, R.R. Detecting Design Intent in Approximate CAD Models Using Symmetry. *CAD Computer Aided Design* **2010**, *42*, 183–201, doi:10.1016/j.cad.2009.10.001.
41. Jiang, J.; Chen, Z.; He, K. A Feature-Based Method of Rapidly Detecting Global Exact Symmetries in CAD Models. *CAD Computer Aided Design* **2013**, *45*, 1081–1094, doi:10.1016/j.cad.2013.04.005.

42. Li, B.; Johan, H.; Ye, Y.; Lu, Y. Efficient 3D Reflection Symmetry Detection: A View-Based Approach. *Graph Models* **2016**, *83*, 2–14, doi:10.1016/j.gmod.2015.09.003.
43. Shi, Y.; Xu, X.; Xi, J.; Hu, X.; Hu, D.; Xu, K. Learning to Detect 3D Symmetry from Single-View RGB-D Images with Weak Supervision. *IEEE Trans Pattern Anal Mach Intell* **2023**, *45*, 4882–4896, doi:10.1109/TPAMI.2022.3186876.
44. Žalik, B.; Strnad, D.; Kohek, Š.; Kolingerová, I.; Nerat, A.; Lukač, N.; Podgorelec, D. A Hierarchical Universal Algorithm for Geometric Objects' Reflection Symmetry Detection. *Symmetry (Basel)* **2022**, *14*, doi:10.3390/sym14051060.
45. Gao, L.; Zhang, L.-X.; Meng, H.-Y.; Ren, Y.-H.; Lai, Y.-K.; Kobbelt, L. PRS-Net: Planar Reflective Symmetry Detection Net for 3D Models. **2019**, doi:10.1109/TVCG.2020.3003823.
46. Sipiran, I.; Gregor, R.; Schreck, T. Approximate Symmetry Detection in Partial 3D Meshes. *Computer Graphics Forum* **2014**, *33*, 131–140, doi:10.1111/cgf.12481.
47. Buric, M.; Bosner, T.; Skec, S. A Framework for Detection of Exact Global and Partial Symmetry in 3D CAD Models. *Symmetry (Basel)* **2023**, *15*, doi:10.3390/sym15051058.
48. Nguyen, T.P.; Truong, H.P.; Nguyen, T.T.; Kim, Y.G. Reflection Symmetry Detection of Shapes Based on Shape Signatures. *Pattern Recognit* **2022**, *128*, doi:10.1016/j.patcog.2022.108667.
49. Zhou, Y.; Liu, S.; Ma, Y. NeRD: Neural 3D Reflection Symmetry Detector. **2021**.
50. Riesen, K. *Advances in Computer Vision and Pattern Recognition Structural Pattern Recognition with Graph Edit Distance Approximation Algorithms and Applications*;
51. Crouse, D.F. On Implementing 2D Rectangular Assignment Algorithms. *IEEE Trans Aerosp Electron Syst* **2016**, *52*, 1679–1696, doi:10.1109/TAES.2016.140952.
52. Nasrollahi, A. Optimum Shape of Large-Span Trusses According to AISC-LRFD Using Ranked Particles Optimization. *J Constr Steel Res* **2017**, *134*, 92–101, doi:10.1016/j.jcsr.2017.03.021.
53. Ranjan, R.; Grover, S.; Medya, S.; Chakravarthy, V.; Sabharwal, Y.; Ranu, S. *GREED: A Neural Framework for Learning Graph Distance Functions*;
54. Labrou, K.; Kotsopoulos, S.D. Making Grammars for Computational Lacemaking. In *Design Computing and Cognition'22*; Springer International Publishing, 2023; pp. 587–604.