ASME Journal of Mechanical Design
Online journal at:
https://asmedigitalcollection.asme.org/mechanicaldesign

Discrete Structural Design Synthesis: A Hierarchical-Inspired Deep Reinforcement Learning Approach Considering Topological and Parametric Actions

Maximilian E. Ororbia[1]
Weitzman School of Design,
University of Pennsylvania,
Philadelphia, PA 19146
e-mail: mororbia@upenn.edu

Gordon P. Warn
Department of Civil Engineering,
The Pennsylvania State University,
University Park, PA 16802
e-mail: gpw1@psu.edu

Structural design synthesis considering discrete elements can be formulated as a sequential decision process solved using deep reinforcement learning, as shown in prior work. By modeling structural design synthesis as a Markov decision process (MDP), the states correspond to specific structural designs, the discrete actions correspond to specific design alterations, and the rewards are related to the improvement in the altered design's performance with respect to the design objective and specified constraints. Here, the MDP action definition is extended by integrating parametric design grammars that further enable the design agent to not only alter a given structural design's topology, but also its element parameters. In considering topological and parametric actions, both the dimensionality of the state and action space and the diversity of the action types available to the agent in each state significantly increase, making the overall MDP learning task more challenging. Hence, this paper also addresses discrete design synthesis problems with large state and action spaces by significantly extending the network architecture. Specifically, a hierarchical-inspired deep neural network architecture is developed to allow the agent to learn the type of action, topological or parametric, to apply, thus reducing the complexity of possible action choices in a given state. This extended framework is applied to the design synthesis of planar structures considering both discrete elements and cross-sectional areas, and it is observed to adeptly learn policies that synthesize high performing design solutions.
[DOI: 10.1115/1.4065488]

Keywords: design process, machine learning

1 Introduction

Design can be regarded as a problem-solving task where a solution is determined by creating and searching through a space of possible alternatives [1]. Computational design synthesis further develops upon this problem-solving task by using various approaches to generate a design space and applying computational systems to automate the synthesis and evaluation of the candidate solutions [2–8]. Viewing design as a problem-solving task also naturally lends itself to machine learning approaches, in particular, reinforcement learning (RL)—an area in machine learning focusing on the behavioral learning of an agent that interacts with a dynamic environment by taking actions in order to maximize a cumulative numerical reward [9–11]. This work contributes to the aim of formulating design as a sequential decision process solved by reinforcement learning.

There exist a variety of computational design tools that employ machine learning to automate design tasks and/or aid designers, for example, by generating and evaluating design alternatives [12–15], exploring design spaces [16], optimizing design solutions [17–19], and mimicking the human design process [20–22]. Here, the application of design synthesis is focused on structures with discrete geometries and discrete design variables, such as, frames and trusses, which in some engineering disciplines are limited to being designed using standardized elements of given shape and cross-sectional area. In this context, several reinforcement learning-based approaches have been developed to either design the geometry [23,24], assign discrete parameters to predefined structural forms [25], or adjust both structural shape and element size [26]. While each approach has particular aims and applications, this work

[1]Corresponding author.

Contributed by Design Automation Committee of ASME for publication in the Journal of Mechanical Design. Manuscript received January 4, 2024; final manuscript received April 22, 2024; published online June 3, 2024. Assoc. Editor: Christopher Mccomb.

presents a framework and a unique deep reinforcement learning (DRL) network architecture for the discrete design of structural geometry and material layout.

In Ororbia and Warn [27,28], the design synthesis of discrete structures considering only topological actions, i.e., altering the topology of a given configuration, was modeled as a Markov decision process (MDP) and solved using a tabular reinforcement learning algorithm in order to validate the overall framework, leveraging tabular RL's strong convergence properties. To apply the MDP framework to larger, more representative design synthesis problems, Ororbia and Warn [29] integrated deep reinforcement learning into the framework through the development of a deep neural network (DNN) that served as an approximator for the action-value function. The benefit of the integrated DNN in the MDP framework is that it has far fewer parameters than the dimensionality of the design problem's state and action space cardinalities and, as such, can adeptly solve design problems with significantly large state and action spaces, noting that in spite of the use of the DNN, each design configuration is represented precisely. In this paper, the MDP framework action definition is extended to include a parametric action type so as to realize the design synthesis of discrete structures considering both topological (geometric) and parametric (size) alterations, i.e., generating both the topology and parameters of structures with discrete elements and discrete cross-sectional properties. Through topological and parametric actions, the agent is capable of altering both the placement of elements and nodes as well as modifying the specific properties of the elements being generated.

The development and integration of a parametric action type here introduce additional complexity to the state and action spaces; in particular, it exponentially increases their cardinality and introduces diversity to the type of action that is available to the agent in a given state. While the prior fully connected, feed-forward network DQN implemented for designing the topology of truss and frame structures performed well [29], the integration of parametric operators in the action definition necessitates a more refined DRL architecture to efficiently handle the corresponding increase in both the dimensionality of the problem and the complexity of the learning task. As such, the DRL architecture presented in Ororbia and Warn [29] is significantly extended in this paper to adeptly handle the increased action complexity while also properly abstracting the relevant features of a given state since elements are now also assigned specific parameters. A hierarchical-inspired deep reinforcement learning (HDRL) approach is developed to address the challenges of high-dimensional and complex state and action spaces associated with discrete structural design synthesis considering both topological and parametric actions. The HDRL architecture uses a separate controller network to determine which type of action to apply when in a given state and utilizes both dueling and branching deep state-action networks enabling the agent to learn how to apply specific topological and parametric action alterations, especially when the policy requires many state transitions and different alteration types.

Hence, this paper presents several important developments in the DRL neural network architecture that inform the agent of a given state's characteristics, which is beneficial given that discrete element properties are accounted for as design variables. The suggested extensions also facilitate a robust approach for applying a sequence of intricate topological and parametric actions. Since the underlying elements of the MDP framework pertain to the alteration of discrete structures with specified performance metrics, the framework is relevant and applicable in other disciplines, including, for example, aircraft [30], composite [31,32], and gearbox [33,34] design in the disciplines of civil, aerospace, materials, and mechanical engineering, respectively. The extended MDP framework is evaluated and applied here to the design of planar truss structures with discrete elements and discrete cross-sectional design variables with the objective of maximizing the cumulative rewards that simultaneously minimizes displacement at a specified node(s) in the design domain subject to stability and volume constraints. Here the parametric action type considers discrete cross-sectional areas. For each example, the agent's design solution, final synthesized structure, is compared to the design determined by either conventional discrete or continuous optimization methods.

2 Modeling Discrete Structural Design Synthesis as a Markov Decision Process

A Markov decision process can provide a solid mathematical framework for modeling the design synthesis of structural systems considering discrete elements and discrete design variables. MDPs belong to a class of discrete-time stochastic control processes based on the theory of multi-stage decision processes, or sequential decision processes, and offer the ability to formulate and accommodate general goals, objectives, and constraints [11]. Hence, given its flexibility, the design synthesis of structural systems can be mathematically modeled as an MDP by considering structural systems as finite graphs and actions as alterations made to the graphs. Specifically, alterations made to the connectivity of the graph, changing the structural topology, and to the properties of the edges, changing, for example, the volume allocated to a given element in the domain.

An undiscounted finite MDP is defined by a four-element tuple: $(\mathcal{S}, \mathcal{A}, \mathcal{P}(s_{t+1} \mid s_t, a_t), \mathcal{R}(s_t, a_t))$, where $\mathcal{S}$ is a finite set of all the possible decision states s_t, $\mathcal{A}$ is a finite set of all valid actions a_t, $\mathcal{P}(s_{t+1} \mid s_t, a_t)$ is the probability of transitioning to the next state s_{t+1} from s_t by taking action a_t, and $\mathcal{R}(s_t, a_t)$ is the immediate reward from s_t after action a_t. In the context of discrete structural design synthesis with the objective of minimizing displacement subject to stability and volume constraints, an MDP state, s_t, represents a given design configuration at time t. In this work, each design configuration in a given state s_t is represented as a finite graph $\mathbb{G}$, defined by the connectivity of a set of vertices and edges as well as the parameters assigned to each edge. An action, $a \in \mathcal{A}(s_t)$, is an alteration made to a given state s_t configuration that results in the generation of a new design configuration (structure), forming the next state, s_{t+1}. Here, in the context of design synthesis, once the agent takes a given action, the design deterministically transitions to the next state. For a given design synthesis problem, the goal state, i.e., the design with the least displacement that satisfies the constraints, is unknown a priori. Therefore in Ororbia and Warn [29], a conditional reward function was developed that encourages the agent to revisit states it has encountered throughout its experience that have the best performance. These states serve as estimates of the problem's goal state until a better performing solution is later synthesized.

The rewards are generated and assigned by a reward function, $\mathcal{R}(s_t, a_t)$, and for the design problem considered here, the reward is computed as follows:

$$\mathcal{R}(s_t, a_t) = \begin{cases} \beta_r(\Delta(s_t) + \Delta(s_0)), & u_{t+1} \leq u_{\min} \\ \beta_r(\Delta(s_t)), & s_t \ (\textit{continuing}) \\ 0, & s_T \end{cases} \tag{1}$$

where u is a given design's displacement at a particular nodal location(s), Δ is the change in displacement between two states, and β_r is a parameter that scales the reward value. For continuing states, the reward is proportional to the change in displacement u at a given nodal location(s) of the design configuration in s_t with the altered configuration's displacement in s_{t+1}, i.e., $\Delta(s_t) = u(s_t) - u(s_{t+1})$. Since the best performing design (goal state) is not known a priori, the agent is given a "bonus" reward when it visits a state in which the displacement is less than a prior minimum encountered, $u_{\min}$. Specifically, the agent receives a reward that is proportional to the difference between the seed configuration's displacement s_0 and the configuration's displacement in s_{t+1}, or, formally, $\Delta(s_0) = u(s_0) - u(s_{t+1})$. If the agent transitions to a terminal state, s_T, then it is assigned a zero reward. Since the reward's magnitude is in part dependent on the structural configuration's displacement, the β_r coefficient is specified to scale the reward value. In particular, the change in displacements can be quite small so they

are scaled so as to have a meaningful impact on the network weights. For the design synthesis application considered here, the structural performance has been normalized, hence the coefficient is set to $\beta_r = 10$ for all examples.

To further broaden the applicability of modeling the design synthesis of discrete structures as an MDP, a parametric action type is formulated and integrated into the action definition introduced in Ororbia and Warn [28,29]. Specifically, to enable the agent to generate both the discrete topology and assign discrete parameters, the MDP action definition is extended to include both available topological, $\mathcal{A}_{topo}(s_t)$, and parametric, $\mathcal{A}_{parm}(s_t)$, action sets, which are state s_t dependent. Hence, the set of actions for a given state $\mathcal{A}(s_t)$ is defined as follows:

$$\mathcal{A}(s_t) = \{\mathcal{A}_{topo}(s_t) = (n_i, m_j, o_k),\ \mathcal{A}_{parm}(s_t) = (m_j, P)\} \tag{2}$$

where the set of available topological actions $\mathcal{A}_{topo}$ are a function of the state's available nodes n_i, existing elements m_j, and applicable operators o_k. Furthermore, the set of available parametric actions $\mathcal{A}_{parm}$ are a function of the states existing elements m_j and applicability of the P operator. By extending the action definition, the agent is capable of altering both the topology (geometry) and parameters (assigned cross-sectional parameters) of a structural configuration in a given state. The topological action function as introduced in Ororbia and Warn [28,29], presented below for convenience, is formally defined as follows:

$$\mathcal{A}_{topo}(s_t) = \{(n_i, m_j, o_k) \mid n_i \in N_{avail.}(s_t),\ m_j \in M_{allow}(s_t, n_i),\ o_k \in O_{legal}(s_t, n_i, m_j)\} \tag{3}$$

where the topological action set $\mathcal{A}_{topo}(s_t)$ is described by three components: (1) the selection of an available inactive node n_i from a set of available inactive nodes, $N_{avail.}$, in the design domain, (2) the selection of a member m_j from a set of eligible members, M_{allow}, that are a part of the current design configuration, and (3) the choice of an operator o_k from a set of applicable, legal operators, O_{legal}, that can add and/or remove elements. The set of inactive nodes that are not a part of a given design configuration is referred to as the design domain Ω. The reader is referred to Ororbia and Warn [28] for further details pertaining to specific action rules that prevent the synthesis of redundant configurations from occurring.

The suggested parametric action type is inspired by generative design grammars [7,34,35], and is specifically formulated here to assign discrete cross-sectional area values to a given element, although it could be generalized further to consider additional discrete section properties, such as moment of inertia, or other material properties. Since the overall MDP framework is a generative sequential task, the number of structural elements increase after each state transition, hence the consideration of multiple discrete cross-sectional areas causes a combinatorial growth in the number of possible area actions that can be assigned to a given structural configuration if each possible element-area combination is considered. Specifically, this is proportional to the dimension of the number of discrete areas specified in a set $\mathbf{A}$, raised to the power of the total number of eligible elements, M_{allow}, that make up a given configuration; mathematically represented as $\dim(\mathbf{A})^{M_{allow}}$. Therefore, to control the growth in the size and diversity of available actions an agent can take in a given state, the parametric action is designed such that it incrementally increases the cross-sectional area of a single selected element to the next largest value in the specified set of areas for a given configuration. In general, this approach reduces the total number of applicable actions in a given state to exactly the number of elements that make up that configuration, yet does not limit the agent's ability to synthesize any feasible designs.

Formally, a parametric action is defined as

$$\mathcal{A}_{parm}(s_t) = \{(m_j, P) \mid m_j \in M_{allow}(s_t),\ \exists P : A_j < \max(\mathbf{A})\} \tag{4}$$

where the parametric action set $\mathcal{A}_{parm}(s_t)$ is defined by two components: (1) the selection of a member m_j from a set of eligible elements M_{allow} that exist in a state s_t, and (2) the application of a P operator given that the selected m_j element's area is less than the maximum value in the set of specified areas $\mathbf{A}$. For a given state, if all element areas are assigned the maximum value in the specified area set, then the agent can not take a parametric action. If the selected element's area is less than the maximum area in the set, a P operator can be applied, incrementally increasing the selected element's current assigned parameter to the next largest value in the specified set of parameters.

Illustrative examples of the application of both parametric and topological actions defined in Eq. (2) are presented in Fig. 1. In particular, Fig. 1(*a*) depicts how, for a given seed configuration with all elements initially assigned the minimum cross-sectional areas considered in the specified set $\mathbf{A}$, there are three available parametric actions. These parametric actions are available for this specific state since there are three elements m_1, m_2, and m_3, all assigned with the minimum cross-sectional area A_1. Action a_1 selects

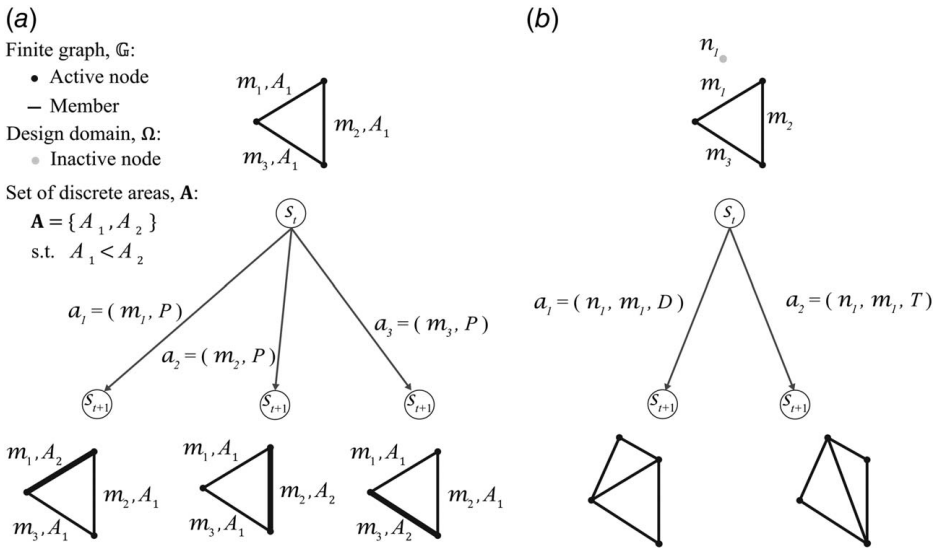

Fig. 1 Example of (*a*) parametric actions: representation of a finite graph at a state s_t and the application of three applicable parametric actions, a_1, a_2, and a_3, and the associated new states s_{t+1} with increased element areas indicated by the thicker line width and (*b*) topological actions: adapted from Ororbia and Warn [28,29] altering the finite graph's geometry

element m_1 and applies a P operator, which incrementally increases the element's area from A_1 to next largest value in the set $\mathbf{A}$, A_2, and transitions the agent from state s_t to state s_{t+1}. Similarly, actions a_2 and a_3 selects elements m_2 and m_3, respectively, and applies a P operator, which increases the elements' area from A_1 to A_2, and transitions the agent from state s_t to state s_{t+1}. In Fig. 1(*b*), all possible topological actions that can be taken considering a design domain with a single inactive node are illustrated, which also shows the effect of applying D and T operators. A T operator removes a selected element and connects a newly activated node, selected from available inactive nodes in the domain, with other active nodes. A D operator activates a new node and connects it to the current configuration without removing an element.

In general, both topological and parametric actions are available to the agent in a given state. The definition of terminal state, s_T, is extended from Ororbia and Warn [28]. In particular, if there are no available topological and parametric actions then the state is considered terminal. In the context of design synthesis of discrete structures, a state is also considered terminal if a specified constraint is violated. Additionally, since the overall MDP framework represents design as a generative task, the seed configuration, s_0, is specified as a sparsely connected structure with discrete elements, each initially assigned the minimum area value considered in the discrete set, allowing for all possible area combinations to be potentially considered throughout the design synthesis process. Also, when a topological action is taken, the newly generated elements are assigned with the minimum area value in the specified set.

3 A Hierarchical-Inspired Deep Reinforcement Learning Approach for Discrete Design Synthesis

While the initial deep Q-network implementation for topological design synthesis in Ororbia and Warn [29] performed well overall, the inclusion of parametric actions substantially increases the size of both the state and action spaces. Not only does the number of actions that are available to the agent in a given state increase, but the policy required to generate high performing design solutions will invariably entail an intricate sequence of actions, varying in type. Furthermore, individual states can no longer be represented by their topologies alone, since each element in a given configuration is assigned a specific cross-sectional parameter. These extensions to the MDP necessitate a refined DQN architecture so as to effectively and adeptly solve the discrete structural design synthesis learning task with parametric actions. As such, the previously implemented DQN is extended to include a new state feature representation and combines several approaches to inform the agent of what action type to apply when in a given state, to guide the agent experiencing and applying intricate topological actions, and to aid the agent in converging to the best performing design solution that it encounters during training.

A hierarchical network architecture that is inspired from feudal reinforcement learning [36,37] is developed in order to reduce the networks' output dimensionality, to increase the diversity of actions experienced by the agent during training, and to better inform the agent on when to apply either a topological or a parametric action. Specifically, three separate networks are designed to follow a hierarchy and are assigned individual "roles", or sub-tasks, with respect to the action space. One network, referred to as the controller network, is trained to learn what specific action type to apply when in a given state. The other two networks are associated with either learning how to apply specific topological actions, defined in Eq. (3), or parametric actions, defined in Eq. (4), and both networks are referred to by the specific action type that they learn to apply.

For problems with large action spaces, dueling network architectures [38] have been shown to be effective at discerning valuable states without having to learn the value of each action under each

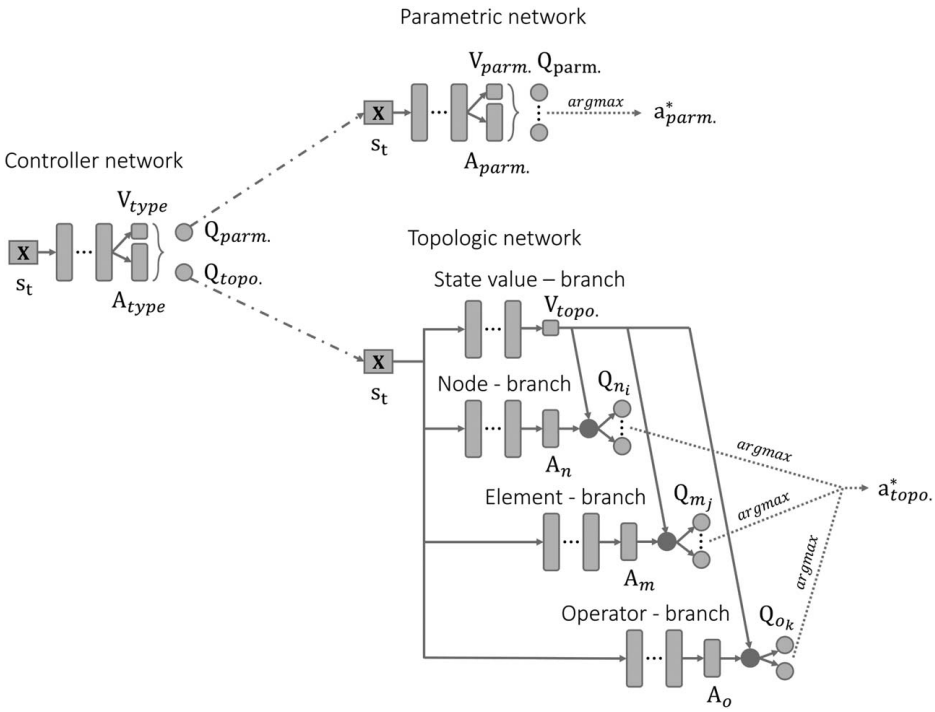

Fig. 2 Illustration of the HDRL architecture developed for the discrete design synthesis MDP framework. The controller network is trained to approximate the state-action value function associated with the type of action that the agent can take in a given state and dictates whether that action is taken. The parametric and topological networks are trained to approximate the state-action values associated with their respective specific action. Both the controller and parametric networks utilize a dueling DQN and the topologic network implements a branching dueling Q-network.

state, achieving high-quality policies [39]. Hence, in this work a dueling DQN approach is implemented for each component network. Additionally, the agent's ability to experience a diverse number of topological alterations, as it did when solely being trained to design the topology of configurations, decreases, since it is able to select between taking either topological or parametric actions. Therefore, to assist the agent in experiencing more diverse topological actions during training, an action branching DRL approach [39,40] is also developed and implemented for the topologic network. Finally, since the agent will experience a variety of different action types throughout its training, this work suggests a simple mechanism based on prioritized experience replay [41] to enable the agent to converge to the design configuration with the best performance it has encountered during training. Figure 2 provides an illustration of the HDRL architecture, including the extensions and various approaches discussed as a unified DRL architecture, referred to herein as the MDP-HDRL framework. Additional details pertaining to the architecture's particular components are discussed in the subsequent subsections.

3.1 Feature Representation. In the context of the design synthesis of discrete structures considering both the topological and parametric actions, a state s_t can be described by its active elements and the parameters assigned to those elements. The feature set defined in Ororbia and Warn [28] is insufficient to distinguish between states when only a parametric action is taken. This is due to the fact that the geometry, element connectivity, remains the same during the state transition, but the specific parameter assigned to an element has changed. Thus to inform the agent with an accurate description of a given state, the specific feature values, represented as an input vector $\mathbf{x}$, indicate the presence of existing elements and the specific parameter values assigned to those elements. For the numerical examples presented in Sec. 4, the parameter values assigned to elements are associated with discrete cross-sectional areas chosen from a specified set $\mathbf{A}$. To assist with the training and performance of the networks in the MDP-HDRL framework, the input features for the topologic and parametric networks are standardized through binary encoding and normalized respectively.

3.2 Hierarchical Deep Reinforcement Learning. If the state-action values associated with parametric actions are directly considered as outputs in the prior DQN architecture [29], several challenges are introduced. The number of actions that the agent can select from, as well as their associated state-action output Q-values, substantially increase. This increase also varies throughout the episodic task; specifically as the agent takes topological actions that synthesize configurations with more elements, more parametric actions become available. Furthermore, the agent may not experience taking parametric actions early in the design synthesis process since there are generally fewer elements in the initial structural configurations and more topological actions are available due to the presence of more inactive nodes in the design domain. Also, having the agent capable of selecting among both topological and parametric actions can affect the model training, potentially confusing the agent on what type of action to take in a given state. These challenges can be addressed by separately training networks to identify the action type to take and how to apply the specific action. To overcome these challenges, the approach taken here is inspired from feudal reinforcement learning [36,37] by applying a hierarchy to the learning task, where the task is divided into subtasks that are managed by a controller. In certain applications, hierarchical architectures can lead to improved exploration through increased and diversified experiences enabled by the controller directly selecting between both action types and faster learning since individual networks are focused on learning their sub-tasks. In the context of discrete design synthesis, topological and parametric actions are represented as sub-tasks and a controller network is implemented to determine the action type to apply when in a given state.

As illustrated in Fig. 2, a controller network is used to determine the state-action values associated with taking a topological or parametric action, Q_{topo} and Q_{parm} respectively. Depending on the action selected by the agent, determined by using the controller network, the topological or parametric state-action values are approximated using either the associated topologic network or parametric network. After the selection of the action type and application of the corresponding topological or parametric actions, the loss and gradient information are calculated to update each of the network's weights using the reward function defined in Eq. (1) and introduced in Ororbia and Warn [29]. The network associated with the action type that is not applied is not updated during this transition. In general, this imposed hierarchy appropriately trains the controller network to make decisions on when to apply either a topological or parametric action. The separate topologic and parametric networks are also trained to apply the appropriate action from their defined sets without being influenced by the other action definition components. Details pertaining to each network are discussed in the following subsection.

3.3 On the Branching Dueling Q-Network. A dueling network architecture [38] is used for each of the component networks in the hierarchy since the discrete design synthesis problems considered have large action spaces and the feature set includes additional information describing the state's assigned parameters. Dueling networks offer an efficient alternative to estimating the value of every action for a certain state [39]. The dueling network augments the standard DQN network architecture by incorporating a layer prior to the output layer to separately estimate the state value function, $\mathrm{V}(s_t;\theta)$, and the action advantage function, $\mathrm{A}(s_t, a_t;\theta)$. Unlike a standard DQN, the dueling network's additional evaluation of the state value function enables the agent to better discern states that may have the same element connectivity but different assigned element parameters. Both the state value and action advantage functions are parameterized with weights θ. These two functions are then aggregated to produce the output layer state-action value function $Q(s_t, a_t;\theta)$.

Formally, the state-action value function $Q(s_t, a_t;\theta)$ is determined by combining the state value branch $\mathrm{V}(s_t;\theta)$ and the advantage branch $\mathrm{A}(s_t, a_t;\theta)$ as

$$Q(s_t, a_t;\theta) = \mathrm{V}(s_t;\theta) + \mathrm{A}(s_t, a_t;\theta) - \frac{1}{\dim\left(\mathcal{A}(s_t)\right)} \sum_{a_{t+1}} \mathrm{A}(s_{t+1}, a_{t+1};\theta) \tag{5}$$

where $\mathcal{A}(s_t)$ is the set of actions corresponding to the specific network used in the hierarchy. In general, this network augmentation enables the agent to be more effective at discerning valuable states without having to learn the value of taking each action for each state, ultimately resulting in high-quality policies [39]. However, as discussed, the number of actions experienced by the agent during training is reduced since it can choose between taking topological and parametric actions. While there exist a variety of exploration techniques [42], they introduce additional challenges associated with data storage and can not be readily implemented due to the discrete and unique nature of the actions considered. Accordingly, to assist the agent in experiencing topological actions and learning which are best to apply in a given state, the dueling network architecture used for the topologic network is further augmented.

An action branching architecture, also referred to as a branching dueling Q-network (BDQ) [40], is adapted and implemented for the topologic network in the hierarchy. BDQs have been shown to effectively scale to learning tasks with increasing action dimensionalities and offer an approach for the agent to better explore the components that make up the topological action defined in Eq. (3). The adapted BDQ used for the topologic network, illustrated in Fig. 2,

assigns sub-networks to each of the individual topological action components, selection of an inactive node n_i, selection of an eligible element m_j, and application of an operator $o_k = \{T, D\}$. These sub-networks, or branches, are referred to here with respect to the specific topological action component that they estimate in the advantage function, for example, the branch that estimates advantage function A_n associated with the selection of an inactive node is referred to as the "node-net." A common state value V_{topo} is estimated and aggregated with each of the sub-network's advantage functions to determine their respective state-action values using Eq. (5). A topological action a_{topo} is determined by combining the particular components selected in each of the sub-networks, where the action is defined as $a_{topo} = (n_i, m_j, o_k)$. After the action is applied and the reward is received, the BDQ is updated via backprop using the following loss function:

$$L = \frac{1}{N}\sum_{d}\left(r(s_{t+1}) + \gamma \max_{a_d \in \mathcal{A}_{\text{topo}}(s_{t+1})} Q(s_{t+1}, a_{t+1}; \theta_d) - Q(s_t, a_t; \theta_d)\right)^2 \tag{6}$$

where N is the number of action components, which, for the defined topological action $N = 3$, and d refers to the specific action component considered. Since each sub-network is associated with a specific applicable topological action component, during training, where an ε-greedy policy is used, the agent will randomly select from each of the individual action components, thus increasing the variety of topological actions explored. In the broader context of the hierarchy, this is beneficial since, when the controller network decides to use a topological action during training, the agent has an increased chance of applying a topological action that it has not yet experienced.

3.4 Episodic Replay for Discrete Design Synthesis. As a design learning task, the agent's objective is to learn a policy that maximizes its accumulated reward, which synthesizes a high performing design solution. The increased state and action spaces that come as a result of this paper's design problem formulation, would suggest that the agent be trained for an extended period of time in order to learn the optimal policy. However, to enable more efficient learning and convergence to the policy that synthesizes the high performing design solution, an approach to update the network parameters is suggested that is inspired from prioritized experience replay [41]. In particular, to assist the agent in converging to the policy that synthesizes the design with the best performance that it has experienced during training, a memory bank, or buffer, is used to store the trajectory, i.e., the set of transitions, associated with the policy that synthesized the best performing design solution observed so far. The buffer is continually wiped and updated with a new trajectory during training whenever the agent synthesizes a design that outperforms the design currently stored in the "best trajectory" buffer. At a specified point in training, in general, when the ε-greedy exploration probability has decayed significantly, the agent separately updates its HDRL network weights using the trajectory stored in the buffer, which is referred to here as optimal experience replay updates.

4 Numerical Examples

This section applies the MDP-HDRL framework to four different design synthesis numerical examples, each involving the design of a planar truss structure. The design problem's objective is to synthesize a truss considering both topology generation and assigned cross-sectional areas that maximizes the cumulative reward and simultaneously minimizes the displacement at a specified noda location(s) for a given external force(s), subject to stability and volume constraints. For all examples, every structural configuration that is synthesized is represented as a finite graph and the length of each element is determined by its nodal connectivity. In the subsequent figures, the nodes marked with triangles represent the pin supporting points and the nodes with arrows represent the location of the applied force(s). In all examples, the displacement for synthesized structures is evaluated at the nodal location of the applied force. For simplicity, structural parameter values are assigned such that the results of the linear finite element analysis are nondimensional. With respect to the planar trusses, element local stiffness matrices, k_{ele}, are assembled assuming an elastic modulus of $E = 10^3$. A set of discrete cross-sectional areas $\mathbf{A}$ is specified for each example.

4.1 Experimental Setup. The HDRL architecture, illustrated in Fig. 2, is used in an online setting for each example. To evaluate the MDP-HDRL framework, five independent experimental trials are conducted. For each trial conducted, the agent is trained starting from a different, random initialization of network weights sampled from a uniform distribution. Each network has two hidden, fully-connected layers with hyperbolic tangent (tanh) activation functions. By considering network capacity and computational resources, all hidden layers have 128 neurons and the output layer for each has a number of neurons equal to the total number of possible actions associated with the respective network. For example, the controller network has two outputs respectively associated with the value function of either taking a parametric action or a topologic action. Since not all actions are valid, the output layer for each network is masked based on the given state such that the agent is only able to select from a set of output Q-values related to applicable actions. The network parameters (θ) are updated using root mean squared propagation (RMSprop) optimization [43] with a learning rate of 0.001.

The agent follows an ε-greedy behavioral policy where the exploration rate for each trial is linearly decayed over a specified window of episodes, and varies between examples since each has different state and action space cardinalities. For all examples, the exploration probability is decayed from 90% exploration and 10% exploitation ($\varepsilon = 0.9$) to 5% exploration and 95% exploitation ($\varepsilon = 0.05$) until the last 1000 final training episodes (where the exploration probability is held constant at 5% for the remaining episodes). The optimal experience replay updates are applied to the last 10,000 episodes for each example. In examples 1 and 2, the agent is trained for 30,000 episodes, on average taking 1.8 and 2.3 h, respectively. The agents in examples 3 and 4 are trained for 50,000 episodes, respectively taking on average 5.3 and 6.2 h owing to the increase in the dimesionality of the state and action spaces. The computational times reported include initialization, training, inference, model evaluations, and pre/post-processing and are with respect to a DELL Precision 7920 Tower: Intel(R) Xeon(R) Platiunum 8168 CPU @ 2.70 GHz, 24 Cores with 0.5 TB RAM. The HDRL agent is evaluated based on its ability to learn an optimal policy and the final performance of the design solution synthesized according to that policy. Using a moving average over 100 points, the HDRL agent's learning is captured for each example by plotting the displacement of the final synthesized, non-terminal structural configuration after every ten episodes for all trials, referred to as the displacement performance.

4.2 Example 1: An Incline-Loaded Michell Truss. The design problem attributes and parameters, including the nodal design domain Ω, seed configuration s_0, support conditions, and externally applied loads f, are presented in Fig. 3(*a*). Two discrete cross-sectional areas $\mathbf{A} = \{3, 5\}$ are considered. Both a stability and volume constraint of $V_{cnst} = 750$ are considered.

The MDP-HDRL framework's performance is presented in Fig. 3(*b*) where the displacement of the final synthesized structural configuration associated with the ultimate, non-terminal state in the learned policy after every ten episodes for all trials, is plotted. The overall decrease in displacement performance in Fig. 3(*b*) illustrates that the agent is effectively learning policies that synthesize high performing design solutions with small displacements relative to the seed configuration.

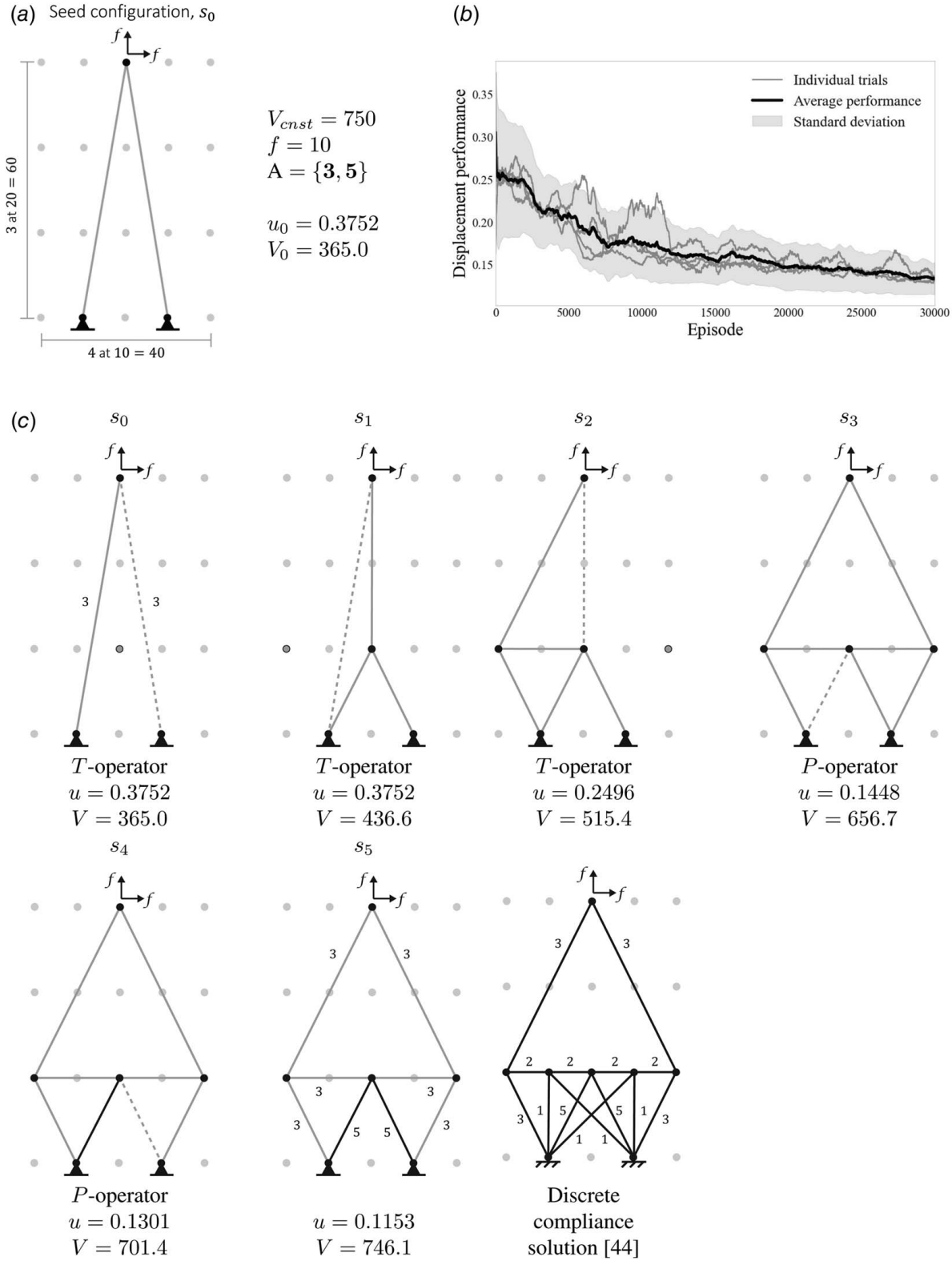

Fig. 3 (*a*) Example 1's 5 × 4 "Michell" design domain Ω, seed configuration s_0, specified volume constraint of $V_{cnst} = 750$, and set of two discrete cross-sectional areas $A = \{3, 5\}$, (*b*) the MDP-HDRL framework's performance based on the agent following a greedy policy with respect to the HDRL network's current approximation of the state-action value function. The displacements of the synthesized structure prior to the terminal state s_T for the five individual trials are plotted as well as their mean and standard deviation. The agent learns the policies that synthesize high performing design solutions as indicated by the decrease of the displacement performance, and (*c*) the MDP-HDRL framework agent's learned optimal policy where discrete cross-sectional areas are depicted by the elements color, specifically, the light gray and solid black elements represent $A = 3$ and $A = 5$, respectively. Action components taken at each state are indicated by highlighted domain node, dashed selected element, and listed operator underneath state illustration. The synthesized optimal design under state s_5 closely resembles the discrete solution determined in Achtziger and Stolpe [44].

The agent's learned optimal policy, determined for all trials conducted, is illustrated in Fig. 3(*c*). The areas assigned are depicted by the elements varying shades of gray, where light gray represents the minimum area in the set $A_1 = 3$ and solid black represents the largest area $A_2 = 5$. Numerical values of the identified element cross-sectional areas are also shown adjacent to the corresponding element in the final synthesized design in Fig. 3(*c*). The agent's learned policy demonstrates the MDP-HDRL framework's ability to appropriately discern and apply topological and parametric actions. In particular, the agent learns to first take a topological action which synthesizes the design configuration to establish the general load path, and then it learns to take parametric actions thereafter to increase the elements' areas, increasing the overall structural stiffness. In other words, the learned optimal policy first established an efficient load path, then allocates volume to elements in this path so as to minimize the displacement at the specified node. The agent's learned policy also resembles the traditional structural engineer's design process, in which a general layout is determined prior to detailing the structural elements.

As further qualitative validation, the synthesized design solution is compared to the discrete Michell solution in Achtziger and Stolpe [44]. While the agent's design solution can not be directly compared since the current MDP-HDRL framework does not consider crossing elements and varying support conditions, both subject of future work, the solutions closely resemble each other based on their topologies and general allocation of area. The MDP-HDRL framework's agent only evaluated an average of 2911 total model evaluations out of 1×10^5 design configurations in the state space. The complete state space was only generated to track the agent's performance and is not needed for the design synthesis learning task.

4.3 Example 2: A Single-Load Truss. This example's design problem attributes and parameters are presented in Fig. 4(*a*). The design domain and seed configuration considered for this example are the same as the one used in Ororbia and Warn [28]. However, here the design problem also includes a set of three discrete cross-sectional areas $\mathbf{A} = \{1, 2, 3\}$ for the agent to choose from. Additionally, the volume constraint is increased to $V_{cnst} = 220$ due to the ability of the agent to assign larger cross-sectional area values to individual elements. The stability constraint for this example is the same as in the prior. Due to the combinatorial increase in states, resulting from the inclusion of parametric choices, the feasible design space was not exhaustively evaluated due to the excessive computational effort required. Alternatively, the size of the state space was approximated based of the number of feasible designs generated using just topological actions and accounting for the average number of elements generated at each step. This approximation approach is also used for the subsequent examples. The state space for this example is made up of approximately 2×10^6 designs, which is a significant increase from an example with the same design domain and seed configuration that considered only topological actions and having a state cardinality of 1058. This significant increase demonstrates how the discrete design problem becomes more challenging when parametric actions are included, even when considering a modest number of discrete areas.

The MDP-HDRL framework's performance is presented in Fig. 4(*b*) where the displacement of the final synthesized structural configuration associated with the ultimate, non-terminal state in the learned policy after every ten episodes for all trials, is plotted. From Fig. 4(*b*), the agent effectively learns policies that synthesize high-performing design solutions, indicated by the decrease in displacement performance, relative to that of the seed configuration.

The agent's learned optimal policy, determined for all trials conducted, is illustrated in Fig. 4(*c*). The areas assigned are depicted by the elements varying shades of gray, where light gray represents the minimum area in the set $A_1 = 1$, dark gray represents $A_2 = 2$, and solid black represents the largest area $A_3 = 3$. Numerical values of the identified element cross-sectional areas are also shown adjacent to the corresponding element in the final synthesized design in Fig. 4(*c*). The agent's learned policy again demonstrates the MDP-HDRL framework's ability to appropriately discern and apply topological and parametric actions. In particular, the agent learns to first take a topological action which synthesizes the design configuration to establish the general load path, and then it learns to take only parametric actions thereafter to increase the elements' areas, increasing the overall structural stiffness. As further qualitative validation, the synthesized design solution is compared to the continuum solution determined using TopOpt [45]. The MDP-HDRL framework's agent only evaluated an average of 22,090 total model evaluations out of $\approx 2 \times 10^6$ design configurations in the state space.

4.4 Example 3: A Single-Load Bridge. The design problem attributes and parameters are presented in Fig. 5(*a*) for this example. The same design domain and seed configuration that were used in example 5 of Ororbia and Warn [28] are used here with the notable exception that this example considers two discrete cross-sectional areas $\mathbf{A} = \{1, 2\}$ (or two choices). Both a stability and volume constraint of $V_{cnst} = 430$ are considered. The state space for this example is made up of approximately 2×10^6 designs.

The MDP-HDRL framework's performance is presented in Fig. 5(*b*) where the displacement of the final synthesized structural configuration associated with the ultimate, non-terminal state in the learned policy after every ten episodes for all trials, is plotted. As observed in Fig. 5(*b*), the agent effectively learns policies that synthesize high performing design solutions, as indicated by the decrease in displacement performance, relative to that of the seed configuration.

The agent's learned optimal policy, determined for all trials conducted, is illustrated in Fig. 5(*c*). Again, the agent learns a policy that initially takes actions that alter the topology of the seed structure before taking parametric actions to increase element areas. The resulting design configuration in state s_6 also makes practical sense from a structural engineering perspective, where the outer elements, making up the top chord of the truss' arch, are assigned the larger areas. Again as further qualitative validation, the synthesized design solution is compared to the continuum solution in Fig. 5(*c*) determined using TopOpt [45]. Notably, the MDP-HDRL framework's agent evaluated an average of 18, 837 total model evaluations out of $\approx 2 \times 10^6$ design configurations in the state space.

4.5 Example 4: A Multiple-Load Truss. The design problem attributes and parameters, similar to example 3 in Ororbia and Warn [28], are presented in Fig. 6(*a*) for this example. This example considers two discrete cross-sectional areas $\mathbf{A} = \{1, 2\}$ choices. Both a stability and volume constraint of $V_{cnst} = 365$ are considered. The state space for this example is considerably large, made up of approximately 4×10^6 designs.

The MDP-HDRL framework's performance is presented in Fig. 6(*b*) where the displacement of the final synthesized, non-terminal structural configuration after every ten episodes for all trials, is plotted. As observed in Fig. 6(*b*), the agent effectively learns policies that synthesize high performing design solutions, as indicated by the decrease in displacement performance, relative to that of the seed configuration.

The agent's learned policy that synthesizes the design with the least displacement across all trials is illustrated in Fig. 6(*c*). Impressively, the agent is capable of learning this intricate policy which requires taking nine actions, the first three being topological actions. Also, the first topological action taken to transition the agent from s_0 to s_1 and the first parametric action taken to transition the agent from s_3 to s_4 synthesizes a design with worse displacement than in the prior state, demonstrating that the agent can adeptly learn policies that may require taking actions that initially result in a worse performing design in order to synthesize a higher performing design solution over longer time horizons. This policy was determined by the agent for 80% of the trials conducted. The other policy resulted in a configuration with a

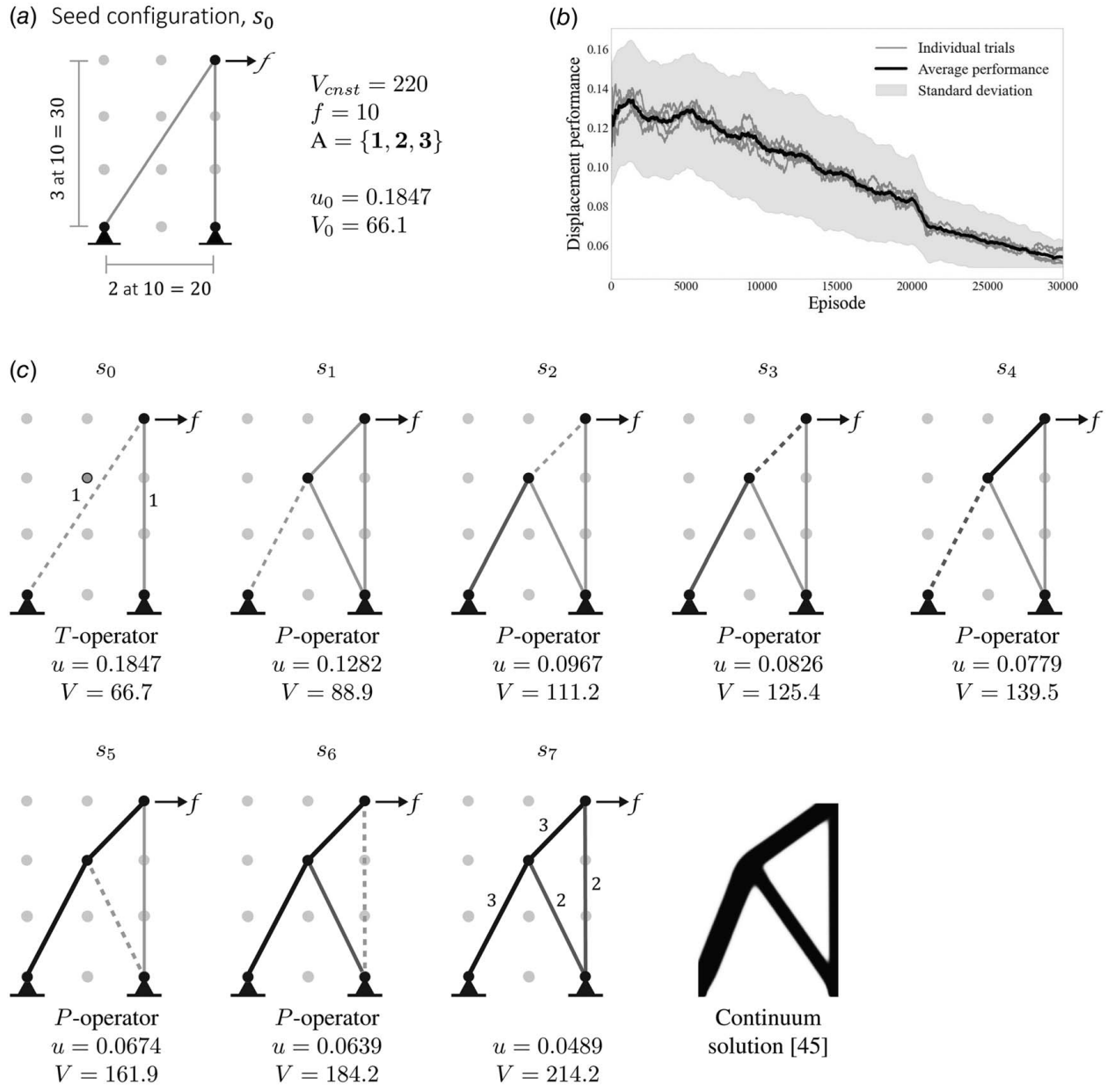

Fig. 4 (*a*) Example 2's 4 × 3 design domain Ω, seed configuration s_0, specified volume constraint of $V_{cnst} = 220$, and set of three discrete cross-sectional areas A = {1, 2, 3}, (*b*) the MDP-HDRL framework's performance based on the agent following a greedy policy with respect to the HDRL network's current approximation of the state-action value function. The displacements of the synthesized structure prior to the terminal state s_T for the five individual trials are plotted as well as their mean and standard deviation. The agent learns the policies that synthesize high performing design solutions as indicated by the decrease of the displacement performance, and (*c*) the MDP-HDRL framework agent's learned optimal policy. Action components taken at each state are indicated by highlighted domain node, dashed selected element, and listed operator underneath state illustration. Assigned discrete cross-sectional areas are depicted by the elements color, specifically light gray represents the minimum area in the set $A = 1$, dark gray represents $A = 2$, and solid black represents the largest area $A = 3$. The synthesized optimal design under state s_7 closely resembles the continuum solution, included to the right of s_7, determined using TopOpt [45]. Similar to the large area assignment to the left most elements in the design under state s_7, the continuum solution also assigns more volume to similar locations in the domain.

displacement of $u = 0.0198$ and volume $v = 348.5$ which is slightly greater than the displacement of the truss synthesized in state s_9 in the policy illustrated in Fig. 6(*c*) and significantly less than the seed's displacement. The synthesized design solution under state s_9 generally matches the continuum design solution determined using TopOpt [45], also shown in Fig. 6(*c*). The MDP-HDRL framework's agent notably evaluated an average of 75, 941 total model evaluations out of $\approx 4 \times 10^6$ design configurations in the state space.

5 Concluding Remarks

To further broaden the applicability of design synthesis of structures to include both discrete elements and discrete cross-sectional properties, this paper presents an MDP framework that considers both topological and parametric actions. This extension of the action definition also necessitated a significant refinement and development of the hierarchical deep reinforcement learning architecture inspired by feudal learning in order to address the increased

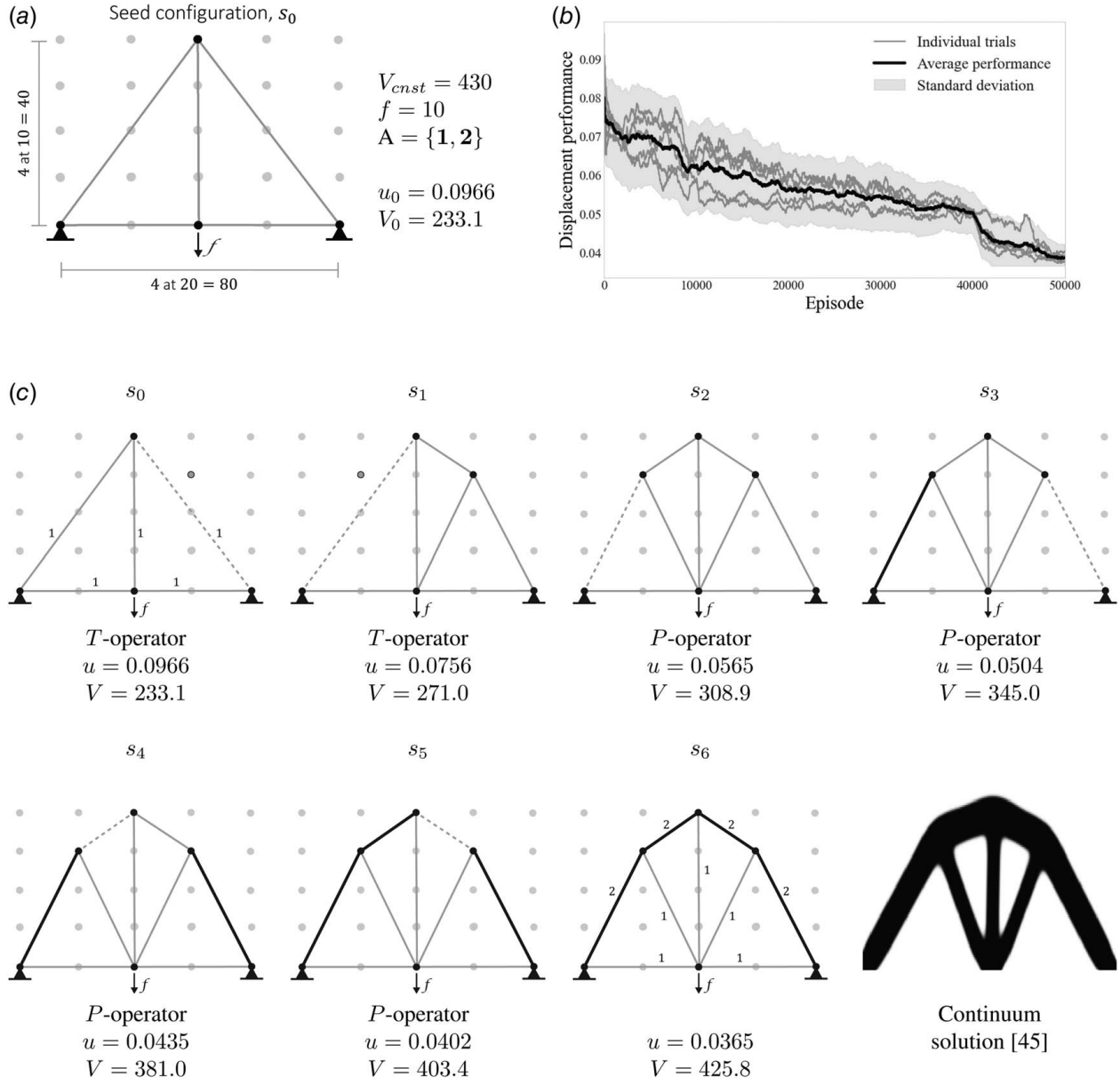

Fig. 5 (*a*) Example 3's 5 × 5 "bridge" design domain Ω, seed configuration s_0, specified volume constraint of $V_{cnst} = 430$, and set of two discrete cross-sectional areas A = {1, 2}, (*b*) the MDP-HDRL framework's performance based on the agent following a greedy policy with respect to the HDRL network's current approximation of the state-action value function. The displacements of the synthesized structure prior to the terminal state s_T for the five individual trials are plotted as well as their mean and standard deviation. The agent learns the policies that synthesize high performing design solutions as indicated by the decrease of the displacement performance, and (*c*) the MDP-HDRL framework agent's learned optimal policy. Assigned discrete cross-sectional areas are depicted by the elements color, specifically, light gray represents the minimum area in the set $A = 1$ and solid black represents the largest area $A = 2$. The synthesized optimal design under state s_6 closely resembles the continuum solution at the bottom right, determined using TopOpt [45]. The volume allocation in the continuum solution matches the element placement and area assignment (larger areas assigned to the outer chord, arch elements) in the design under state s_6.

size and action space complexity. A set of features were developed to accurately describe a given design configuration's topology and element parameters. Particular component networks were developed and assigned for determining the action type. The specific applied actions were also implemented with dueling deep Q-network structures in order to achieve efficient generalization over large state and action spaces. To enable the agent to experience a variety of topological actions during training, a branching action neural structure was formulated for the topological network within the modular, hierarchical model. Lastly, a prioritized experience replay inspired approach, which is referred to as the optimal experience replay, was developed to aid the agent in learning and converging to the policy that synthesizes the best performing design that it experiences throughout training.

Through the application to the design synthesis of discrete planar truss structures, considering parametric and topological actions, it was observed that the MDP-HDRL framework performed well, i.e., synthesized high performing designs that qualitatively agreed with those obtained from analogous discrete and continuum problems, for each of the different numerical examples with significantly large state space cardinalities. For all examples, the learned optimal policy first establishes an efficient load path, then allocates volume to elements in this path so as to minimize the displacement at the specified node. This sequence of establishing the load path, then

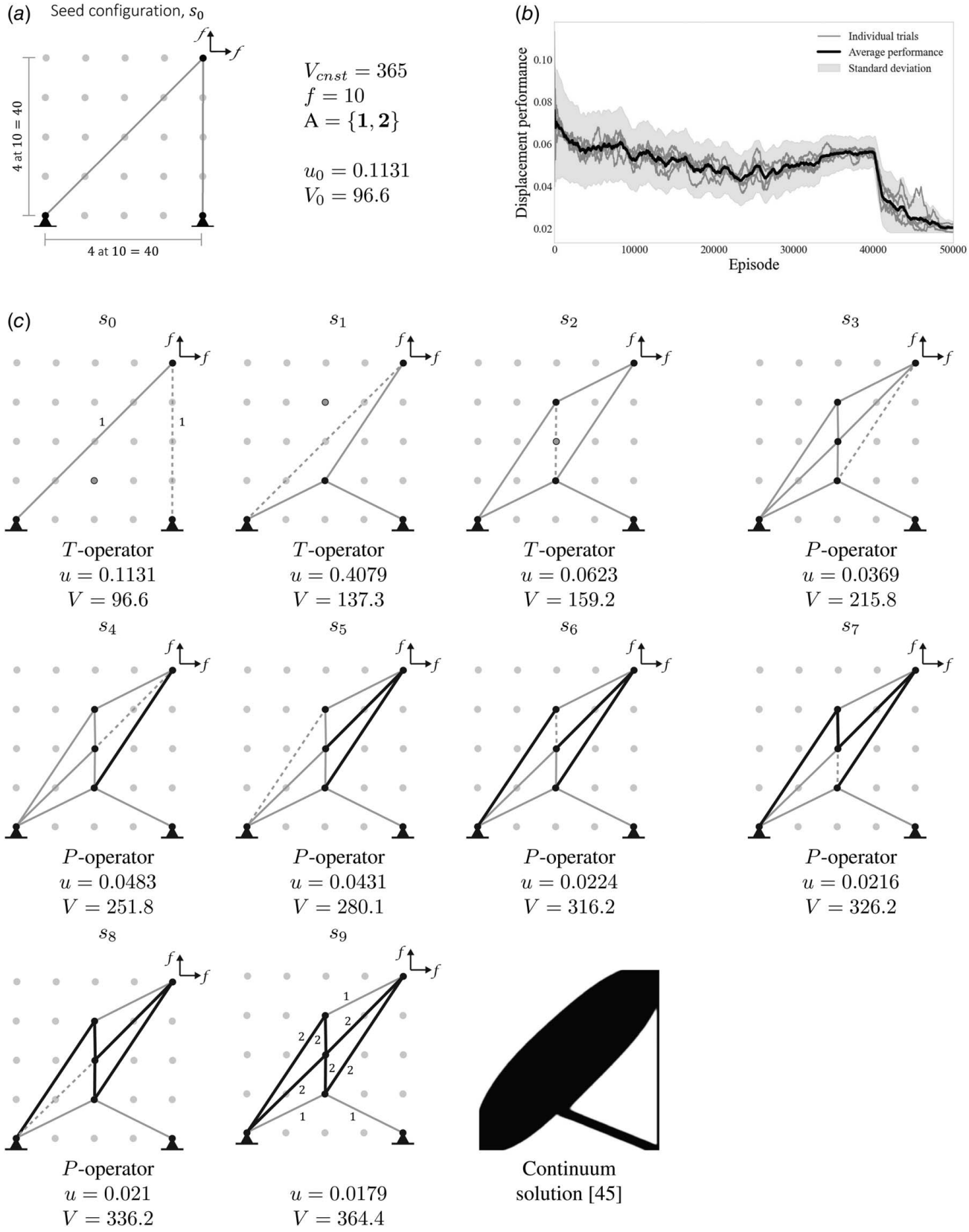

Fig. 6 (*a*) Example 4's 5 × 5 design domain Ω, seed configuration s_0, specified volume constraint of $V_{cnst} = 365$, and set of two discrete cross-sectional areas A = {1, 2}, (*b*) the MDP-HDRL framework's performance based on the agent following a greedy policy with respect to the HDRL network's current approximation of the state-action value function. The agent learns the policies that synthesize high performing design solutions as indicated by the decrease of the displacement performance, and (*c*) the MDP-HDRL framework agent's learned optimal policy. Assigned discrete cross-sectional areas are depicted by the elements color, specifically light gray represents the minimum area in the set $A = 1$ and solid black represents the largest area $A = 2$. The synthesized optimal design under state s_9 generally matches the continuum design solution's volume allocation, resembling the same load path, determined using TopOpt [45].

allocating volume, or sizing the elements, in general, agrees with the historical and conventional approach to structural design learned by human designers over decades of practice. Furthermore, the MDP-HDRL framework adeptly learned optimal policies even when the policy required the agent to take initial actions that resulted in poor performing design configurations, yet were required in order to synthesize the best performing configuration in the long run. For all examples, it was observed that the MDP-HDRL framework requires significantly few finite element model evaluations in order to learn a policy that synthesizes a high performing design solution in relation to each problem's state space cardinality.

Beyond the contributions of this paper, the MDP framework is general such that other methodological enhancements and developments for other applications are possible, although outside of the scope of this work. For example, refinements to the actions and rewards could expand the applicability of the suggested MDP-HDRL framework and enable it to be readily benchmarked to established discrete optimization approaches. Also, the design domain's nodal grid dependency could be relaxed. Typically this nodal grid dependency can be relaxed by refining the mesh, decreasing the node spacing, and increasing the element connectivity; however, this generally is achieved at the expense of increasing the state and action space as well as the computational cost to obtain a high performing design solution. The ability to connect to a node placed at any point in the design domain, not only at fixed points, could alleviate the mesh dependency without adding any, or significant, computational cost. Additionally, there are many suitable deep reinforcement learning techniques that can readily accommodate continuous actions that could be explored and incorporated into the framework. These topics are a part of future work.

Overall, the potential enhancements that could be made to the MDP-HDRL framework and other unknown, yet complementary research applications can lead to fully automated as well as interactive approaches, that assist designers in directly generating discrete design solutions that offer new structural forms, particularly when designing structures constructed from standardized sections/elements as with steel and timber. These new structural designs may also have the potential to reduce the consumption of natural resources and hence yield positive environmental impacts.

Acknowledgment

Gordon P. Warn acknowledges the support of the U.S. National Science Foundation under CMMI Grant No. 2322853. All opinions, findings, and conclusions expressed in this paper are those of the authors and do not necessarily reflect the views of the sponsor.

Conflict of Interest

There are no conflicts of interest.

Data Availability Statement

The datasets generated and supporting the findings of this article are obtainable from the corresponding author upon reasonable request.

References

[1] Stiny, G., and Gips, J., 1980, "Production Systems and Grammars: A Uniform Characterization," Environ. Plann. B: Plann. Des., **7**(4), pp. 399–408.
[2] Shea, K., and Cagan, J., 1999, "Languages and Semantics of Grammatical Discrete Structures," AI EDAM, **13**(4), pp. 241–251.
[3] Antonsson, E. K., and Cagan, J., 2005, *Formal Engineering Design Synthesis*, Cambridge University Press, Cambridge.
[4] Chakrabarti, A., 2013, *Engineering Design Synthesis: Understanding, Approaches and Tools*, Springer Science & Business Media, London.
[5] Campbell, M. I., and Shea, K., 2014, "Computational Design Synthesis," AI EDAM, **28**(3), pp. 207–208.
[6] Hooshmand, A., and Campbell, M. I., 2016, "Truss Layout Design and Optimization Using a Generative Synthesis Approach," Comput. Struct., **163**, pp. 1–28.
[7] Königseder, C., and Shea, K., 2016, "Visualizing Relations Between Grammar Rules, Objectives, and Search Space Exploration in Grammar-Based Computational Design Synthesis," ASME J. Mech. Des., **138**(10), p. 101101.
[8] Mata, M. P., Ahmed-Kristensen, S., and Shea, K., 2019, "Implementation of Design Rules for Perception Into a Tool for Three-Dimensional Shape Generation Using a Shape Grammar and a Parametric Model," ASME J. Mech. Des., **141**(1), p. 011101.
[9] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., and Ostrovski, G., 2015, "Human-Level Control Through Deep Reinforcement Learning," Nature, **518**(7540), p. 529.
[10] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., and Lanctot, M., 2016, "Mastering the Game of Go With Deep Neural Networks and Tree Search," Nature, **529**(7587), p. 484.
[11] Sutton, R. S., and Barto, A. G., 2018, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge.
[12] Burnap, A., Liu, Y., Pan, Y., Lee, H., Gonzalez, R., and Papalambros, P. Y., 2016, "Estimating and Exploring the Product Form Design Space Using Deep Generative Models," International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, **50107**.
[13] Dering, M. L., and Tucker, C. S., 2017, "Generative Adversarial Networks for Increasing the Veracity of Big Data," IEEE International Conference on Big Data (Big Data), pp. 2595–2602.
[14] Dering, M. L., and Tucker, C. S., 2017, "Implications of Generative Models in Government," AAAI Fall Symposium Series.
[15] Vermeer, K., Kuppens, R., and Herder, J., 2018, "Kinematic Synthesis Using Reinforcement Learning," International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, **51753**.
[16] Shu, D., Cunningham, J., Stump, G., Miller, S. W., Yukish, M. A., Simpson, T. W., and Tucker, C. S., 2020, "3D Design Using Generative Adversarial Networks and Physics-Based Validation," ASME J. Mech. Des., **142**(7), p. 071701.
[17] Yu, Y., Hur, T., Jung, J., and Jang, I. G., 2019, "Deep Learning for Determining a Near-Optimal Topological Design Without Any Iteration," Struct. Multidiscipl. Optim., **59**(3), pp. 787–799.
[18] Jang, S., Yoo, S., and Kang, N., 2022, "Generative Design by Reinforcement Learning: Enhancing the Diversity of Topology Optimization Designs," Computer-Aided Design, **146**, p. 103225.
[19] Sun, H., and Ma, L., 2020, "Generative Design by Using Exploration Approaches of Reinforcement Learning in Density-Based Structural Topology Optimization," Designs, **4**(2), p. 10.
[20] Raina, A., McComb, C., and Cagan, J., 2019, "Learning to Design From Humans: Imitating Human Designers Through Deep Learning," ASME J. Mech. Des., **141**(11), p. 111102.
[21] Puentes, L., Raina, A., Cagan, J., and McComb, C., 2020, "Modeling a Strategic Human Engineering Design Process: Human-Inspired Heuristic Guidance Through Learned Visual Design Agents," Proceedings of the Design Society: DESIGN Conference, **1**, pp. 355–364.
[22] Raina, A., Puentes, L., Cagan, J., and McComb, C., 2021, "Goal-Directed Design Agents: Integrating Visual Imitation With One-Step Lookahead Optimization for Generative Design," ASME J. Mech. Des., **143**(12), p. 124501.
[23] Hayashi, K., and Ohsaki, M., 2020, "Reinforcement Learning and Graph Embedding for Binary Truss Topology Optimization Under Stress and Displacement Constraints," Front. Built Environ., **6**, p. 59.
[24] Zhu, S., Ohsaki, M., Hayashi, K., and Guo, X., 2021, "Machine-Specified Ground Structures for Topology Optimization of Binary Trusses Using Graph Embedding Policy Network," Adv. Eng. Softw., **159**, p. 103032.
[25] Hayashi, K., and Ohsaki, M., 2022, "Graph-Based Reinforcement Learning for Discrete Cross-Section Optimization of Planar Steel Frames," Adv. Eng. Inform., **51**, p. 101512.
[26] Kupwiwat, C.-T., Hayashi, K., and Ohsaki, M., 2024, "Multi-objective Optimization of Truss Structure Using Multi-agent Reinforcement Learning and Graph Representation," Eng. Appl. Artificial Intell., **129**, p. 107594.
[27] Ororbia, M. E., and Warn, G. P., 2020, "Structural Design Synthesis Through a Sequential Decision Process," International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, **83983**, p. V009T09A045.
[28] Ororbia, M. E., and Warn, G. P., 2021, "Design Synthesis Through a Markov Decision Process and Reinforcement Learning Framework," ASME J. Comput. Inf. Sci. Eng., **22**(2), p. 021002.
[29] Ororbia, M. E., and Warn, G. P., 2023, "Design Synthesis of Structural Systems as a Markov Decision Process Solved With Deep Reinforcement Learning," ASME J. Mech. Des., **145**(6), p. 061701.
[30] Oberhauser, M., Sartorius, S., Gmeiner, T., and Shea, K., 2015, "Computational Design Synthesis of Aircraft Configurations With Shape Grammars," *Design Computing and Cognition'14*, Springer, Switzerland, pp. 21–39.
[31] Spallino, R., and Rizzo, S., 2002, "Multi-objective Discrete Optimization of Laminated Structures," Mech. Res. Commun., **29**(1), pp. 17–25.
[32] Sjølund, J., Peeters, D., and Lund, E., 2019, "Discrete Material and Thickness Optimization of Sandwich Structures," Composite Struct., **217**, pp. 75–88.
[33] Marjanovic, N., Isailovic, B., Marjanovic, V., Milojevic, Z., Blagojevic, M., and Bojic, M., 2012, "A Practical Approach to the Optimization of Gear Trains With Spur Gears," Mech. Mach. Theory, **53**, pp. 1–16.

[34] Königseder, C., and Shea, K., 2016, "Comparing Strategies for Topologic and Parametric Rule Application in Automated Computational Design Synthesis," ASME J. Mech. Des., **138**(1), p. 011102.
[35] Shea, K., Cagan, J., and Fenves, S. J., 1997, "A Shape Annealing Approach to Optimal Truss Design With Dynamic Grouping of Members," ASME J. Mech. Des., **119**(3), pp. 388–394.
[36] Dayan, P., and Hinton, G. E., 1992, "Feudal Reinforcement Learning," Advances in Neural Information Processing Systems, Vol. 5.
[37] Dietterich, T. G., 1998, "The Maxq Method for Hierarchical Reinforcement Learning," ICML, Vol. 98, pp. 118–126.
[38] Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N., 2016, "Dueling Network Architectures for Deep Reinforcement Learning," International Conference on Machine Learning, PMLR, pp. 1995–2003.
[39] Wei, F., Feng, G., Sun, Y., Wang, Y., Qin, S., and Liang, Y.-C., 2020, "Network Slice Reconfiguration by Exploiting Deep Reinforcement Learning With Large Action Space," IEEE Trans. Netw. Serv. Manage., **17**(4), pp. 2197–2211.
[40] Tavakoli, A., Pardo, F., and Kormushev, P., 2018, "Action Branching Architectures for Deep Reinforcement Learning," Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32.
[41] Schaul, T., Quan, J., Antonoglou, I., and Silver, D., 2015, "Prioritized Experience Replay," preprint arXiv:1511.05952.
[42] Yang, T., Tang, H., Bai, C., Liu, J., Hao, J., Meng, Z., Liu, P., and Wang, Z., 2021, "Exploration in Deep Reinforcement Learning: A Comprehensive Survey," preprint arXiv:2109.06668.
[43] Tieleman, T., and Hinton, G., 2012, "Lecture 6.5-rmsprop: Divide the Gradient by a Running Average of Its Recent Magnitude," COURSERA: Neural Networks for Machine Learning, **4**(2), pp. 26–31.
[44] Achtziger, W., and Stolpe, M., 2007, "Truss Topology Optimization With Discrete Design Variables-Guaranteed Global Optimality and Benchmark Examples," Struct. Multidiscipl. Optim., **34**, pp. 1–20.
[45] Sigmund, O., 2001, "A 99 Line Topology Optimization Code Written in Matlab," Struct. Multidiscipl. Optim., **21**(2), pp. 120–127.