

Design Synthesis Through a Markov Decision Process and Reinforcement Learning Framework

Maximilian E. Ororbia¹

Department of Civil and Environmental Engineering,
The Pennsylvania State University,
State College, PA 16802
e-mail: meo9@psu.edu

Gordon P. Warn

Department of Civil and Environmental Engineering,
The Pennsylvania State University,
State College, PA 16802
e-mail: gpw1@psu.edu

This article presents a framework that mathematically models optimal design synthesis as a Markov Decision Process (MDP) that is solved with reinforcement learning. In this context, the states correspond to specific design configurations, the actions correspond to the available alterations modeled after generative design grammars, and the immediate rewards are constructed to be related to the improvement in the altered configuration's performance with respect to the design objective. Since in the context of optimal design synthesis the immediate rewards are in general not known at the onset of the process, reinforcement learning is employed to efficiently solve the MDP. The goal of the reinforcement learning agent is to maximize the cumulative rewards and hence synthesize the best performing or optimal design. The framework is demonstrated for the optimization of planar trusses with binary cross-sectional areas, and its utility is investigated with four numerical examples, each with a unique combination of domain, constraint, and external force(s) considering both linear-elastic and elastic-plastic material behaviors. The design solutions obtained with the framework are also compared with other methods in order to demonstrate its efficiency and accuracy. [DOI: 10.1115/1.4051598]

Keywords: computational synthesis, truss optimization, design synthesis, machine learning for engineering applications, reinforcement learning

1 Introduction

Computational design synthesis (CDS) is an area that explores the generative capabilities of computational systems composed of various techniques each with the goal of automating the synthesis of candidate design solutions [1–4]. In general, CDS involves the generation, evaluation, and selection of design alternatives for a specified design problem [5]. A commonality among the CDS approaches is the use of generative design grammars to modify candidate solutions through topological, spatial, and/or parametric rules. To some extent, CDS mimics human designers' quest for improvement, and as such, a number of CDS approaches have combined grammar rules with different optimization search heuristics, for example, branch-and-bound methods [6], simulated annealing [7–9], genetic algorithms [10,11], and a machine learning-based search algorithm [12], to synthesize efficient and/or optimized design solutions for a range of engineering design problems.

Design tools employing machine learning are being used for the purpose of, for example, generating and evaluating design alternatives [13–15], design space exploration [16], and structural optimization [17–19]. With respect to structural optimization considering discrete design variables, Hayashi and Ohsaki [20] use a machine learning-based method to determine near-optimal topological designs using the ground structure approach [21]. Specifically, Hayashi and Ohsaki [20] employ a neural network combined with a graph embedding model for the purpose of binary optimization of trusses to minimize structural volume under stress and displacement constraints, thus demonstrating that a machine learning method could find suboptimal solutions at low computational cost.

In this article, a framework that aligns with CDS is presented for optimal design synthesis. However, unique to this study is mathematically modeling optimal design synthesis as a Markov Decision Process (MDP) that is solved with a reinforcement learning (RL) algorithm, namely, Q-learning [22,23]. Similar to other CDS approaches, this framework adopts grammar rules, however, the grammar rules are modeled through the actions of the MDP to generate new design configurations. RL is then employed to solve the MDP, learning to take optimal actions in a given state. A beneficial aspect of modeling design synthesis as an MDP solved with Q-learning is that it naturally takes advantage of key mathematical properties to achieve strong convergence to global optimal design solutions. To demonstrate this, the framework is applied to the optimization of planar trusses, a common problem used to assess various engineering design methods. The numerical examples considered collectively include the following characteristics: binary areas, multiple external forces, and linear-elastic and elastic-plastic material behaviors.

A finite MDP is a discrete time stochastic control process that is represented by a 4 tuple $(S, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where S is a finite set of all possible decision states, \mathcal{A} is a finite set of all valid actions, \mathcal{P} is a set of state transition probabilities, and \mathcal{R} is a set of all possible rewards. In the context of optimal design synthesis, the *states* correspond to specific design configurations, the *actions* correspond to the available alterations modeled after grammar rules, and the immediate *rewards* are constructed to be related to the improvement in the altered configuration's performance. Since the actions are discrete, the MDP is naturally suitable for design problems with binary or discrete variables. In the context of optimal design synthesis, the parameters of the MDP will in general not be known at the onset of the process; hence RL, a subclass of machine learning, can be employed to efficiently solve the MDP. In the RL setting, an agent learns to take optimal actions through experienced transitions rather than with a priori knowledge of transitions. Therefore, the goal of the agent is to maximize the cumulative reward by learning how to optimally

¹Corresponding author.

Contributed by the Computers and Information Division of ASME for publication in the JOURNAL OF COMPUTING AND INFORMATION SCIENCE IN ENGINEERING. Manuscript received February 3, 2021; final manuscript received June 17, 2021; published online July 21, 2021. Assoc. Editor: Matthew I. Campbell.

alter a given configuration so as to maximize the performance of the configuration and hence synthesize an optimal design.

The remainder of this article is organized as follows: Sec. 2 provides a brief background as well as the associated mathematics for finite MDPs and the RL method employed. Following this, in Sec. 3, the presented framework, herein referred to as MDP-RL synthesis, is described. In Sec. 4, the framework is applied to the optimization of planar trusses with binary areas with the objective of minimizing displacement for a given external force(s), subject to volume and stability constraints. Four numerical examples are presented in Sec. 5, each with a unique combination of domain, volume constraint, and external force(s) considering either linear-elastic or elastic-plastic material behaviors. As a point of comparison, the efficiency and accuracy of the MDP-RL synthesis framework are compared with other methods in Sec. 6.

2 Background

This section presents a brief overview of finite MDPs and the tabular implementation of RL that is employed in this paper.

2.1 Markov Decision Process. As previously stated, a finite MDP is a discrete time stochastic control process represented by a 4 tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$. At any time t , an agent finds the system in state $s_t \in \mathcal{S}$ and executes a valid action $a_t \in \mathcal{A}(s_t)$ that causes the environment to transition to a new state s_{t+1} with a probability $p(s_{t+1}|s_t, a_t)$. When the agent transitions to a new state, it receives a reward $R_{t+1} = r(s_t, a_t, s_{t+1})$. In a given state, the agent chooses an action according to the policy function, $\pi_t(a|s): \mathcal{S} \rightarrow \mathcal{A}$ that maps states to actions, which can be either deterministic or stochastic. Starting from an initial state s_0 , the goal of the agent is to maximize the expected rewards over the long run. Specifically, the agent takes actions that maximize the discounted return:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (1)$$

where γ is the discount rate. The discount rate determines the importance of future rewards when the agent is in state s_t .

The expected return when the agent takes an action a in state s following the policy π thereafter is referred to as the action-value function, denoted as $q_\pi(s, a)$, and is defined as follows:

$$\begin{aligned} q_\pi(s, a) &\doteq \mathbb{E}_\pi[G_t | s_t = s, a_t = a] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | s_t = s, a_t = a \right] \end{aligned} \quad (2)$$

where $\mathbb{E}_\pi[\cdot]$ is the expectation operator. Equation (2) is also referred to as the *Bellman* equation [24]. The corresponding optimal action-value function is defined by Eq. (3).

$$\begin{aligned} q_*(s, a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a_{t+1}} (s_{t+1}, a_{t+1} | s_t = s, a_t = a)] \\ &= \sum_{s_{t+1}} p(s_{t+1} | s_t, a_t) [r(s_t, a_t, s_{t+1}) + \gamma \max_{a_{t+1}} q_*(s_{t+1}, a_{t+1})] \end{aligned} \quad (3)$$

By first solving the optimal action-value function, Eq. (3), using, for example, a suitable dynamic programming (DP) algorithm [23], an optimal policy (or policies) can be obtained by deterministically choosing the action with the maximum action-value in each state $s \in \mathcal{S}$ such as:

$$\pi_*(a|s) = \arg \max_{a \in \mathcal{A}(s)} q_*(s, a) \quad (4)$$

where π_* denotes the optimal policy. However, if the rewards \mathcal{R} and/or transition probabilities \mathcal{P} are not known at the onset of the process, RL methods can be used to learn an optimal policy.

2.2 Reinforcement Learning. RL is a subclass of machine learning and is described as the behavioral learning of an agent that interacts with a dynamic environment [25]. RL methods maximize the cumulative numerical reward signal by learning to map different states in a sequential decision process to optimal actions. Central to RL is the idea of temporal difference (TD) learning [23]. TD learning is a class of model-free methods that update current estimates of the value, or action-value, function rather than waiting until a terminal state is reached. Similar to Monte Carlo and DP methods, TD methods learn directly from experience in the form of numerical rewards sampled from an environment. However, TD methods perform updates to the current state-action pair based on learned estimates by bootstrapping [23].

Q-learning [22] is an off-policy TD control algorithm where the action-value function is updated according to:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t \cdot [R_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (5)$$

where $Q(s_t, a_t)$ is the learned action-value function that directly approximates Eq. (2) regardless of the policy being followed. The bracketed quantity, representing the temporal difference, $[R_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} Q(s_{t+1}, a_{t+1})]$, is the bootstrapped sample estimate of the quantity in the expectation operator of Eq. (2). The learning rate, denoted as α_t , controls the weight of the update.

For an episodic learning task, the action-value function is initialized, Q_0 , arbitrarily. Thereafter, the agent selects a greedy action and transitions from the initial state s_0 to a next state, that could be either a continuing state, s_{t+1} , or a terminal state, s_T . If entering a terminal state, the value of $Q(s_T, a_T)$ is set equal to zero. A set of transitions from s_0 to s_T is referred to as an episode.

Q-learning has been shown to possess strong convergence properties, converging to an optimal action-value function in a stationary environment with discrete domains [23]. Specifically, an agent is guaranteed to converge to the optimal policy in the limit if two conditions are met: (1) the learning rate must gradually approach zero; and (2) all state-action pairs are visited for an infinite number of episodes [22]. In practice, to satisfy the second condition and to prevent the possibility of converging to local optima by always choosing the action with the maximum immediate reward, an ϵ -greedy policy can be adopted. By following an ϵ -greedy policy, an agent selects the action with the greatest estimated state action-value with probability $(1-\epsilon)$ and chooses a random action with probability ϵ . Choosing a random action encourages the agent to explore and find, potentially, more optimal actions and to refine its estimate of the action-value function.

3 Optimal Design Synthesis Through a Finite Markov Decision Process

This section describes the formulation of optimal design synthesis as a finite MDP solved with RL. As previously described, an MDP is a 4 tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where in the context of design synthesis, at any time t , a given design configuration represents the state s_t , and the available actions, $a \in \mathcal{A}(s_t)$, are the available grammar rules that can be applied to the given configuration that, when executed, results in a new configuration and hence transitions to the next state, s_{t+1} . Optimal design synthesis is achieved by formulating the rewards to be related to the improvement in the performance of a given design configuration. For now, the specifics of the rewards, \mathcal{R} , and transition probabilities, \mathcal{P} , are deferred to the subsequent section since these are problem specific. However, since both states and actions are relatively general, they are further discussed in this section.

The optimal design synthesis process starts with an initial state, s_0 , which is an initial configuration, or guess, that is then altered by a sequence of discrete actions chosen by an agent. Here, the initial configuration is composed of a sparsely connected set of nodes selected from a specified nodal domain, which will be

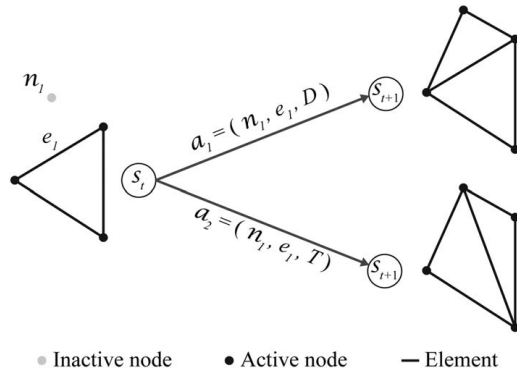


Fig. 1 Example of a configuration altered by two actions, a_1 and a_2 , and the associated new states. Operators D and T on element 1, e_1 , and node 1, n_1 , generate two new configurations.

referred to herein as the seed. For this study, the MDP actions are modeled after topological grammar rules. The presented framework can be extended to consider spatial and parametric grammar rules; however, this is outside the scope of this article. Adapted from Lipson [26], the set of actions that alter the seed configuration can be represented by a decision tree, whereby in a given state an agent is provided unique actions based on two operators, denoted as T or D . Figure 1 provides an illustration of a configuration in state s_i altered by two actions and the resulting new states, or new topological configurations, as a result of the T and D operators. A T operator removes a selected element and connects a newly activated node, selected from the specified domain, with other active nodes. A D operator activates a new node and connects it with the existing configuration without removing an element.

In the context of this study, an action a consists of three components: (1) the selection of an available inactive node, (2) the selection of an eligible element for that node, and (3) the choice of a legal operator based on the location of the node relative to the selected element. As such, an action set is defined as follows:

$$A(s_i) = \{(n_i, e_j, o_k) | n_i \in N_{avail}(s_i), e_j \in E_{allow}(s_i, n_i), o_k \in O_{legal}(n_i, e_j)\} \quad (6)$$

where, depending on the given configuration at s_i , an agent selects a node n_i from a set of available nodes, N_{avail} , an element e_j from a set of eligible elements, E_{allow} , and an operator o_k from a set of legal operators, O_{legal} . The location of an available inactive node in the domain dictates which elements and operators comprise the set of actions. For example, a T operator can be applied to an element for any selected available node. However, if an existing element passes through an available node, then a D operator cannot be applied to that element. Furthermore, if the selected node is “inside” the configuration, then only a single D operator can be applied to any of the immediate surrounding elements. Finally, elements are connected between the selected node and other active nodes so long as they do not intersect any other existing elements. These rules prevent redundant configurations from forming as a result of different actions on a given configuration.

The process of altering a given configuration is repeated until a terminal state s_T is reached. In general, a terminal state is reached if all available nodes in the domain are active. Other terminal states will exist depending on the design application, objective, and/or constraints, as will be made evident in the subsequent section. The described initial states, transition states, and actions are generally applicable and thus are employed for the optimization of planar trusses. However, specifics with respect to the rewards, transition probabilities, and additional terminal states will vary depending on the specified design objective and constraints. In the following sections, an illustrative design problem is presented,

including specific details regarding the terminal states, transition probabilities, and rewards for the given design objective along with numerical examples for different nodal domains and constraint values.

4 Application to Optimal Truss Design Synthesis

In this section, the MDP-RL synthesis framework is applied for the optimization of planar trusses considering discrete design variables, specifically binary areas. Traditional topology optimization methods that assume the design variables to be continuous are typically assisted by gradients of the objective function and constraint(s) [27,28], and if a binary outcome is desired, then a penalization method can be employed [29,30]. Alternatively, topology optimization methods that directly deal with discrete or mixed-integer variables include branch-and-bound [31–33] and nongradient-based methods [34–37]. While each method has its own merit, they can be limited in applicability due to their reliance on mathematical relaxations and can be computationally expensive with respect to converging to global optima [30,38,39]. The developed MDP-RL synthesis framework models structural optimization as an MDP, which naturally accommodates discrete actions, making it suitable for discrete structural optimization for a variety of problem characteristics. Specifically, as described in Sec. 5, the framework is used to solve a variety of truss optimization examples each with different specified nodal domains and boundary conditions, seed configurations, single and multiple external forces, and consideration of different material properties.

4.1 Design Problem. The design goal is to generate a stable truss design that minimizes displacement under an externally applied force(s), f , such that the total structural volume does not exceed a specified volume constraint. This design formulation is comparable to compliance minimization problems for the optimization of truss structures [40–42] and is formulated as follows:

$$\begin{aligned} &\text{minimize} && u(\mathbf{A}) \\ &\text{subject to} && \sum_{i=1}^{n_m} A_i L_i \leq V_{cns} \\ &&& A_i \in \{0, 1\} \quad (i = 1, \dots, n_m) \end{aligned} \quad (7)$$

where u is the truss configuration’s displacement at the specified node, \mathbf{A} denotes the vector of cross-sectional areas that are assigned to the n_m number of truss elements, and V_{cns} is the volume constraint. The optimization problem is reformulated into the MDP-RL synthesis framework, in Sec. 4.4, where the discrete actions of the MDP translate into discrete design variables and the maximum reward for the agent is determined by maximizing the difference between the displacement of the initial configuration and the displacement that can be achieved by an altered configuration. The displacement of each truss configuration is evaluated using either a linear or nonlinear finite element analysis (FEA) depending on the assumed material behavior as will be subsequently described. Before the FEA, the stability of each truss configuration is assessed by checking the condition of its partitioned global stiffness matrix. The outcome of the FEA is the displacement at a specified node that is used to determine the immediate rewards.

4.2 Linear Finite Element Analysis. A linear FEA is implemented when the material behavior of a truss structure is assumed to be linear-elastic, i.e., truss elements remain elastic after the yield limit σ_y is reached, as illustrated in Fig. 2(a). For the linear FEA, element local stiffness matrices, k_{ele} , are transformed, according to a given truss configuration, using conventional coordinate transformation matrices [43], and then mapped to the global system equilibrium equation: $F_g - K_g u_g = 0$. After assembling the global equilibrium equation, the appropriate boundary conditions

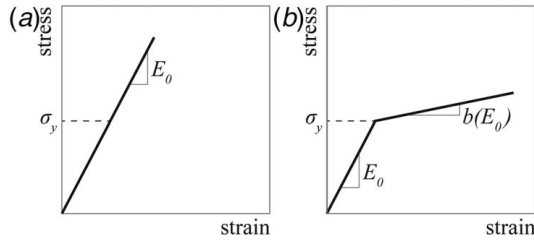


Fig. 2 Illustrations of (a) linear-elastic and (b) elastic-plastic material behavior models where E_0 is the elastic modulus, σ_y is a yield stress limit, and b is a strain-hardening ratio.

are applied at the system level to obtain the system equilibrium equation: $F_s - K_s u_s = 0$, where F_s is the externally applied force vector, K_s is the partitioned global stiffness matrix of the system, and u_s is the vector of free nodal displacements.

4.3 Nonlinear Finite Element Analysis. A nonlinear FEA is implemented when the material behavior of a truss structure is assumed to be elastic-plastic, that is, an element is assumed to be linear-elastic up to its yield limit and exhibits plastic deformation after exceeding that limit, as illustrated in Fig. 2(b). For the nonlinear FEA implemented in this article, each truss configuration is modeled and analyzed in OPENSEES [44], an open source structural analysis software developed for earthquake engineering applications. All truss elements are assigned *Steel01* material and a strain-hardening ratio b (ratio between elastic and post-elastic stiffness). An external force is applied in 100 increments until a maximum specified value is reached. For each increment, the static equilibrium equations are iteratively solved using the Newton method until a specified error tolerance is met. Unlike the linear FEA, the local stiffness matrices for a nonlinear FEA do not remain constant throughout the analysis and are therefore constantly updated for each Newton iteration if an element exceeds its yield limit.

4.4 Learning Environment. The optimal truss design synthesis problem is formulated as an undiscounted ($\gamma = 1$), episodic, finite MDP, and solved using a tabular Q-learning (RL) algorithm. The tabular Q-learning algorithm implemented in this article is outlined in Algorithm 1. For this application, the terminal states include the general state of no inactive nodes as well as if the altered truss structure is unstable or if the volume constraint is exceeded. The probability with which a given state will transition to a continuing or terminal state is either 0 or 1, depending on the current state and action.

Algorithm 1 Tabular Q-learning for optimal truss topology synthesis

Data: Seed configuration s_0 , discount factor γ , exploration probability ϵ , learning rate α , number of episodes N

```

1 for episodes = 1:N do
2   Initialize with seed configuration,  $s_0$ 
3   repeat
4     Choose  $a_t$  from  $Q(s_t, a)$  using  $\epsilon$ -greedy policy
5     Execute action  $a_t$ , transition to state  $s_{t+1}$ , if  $s_{t+1}$  is a continuing
      state perform FEA and observe reward  $R_{t+1}$ 
6     Perform Q-learning update using Eq. (5)
7   until  $s_t = s_T$ ;
```

The reward signal for the design synthesis task is constructed such that the immediate reward is related to the change in performance of the design in s_t with the altered configuration's performance in s_{t+1} . In other words, if the displacement of the altered

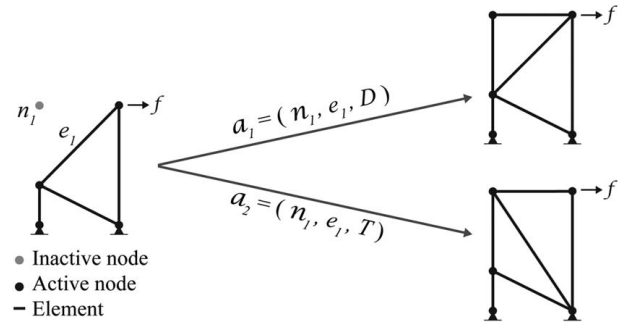


Fig. 3 Example of a truss configuration altered by two actions, a_1 and a_2 , and the associated new states. Specifically D and T operations on element 1, e_1 , and node 1, n_1 , generate two new truss configurations.

configuration in s_{t+1} is less than the configuration's displacement in s_t , then the agent receives a positive reward equal to the difference, whereas a larger displacement in s_{t+1} results in a negative reward. With the goal of maximizing the cumulative reward over the long run, the agent learns an optimal sequence of alterations starting from the seed configuration that improves the performance of the design with respect to the objective. Thus, if the agent learns a sequence of alterations that produces the greatest reward, it has identified the optimal design.

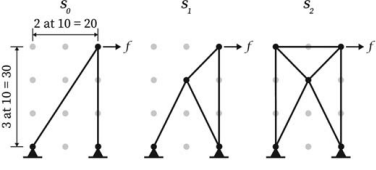
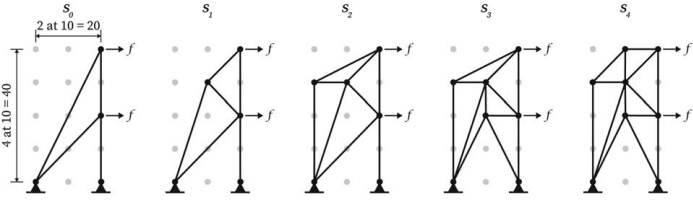
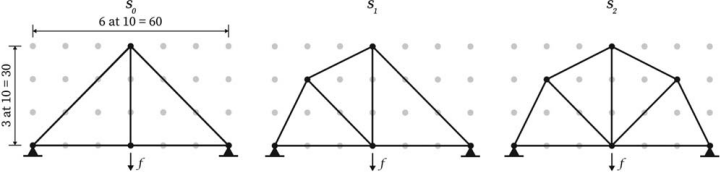
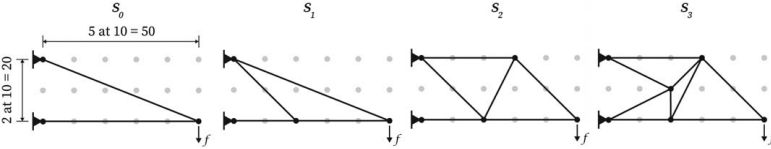
The actions described in Sec. 3 are employed for optimal truss design synthesis. For example, as introduced in the study by Ororbia and Warn [45], Fig. 3 illustrates how two actions, a_1 and a_2 , alter a given truss configuration in state s_t into new truss designs. Specifically, action $a_1 = (n_1, e_1, D)$ alters the truss configuration in state s_t by activating node n_1 and generating two elements that connect to the truss in state s_t without removing e_1 . Alternatively, action $a_2 = (n_1, e_1, T)$ alters the truss in state s_t by activating node n_1 and generating new elements after removing element e_1 . Although two actions were described for this illustrative example, the number of available actions depend on the given state.

An ϵ -greedy policy is employed to choose actions, in which the exploration rate, ϵ_t , is decayed over a specified number of episodes. The exploration rate is decayed according to a cosinusoidal function of $\epsilon_t = A_\epsilon \cos(B_\epsilon e)$, where A_ϵ and B_ϵ are parameters used to specify the amplitude and period of the cosine function, and e is the current episode. For the numerical examples presented, the learning rate, α_t , is also decayed using the same cosinusoidal function but with unique parameter values. For a set of episodes, constituting an experiment, both the learning and exploration rates (α_t and ϵ_t) are initialized at $e = 1$ to $\alpha_e = 1$ and $A_\epsilon = 0.9$ and decayed to 0.1 and 0.01, respectively, for all numerical examples presented in the following section. Both parameters are held constant at their respective decayed values for the remaining episodes.

5 Numerical Examples

Four numerical examples are presented in this section to demonstrate and investigate the utility of the suggested MDP-RL synthesis framework applied to the optimization of planar trusses with binary cross-sectional areas and are summarized in Table 1. The performance of the implemented tabular Q-learning algorithm is assessed based on the average of 50 experiments, where each experiment consists of a specified set of episodes. Each example begins with a truss seed configuration, an externally applied force(s) f , and boundary conditions. Structural parameter values are assigned so that the results of the linear and nonlinear FEA are nondimensional. Specifically, for the linear FEA performed in examples 1–3, an elastic modulus of $E_0 = 10^3$ and an area of $A = 1$ are assumed. In example 4, all truss elements are assigned *Steel01* material for the nonlinear FEA with a yield strength $\sigma_y = 25$, an initial modulus of elasticity $E_0 = 2 \times 10^6$, an area of $A = 1$, and a strain-hardening

Table 1 Summary of numerical design examples

Attributes and parameters		Sequence of feasible truss configurations in learned optimal policy	Optimal truss performance
(a) Example 1	Material: linear-elastic $f = 10$ $V_{cnst} = 160$ $u_0 = 0.1847$ $V_0 = 66.1$ $B_\alpha = 9.7 \times 10^{-4}$ $B_e = 1.6 \times 10^{-3}$		Optimal truss in s_2 $u_{opt} = 0.0895$ $V_{opt} = 153.0$
(b) Example 2	Material: linear-elastic $f = 10$ $V_{cnst} = 240$ $u_0 = 0.4236$ $V_0 = 113.0$ $B_\alpha = 2.65 \times 10^{-4}$ $B_e = 3.1 \times 10^{-4}$		Optimal truss in s_4 $u_{opt} = 0.1859$ $V_{opt} = 238.8$
(c) Example 3	Material: linear-elastic $f = 10$ $V_{cnst} = 250$ $u_0 = 0.0724$ $V_0 = 174.9$ $B_\alpha = 7.5 \times 10^{-5}$ $B_e = 8.21 \times 10^{-5}$		Optimal truss in s_2 $u_{opt} = 0.0425$ $V_{opt} = 236.0$
(d) Example 4	Material: elastic-plastic $f = 100$ $V_{cnst} = 200$ $u_0 = 0.6388$ $V_0 = 103.9$ $B_\alpha = 9.7 \times 10^{-4}$ $B_e = 1.6 \times 10^{-3}$		Optimal truss in s_3 $u_{opt} = 0.2291$ $V_{opt} = 199.5$

ratio $b = 0.05$. The length of each element for all examples is determined based on its connectivity.

5.1 Example 1: 4x3 Nodal Domain With a Single Concentrated External Force. The attributes and parameters for this example are presented in Table 1(a). Specifically, the nodal domain and seed configuration considered for this example are shown as state s_0 . Two pin supports are specified at the bottom left and right nodes of the design domain. A horizontal external force of $f = 10$ is applied to the upper right node in the domain and a volume constraint, V_{cnst} , of 160 is specified. The seed configuration has a volume, V_0 , of 66.1 and a displacement, u_0 , of 0.1847

at the upper right node in the domain under the specified external force as determined by a linear FEA.

The learning and exploration rates are initialized and decayed for each experiment as previously described. The learning rate and exploration rate are decayed by, respectively, setting $B_\alpha = 9.7 \times 10^{-4}$ and $B_e = 1.6 \times 10^{-3}$. As with all learning systems, the balance between exploration and exploitation is task dependent, as such, the value of these parameters was determined based on preliminary investigations. For example, Fig. 4(a) shows the effect that different B_e parameter values have on the number of model evaluations and Fig. 4(b) presents the average percentage that the learned optimal policy synthesizes the same optimal truss for each experiment

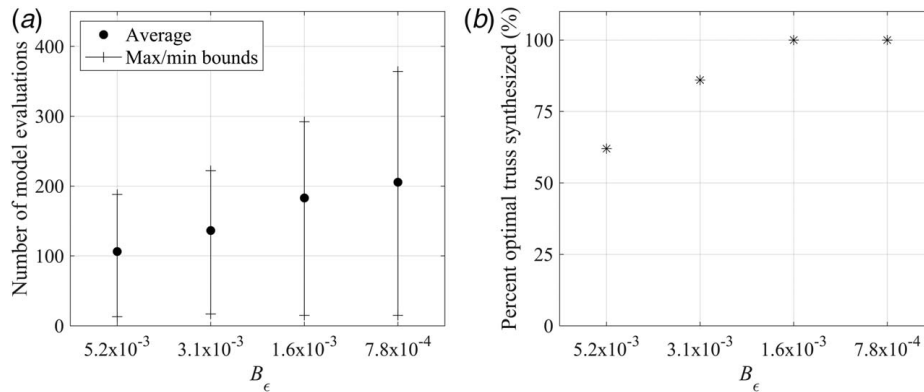


Fig. 4 For Example 1, (a) average number of model evaluations with different exploration rate decay parameters B_e and (b) average percent that the agent synthesizes the optimal truss in the learned optimal policy. The increase of model evaluations with smaller B_e parameters is evident and relates to the agent exploring longer.

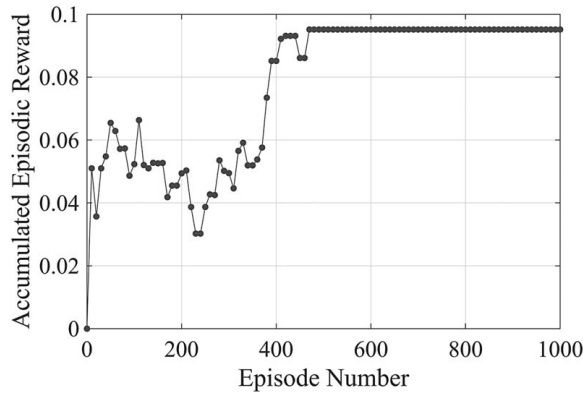


Fig. 5 Average performance of the agent at every ten episodes for Example 1. For each experiment, the agent learned the optimal policy producing the same optimal truss configuration.

conducted, which is separately verified through an exhaustive evaluation to be the global optimal solution. As shown in Fig. 4(b), there is a point where the agent learns to synthesize the optimal design in each episode such that forcing the agent to continue to explore does not improve its performance and needlessly increases the number of model evaluations. Similar results were observed for the other numerical examples but are omitted for brevity.

The sequence of altered configurations that the agent makes from the learned optimal policy is illustrated in Table 1(a). The truss configuration shown for s_2 is the optimal truss configuration determined by the agent, for all experiments. The optimal truss has a volume, V_{opt} , of 153.0 and under the specified external force a displacement, u_{opt} , of 0.0895 at the upper right node in the domain.

The learning process was paused after every ten episodes to evaluate the current policy. Specifically, the policy was evaluated by selecting greedy actions with respect to the current action-value function to obtain a cumulative reward statistic. The maximum reward that the agent can attain is 0.0952, but is unknown to the agent at the onset. As shown in Fig. 5, the agent converges to the maximum reward, where the average performance of the agent is shown for the first 1000 episodes since the accumulated reward remained constant for all subsequent episodes. Converging to the maximum reward indicates that the agent has learned an optimal policy that produces an optimal truss configuration with minimal displacement. On average, the agent evaluated 183 truss configurations per experiment to learn the optimal policy.

5.2 Example 2: 5x3 Nodal Domain With Two External Forces. To explore a state space with a greater variability in truss structure configurations, the MDP-RL synthesis framework is applied to a domain with a larger number of inactive nodes than in the previous example and with two externally applied forces. Table 1(b) presents the specified nodal domain and seed configuration in state s_0 and the attributes and parameters used for this example. For all experiments, the optimal policy produced the optimal truss configuration shown in the fourth state, s_4 . The optimal configuration that satisfies the volume constraint has a volume of 238.8 and a displacement of 0.1859 at the upper right edge node in the domain under the specified external forces.

The maximum reward that the agent can attain is 0.2377, again unknown at the onset. The agent's average accumulated episodic reward converges to the maximum value, which indicates that the agent learns an optimal policy. Figure 6 presents the average performance for the first 3000 episodes. On average, the agent evaluated 1334 truss configurations per experiment to learn the optimal policy.

5.3 Example 3: 4x7 Nodal Domain With a Single Concentrated External Force. To show its versatility, the MDP-RL synthesis framework was applied to a bridge design

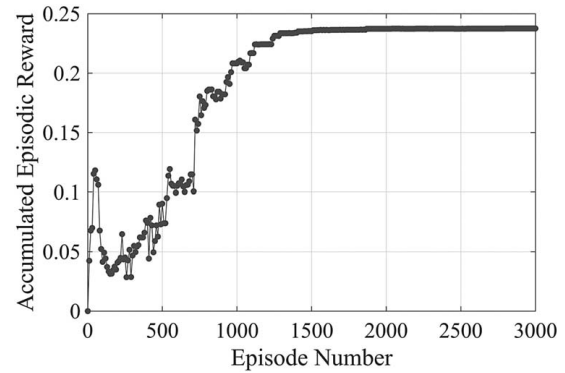


Fig. 6 Average performance of the agent at every ten episodes for Example 2. For each experiment, the agent learned the optimal policy producing the same optimal truss configuration.

domain with a single external force. See Table 1(c) for this example's attributes and parameters and for the sequence of altered configurations resulting from a learned optimal policy. The truss bridge configuration shown for s_2 is the optimal truss configuration and has a volume of 236.0 and under the specified external force a displacement of 0.0425 at the bottom middle node of the domain.

For this example, the maximum reward that the agent can attain is 0.0299, again unknown at the onset. The agent converges to the maximum reward value, as shown in Fig. 7, indicating that the agent has learned an optimal policy that produces an optimal truss configuration. On average, the agent evaluated 324 truss configurations per experiment to learn the optimal policy.

5.4 Example 4: 3x6 Nodal Domain With a Single External Force and Considering Material Nonlinearity. This example is presented to further demonstrate the generality of the MDP-RL synthesis framework by evaluating the performance of each truss configuration considering elastic-plastic material behavior. Table 1(d) illustrates the specified nodal domain and seed configuration in state s_0 and the attributes and parameters used for this example. A vertical force, with a value of $f = 100$, is applied to the bottom node of the right edge of the domain incrementally in 100 steps. The altered configurations from the learned optimal policy are sequentially presented in Table 1(d). The truss configuration shown for s_3 is the optimal truss configuration and has a volume of 199.5 and a displacement of 0.2291 at the bottom right edge node of the domain under the specified external force.

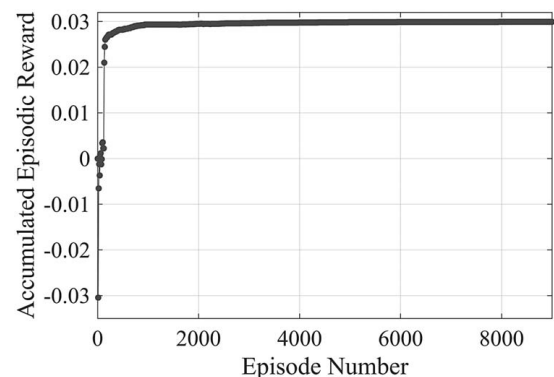


Fig. 7 Average performance of the agent at every ten episodes for Example 3. For each experiment, the agent learned the optimal policy producing the same optimal truss configuration.

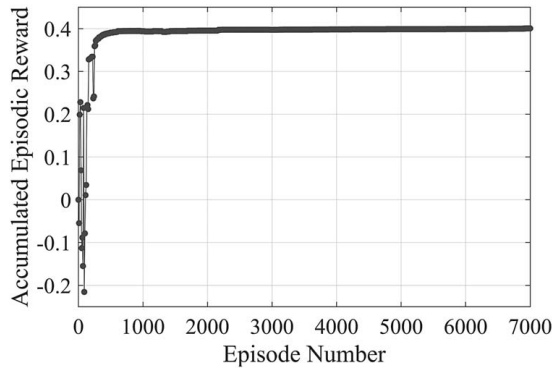


Fig. 8 Average performance of the agent at every ten episodes for Example 4. For each experiment, the agent learned the optimal policy producing the same optimal truss configuration.

The maximum reward that the agent can attain is 0.4097, again unknown to the agent at the onset. Figure 8 presents the average accumulated episodic reward of the agent. On average, the agent evaluated 1176 truss configurations per experiment to learn the optimal policy.

6 Comparison of the MDP-RL Synthesis Framework With Other Methods

The results of the numerical examples employing the presented MDP-RL synthesis framework were compared to the results of two other methods applicable to the design problem; specifically a genetic algorithm (GA) and an exhaustive evaluation. GAs have been used for a wide-range of applications and are amenable to many problem formulations, including those with discrete design variables as well as linear-elastic and elastic-plastic material behaviors. The exhaustive evaluation was performed to verify the accuracy of the solution of the MDP-RL synthesis framework. The GA used here was adapted from Hayashi and Ohsaki [20], and the results, in terms of solution and number of model evaluations, of the GA were compared to those of the MDP-RL synthesis framework. Pseudo-code for the GA used in this study is presented in Algorithm 2.

The GA in Algorithm 2 was adapted to be compatible with the previously described design examples so that the outcome of the GA is directly comparable to the MDP-RL synthesis framework. For the GA, each candidate truss solution's displacement is evaluated again using either a linear or nonlinear FEA as described in Sec. 4, and the same corresponding material properties described in Sec. 5 were specified. The objective of the GA is to minimize the displacement of the truss under the specified external force(s); however, to account for the volume constraint V_{cnst} in the GA, the fitness function $F(\mathbf{x})$ is augmented using a penalty term $C(\mathbf{x})$ as follows:

$$F(\mathbf{x}) = u(\mathbf{x}) + \beta C(\mathbf{x}) \quad (8a)$$

$$C(\mathbf{x}) = \max\left(\left(\frac{V(\mathbf{x})}{V_{cnst}} - 1\right), 0\right) \quad (8b)$$

where β is a penalty coefficient for the volume constraint and is set to 1000 for all examples. Since crossing elements are not currently permitted in the MDP-RL synthesis framework, a death penalty scheme is employed in the GA for design realizations that possess crossing elements so as to converge to a solution that is directly comparable with that of the MDP-RL synthesis framework. Thus, in the GA, designs that have crossing elements are rejected from the population and replaced with randomly selected stable structures that do not possess crossing elements and satisfy the volume constraint. This scheme was implemented to aid the GA in converging to the optimized solution. Other than the optimized solution, the primary

computational metric for comparing the truss MDP-RL synthesis framework with the GA is the number of model evaluations.

Algorithm 2 Genetic algorithm

Data: Population size n_p , initial population $\mathbf{X} = \{x_1, \dots, x_{n_p}\}$, set of feasible configurations without crossing elements \mathbf{X}_f , number of genes n_m (elements), fitness function $F(x)$, elite rate $r_e (= 0.2)$, mutation probability $r_m (= 0.1)$, stopping criteria $n_s (= 10)$

```

1   $I \leftarrow 0$ 
2   $n_e \leftarrow \text{round}(r_e n_p)$ 
3   $f_b \leftarrow \infty$ 
4  while  $I \leq n_s$  do
5     $I \leftarrow I + 1$ 
6    Perform selection:
7    for  $i = 1:n_p$  do
8      if elements in  $x_i$  cross then
9        Replace  $x_i$  with randomly selected design from  $\mathbf{X}_f$ 
10      $f_i \leftarrow F(x_i)$ 
11     if  $f_i \leq f_b$  then
12        $x_b \leftarrow x_i$ 
13        $f_b \leftarrow f_i$ 
14        $I \leftarrow 0$ 
15    $\mathbf{X}_e \leftarrow n_e$  elite solutions from  $\mathbf{X}$ 
16    $\mathbf{X} \leftarrow \mathbf{X}_e$ 
17   Perform crossover:
18   Select parents:  $x_1, x_2 \leftarrow$  randomly selected from  $\mathbf{X}_e$ 
19   Perform one bit crossover, where  $x_{i,j}$  represents a gene of design  $x_i$ :
20    $k \leftarrow$  randomly selected from  $\{1, \dots, n_m - 1\}$ 
21    $x_1 \leftarrow [x_{1,j}, \dots, x_{1,k}, x_{2,k+1}, \dots, x_{2,n_m}]$ 
22    $x_2 \leftarrow [x_{2,j}, \dots, x_{2,k}, x_{1,k+1}, \dots, x_{1,n_m}]$ 
23   Append  $x_1$  and  $x_2$  into  $\mathbf{X}$ 
24   Perform mutation:
25   for  $i = n_e + 1:n_p$  do
26     for  $k = 1:n_m$  do
27       if a sample from uniform distribution  $[0, 1] < r_m$  then
28         if  $x_{i,k} = 1$  then
29            $x_{i,k} \leftarrow 10^{-6}$ 
30         else
31            $x_{i,k} \leftarrow 1$ 
32   Append  $x_i$  into  $\mathbf{X}$ 
33 return  $x_b$ 

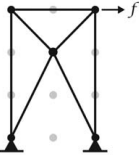
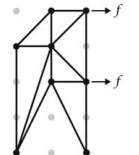
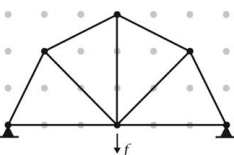
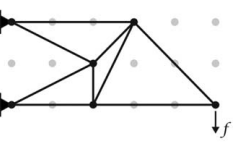
```

For the GA, a candidate truss solution is obtained based on the initial guess of a densely connected ground structure [21] and is represented as a chromosome \mathbf{x} , which is a matrix of genes. Each gene represents a structural element's cross-sectional area that can take on a value of $A_i \in [10^{-6}, 1]$. To prevent singularity, a value of 10^{-6} is assigned to elements instead of 0. The total number of genes, n_m , for each numerical example is determined based on the number of nodal pairs available given a specified domain ($n_m = n_n(n_n - 1)/2$, where n_n is the total number of nodes).

With the GA, initial populations were randomly generated for each experiment. A population is composed of n_p chromosomes, and the algorithm is terminated if the best fitness function value is not improved after ten consecutive generations ($n_s = 10$). For each example, multiple population sizes were considered and are reported in Table 2. The truss with the greatest fitness value, after termination, is determined to be the optimized truss solution x_b . As with the MDP-RL synthesis framework, the GA was evaluated based on the average of 50 experiments.

The MDP-RL synthesis framework was compared to the GA in terms of identified solution, the number of model evaluations required to obtain the solution, and the average computational time required for the model evaluations. The computational times

Table 2 Comparison of results obtained from MDP-RL synthesis method, GA, and exhaustive evaluation

Optimal truss design	Material behavior	Approach	Average model evaluations	Average time (s)	Displacement u of optimized truss	
					Maximum	Minimum
 <p>$u_{\text{opt}} = 0.0895$ $V_{\text{opt}} = 153.0$</p>	Linear-elastic	MDP-RL synthesis	183	8.2	0.0895	0.0895
		GA				
		$n_p = 100$	846	39.3	0.0926	0.0895
		$n_p = 200$	1181	52.9	0.0898	0.0895
		$n_p = 300$	1356	63.0	0.0898	0.0895
		Exhaustive	1058	49.2	0.0895	0.0895
 <p>$u_{\text{opt}} = 0.1859$ $V_{\text{opt}} = 238.8$</p>	Linear-elastic	MDP-RL synthesis	1334	57.8	0.1859	0.1859
		GA				
		$n_p = 300$	3607	172.4	0.2022	0.1859
		$n_p = 600$	5776	250.2	0.1920	0.1859
		$n_p = 1200$	8043	390.1	0.1877	0.1859
		Exhaustive	9836	479.7	0.1859	0.1859
 <p>$u_{\text{opt}} = 0.0425$ $V_{\text{opt}} = 236.0$</p>	Linear-elastic	MDP-RL synthesis	324	36.2	0.0425	0.0425
		GA				
		$n_p = 100$	692	74.4	0.0438	0.0425
		$n_p = 200$	846	81.4	0.0438	0.0425
		$n_p = 300$	883	94.9	0.0438	0.0425
		Exhaustive	903	97.1	0.0425	0.0425
 <p>$u_{\text{opt}} = 0.2291$ $V_{\text{opt}} = 199.5$</p>	Elastic-plastic	MDP-RL synthesis	1176	137.5	0.2291	0.2291
		GA				
		$n_p = 200$	1764	290.4	0.2501	0.2291
		$n_p = 300$	2060	426.2	0.2454	0.2291
		$n_p = 600$	2569	531.5	0.2454	0.2291
		Exhaustive	2788	593.6	0.2291	0.2291

reported are with respect to an Intel Core i7-3770 CPU @ 3.40 GHz machine with 24.0 GB RAM. In addition, both the MDP-RL synthesis framework and the GA were compared to an exhaustive evaluation. The results from the MDP-RL synthesis framework, GA, and exhaustive evaluation are reported in Table 2. All optimal truss configurations presented in Table 2 are global optimal solutions verified using the exhaustive evaluation.

In Table 2, observe that the MDP-RL synthesis has identified the global optimal solution in all experiments for each example, whereas the GA, on occasion, converges to a suboptimal solution, as indicated by the range in displacement of the optimized truss. Although increasing the GAs' population size allowed for solutions to be identified with a smaller range of displacements, both the average number of model evaluations and computational time increased, further demonstrating that the MDP-RL synthesis framework determined the global optimal solution more efficiently.

Based on these numerical examples, the MDP-RL synthesis framework is more efficient than the GA in terms of the number of model evaluations and computational time. In particular, the MDP-RL synthesis framework requires fewer model evaluations to identify the global optimal design than the GA, specifically at least 50%

fewer model evaluations for the first three examples and 30% fewer evaluations for the fourth example. In the first example, the MDP-RL synthesis framework required on average 183 model evaluations, which on average took 8.2 s to evaluate, and determined the global optimal design across all experiments. In contrast, the GA required a range of 846 to 1356 model evaluations for population sizes between 100 and 300, and on average took between 39.3 and 49.2 s, respectively. Even with increased population sizes, the GA, for example, 1, is unable to identify the global optimal solution for all experiments. Population sizes greater than 300 were not considered for the GA in example 1 since these would further result in a greater number of model evaluations compared to the exhaustive evaluation. This is because the GA can explore truss configurations in the infeasible design space, whereas the exhaustive evaluation only evaluates feasible truss configurations. Similarly for the remaining examples, the MDP-RL synthesis framework outperformed the GA in terms of model evaluations and computational time for all considered population sizes. Furthermore, as the GA population size increased for each example, the number of model evaluations approached that of the exhaustive evaluation and still the GA was not able to identify the global optimal solution for all experiments.

In comparison to an exhaustive evaluation, the MDP-RL synthesis framework required significantly fewer model evaluations. Specifically, the MDP-RL synthesis framework achieved greater than 50% savings in model evaluations and computational time for all numerical examples while determining the global optimal solution.

7 Summary and Closing Remarks

This article presents a framework whereby optimal design synthesis is mathematically formulated as a finite MDP and solved with a tabular RL algorithm, specifically Q-learning. Topological grammar rules were modeled through the actions in the MDP to allow for the generation of new design configurations. The utility of the developed framework was demonstrated through application to a planar truss optimization problem with binary areas under the objective of minimizing displacement for a given external force(s) subject to volume and stability constraints. To demonstrate the generality and versatility of the MDP-RL synthesis framework, four numerical examples were investigated. The general framework and rules remained the same for all examples, the only differences being the specifics of the problem, that is domain, seed configuration, volume constraint, external force(s), as well as linear-elastic and elastic-plastic material behaviors. Each numerical example served to illustrate that the agent identifies an optimal policy and that the optimal solution is in the optimal policy for problems with different characteristics based on relatively straightforward and general rules for altering a given configuration.

A comparative assessment of the MDP-RL synthesis framework was made to benchmark its efficiency and accuracy against a GA adapted from Hayashi and Ohsaki [20] and an exhaustive evaluation of all feasible solutions. From this comparison, the MDP-RL synthesis framework was shown to outperform both the GA and exhaustive evaluation in terms of model evaluations and in the case of GAs convergence to global optima. Specifically, the MDP-RL synthesis framework was shown to determine the global optimal solution for all numerical examples with significantly fewer number of designs evaluated than required by both the GA and exhaustive search. The MDP-RL synthesis framework required a fewer number of model evaluations because with RL, specifically Q-learning, an optimal policy is often determined well in advance of the action-value function's convergence [22]. Thus, it is not an absolute requirement to wait for the action-value function to converge before determining an optimal policy capable of altering the seed configuration into the optimal solution.

This article considered topological grammars that altered design configurations in the context of MDPs. However, the framework is sufficiently general such that it can be extended to consider parametric and spatial grammars, where there are more than two parametric choices (0 or 1). In the context of truss optimization, this corresponds to multiple discrete values for cross-sectional areas. This and other possible extensions, for example, large nodal domains, three-dimensional design problems, and large volume allocations, will benefit from alternative RL algorithms, such as policy and function approximation, that have been shown to efficiently solve high-dimensional planning and learning problems. These topics are the subject of ongoing research.

Acknowledgment

The authors gratefully acknowledge support of the Penn State ROCKET Seed Grant. All opinions, findings, and conclusions or recommendations 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. The authors attest that all data for this study are included in the paper.

References

- [1] Antonsson, E. K., and Cagan, J., 2005, *Formal Engineering Design Synthesis*, Cambridge University Press, Cambridge, UK.
- [2] Chakrabarti, A., 2013, *Engineering Design Synthesis: Understanding, Approaches and Tools*, Springer Science & Business Media, New York.
- [3] Campbell, M. I., and Shea, K., 2014, "Computational Design Synthesis," *AI EDAM*, **28**(3), pp. 207–208.
- [4] Hooshmand, A., and Campbell, M. I., 2016, "Truss Layout Design and Optimization Using a Generative Synthesis Approach," *Comput. Struct.*, **163**, pp. 1–28.
- [5] Cagan, J., Campbell, M. I., Finger, S., and Tomiyama, T., 2005, "A Framework for Computational Design Synthesis: Model and Applications," *ASME J. Comput. Inf. Sci. Eng.*, **5**(3), pp. 171–181.
- [6] Swantner, A., and Campbell, M. I., 2012, "Topological and Parametric Optimization of Gear Trains," *Eng. Optim.*, **44**(11), pp. 1351–1368.
- [7] Shea, K., and Cagan, J., 1997, "Innovative Dome Design: Applying Geodesic Patterns With Shape Annealing," *AI EDAM*, **11**(5), pp. 379–394.
- [8] Lin, Y.-C., Shea, K., Johnson, A., Coultate, J., and Pears, J., 2009, "A Method and Software Tool for Automated Gearbox Synthesis," *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, San Diego, CA, Aug. 30–Sept. 2, Vol. 49026, pp. 111–121.
- [9] Puentes, L., Cagan, J., and McComb, C., 2020, "Heuristic-Guided Solution Search Through a Two-Tiered Design Grammar," *ASME J. Comput. Inf. Sci. Eng.*, **20**(1), p. 011008.
- [10] Tai, K., Cui, G. Y., and Ray, T., 2002, "Design Synthesis of Path Generating Compliant Mechanisms by Evolutionary Optimization of Topology and Shape," *ASME J. Mech. Des.*, **124**(3), pp. 492–500.
- [11] Königseder, C., Shea, K., and Campbell, M. I., 2013, "Comparing a Graph-Grammar Approach to Genetic Algorithms for Computational Synthesis of PV Arrays," *CIRP Design 2012*, Bangalore, India, Mar. 28–31, Springer, pp. 105–114.
- [12] Vale, C. A., and Shea, K., 2003, "A Machine Learning-Based Approach to Accelerating Computational Design Synthesis," *DS 31: Proceedings of ICED 03*, the 14th International Conference on Engineering Design, Stockholm, Sweden, Aug. 19–21.
- [13] 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," *ASME 2016 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Charlotte, NC, Aug. 21–24.
- [14] Dering, M. L., and Tucker, C. S., 2017, "Generative Adversarial Networks for Increasing the Veracity of Big Data," *2017 IEEE International Conference on Big Data*, Boston, MA, Dec. 11–14, IEEE, pp. 2595–2602.
- [15] Dering, M. L., and Tucker, C. S., 2017, "Implications of Generative Models in Government," *2017 AAAI Fall Symposium Series*, Arlington, VA, Nov. 9–11.
- [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. Multidiscip. Optim.*, **59**(3), pp. 787–799.
- [18] Jang, S., Yoo, S., and Kang, N., 2020, "Generative Design by Reinforcement Learning: Enhancing the Diversity of Topology Optimization Designs," *arXiv preprint arXiv:2008.07119*.
- [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] 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.
- [21] Dorn, W. S., Gomory, R. E., and Greenberg, H. J., 1964, "Automatic Design of Optimal Structures," *J. Mec.*, **3**(1), pp. 25–52.
- [22] Watkins, C. J., and Dayan, P., 1992, "Q-learning," *Mach. Learn.*, **8**(3–4), pp. 279–292.
- [23] Sutton, R. S., and Barto, A. G., 2018, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA.
- [24] Bellman, R., 1957, *Dynamic Programming*, Princeton University Press, Princeton, NJ.
- [25] Kaelbling, L. P., Littman, M. L., and Moore, A. W., 1996, "Reinforcement Learning: A Survey," *J. Artif. Intell. Res.*, **4**, pp. 237–285.
- [26] Lipson, H., 2008, "Evolutionary Synthesis of Kinematic Mechanisms," *AI EDAM*, **22**(3), pp. 195–205.
- [27] Christensen, P. W., and Klarbring, A., 2008, *An Introduction to Structural Optimization*, Vol. 153, Springer Science & Business Media, New York.

- [28] Bendsoe, M. P., and Sigmund, O., 2013, *Topology Optimization: Theory, Methods, and Applications*, Springer Science & Business Media, New York.
- [29] Rozvany, G. I., 2009, "A Critical Review of Established Methods of Structural Topology Optimization," *Struct. Multidiscipl. Optim.*, **37**(3), pp. 217–237.
- [30] Sigmund, O., and Maute, K., 2013, "Topology Optimization Approaches," *Struct. Multidiscipl. Optim.*, **48**(6), pp. 1031–1055.
- [31] Achtziger, W., and Stolpe, M., 2008, "Global Optimization of Truss Topology With Discrete Bar Areas—Part I: Theory of Relaxed Problems," *Computat. Optim. Appl.*, **40**(2), pp. 247–280.
- [32] Achtziger, W., and Stolpe, M., 2009, "Global Optimization of Truss Topology With Discrete Bar Areas—Part II: Implementation and Numerical Results," *Computat. Optim. Appl.*, **44**(2), p. 315.
- [33] Stolpe, M., 2015, "Truss Topology Optimization With Discrete Design Variables by Outer Approximation," *J. Global Optim.*, **61**(1), pp. 139–163.
- [34] Kaveh, A., and Talatahari, S., 2009, "Particle Swarm Optimizer, Ant Colony Strategy and Harmony Search Scheme Hybridized for Optimization of Truss Structures," *Comput. Struct.*, **87**(5–6), pp. 267–283.
- [35] Li, L., Huang, Z., and Liu, F., 2009, "A Heuristic Particle Swarm Optimization Method for Truss Structures With Discrete Variables," *Comput. Struct.*, **87**(7–8), pp. 435–443.
- [36] Kripka, M., 2004, "Discrete Optimization of Trusses by Simulated Annealing," *J. Braz. Soc. Mech. Sci. Eng.*, **26**(2), pp. 170–173.
- [37] Kaveh, A., and Kalatjari, V., 2003, "Topology Optimization of Trusses Using Genetic Algorithm, Force Method and Graph Theory," *Int. J. Numer. Methods Eng.*, **58**(5), pp. 771–791.
- [38] Wu, S.-J., and Chow, P.-T., 1994, "Genetic Algorithms for Solving Mixed-Discrete Optimization Problems," *J. Franklin Inst.*, **331**(4), pp. 381–401.
- [39] Sigmund, O., 2011, "On the Usefulness of Non-Gradient Approaches in Topology Optimization," *Struct. Multidiscipl. Optim.*, **43**(5), pp. 589–596.
- [40] Bendsoe, M. P., and Sigmund, O., 1995, *Optimization of Structural Topology, Shape, and Material*, Vol. 414, Springer, New York.
- [41] Stromberg, L. L., Beghini, A., Baker, W. F., and Paulino, G. H., 2012, "Topology Optimization for Braced Frames: Combining Continuum and Beam/Column Elements," *Eng. Struct.*, **37**, pp. 106–124.
- [42] Stolpe, M., 2016, "Truss Optimization With Discrete Design Variables: A Critical Review," *Struct. Multidiscipl. Optim.*, **53**(2), pp. 349–374.
- [43] Bathe, K.-J., 2006, *Finite Element Procedures*, Klaus-Jurgen Bathe, Watertown, MA.
- [44] Mazzoni, S., McKenna, F., Scott, M. H., and Fenves, G. L., 2006, "OpenSees Command Language Manual," Pac. Earthq. Eng. Res. (PEER) Center, p. 264.
- [45] Ororbia, M. E., and Warn, G. P., 2020, "Structural Design Synthesis Through a Sequential Decision Process," ASME 2020 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Virtual Online, Aug. 17–19.