Proceedings of the ASME 2020
International Design Engineering Technical Conferences
and Computers and Information in Engineering Conference
IDETC/CIE2020
August 17-19, 2020, Virtual, Online

DETC2020-22647

STRUCTURAL DESIGN SYNTHESIS THROUGH A SEQUENTIAL DECISION PROCESS

Maximilian E. Ororbia[1], **Gordon P. Warn**[1]

[1]The Pennsylvania State University, State College, PA

ABSTRACT

This article illustrates that structural design synthesis can be achieved through a sequential decision process, whereby a sparsely connected seed configuration is sequentially altered through discrete actions to generate the best design solution, with respect to a specified objective and constraints. Specifically, the generative design synthesis is mathematically formulated as a finite Markov Decision Process. In this context, the *states* correspond to a specific structural configuration, the *actions* correspond to the available alterations that can be made to a given configuration, and the immediate *rewards* are constructed to be proportional to the improvement in the altered configuration's performance. In the context of generative structural design synthesis, since the immediate rewards are not known at the onset of the process, reinforcement learning is employed to obtain an approximately optimal policy by which to alter the seed configuration to synthesize the best design solution. The approach is applied for the optimization of planar truss structures and its utility is investigated with three numerical examples, each with unique domains and constraints.

INTRODUCTION

There exist a variety of paradigms for engineering design that are generally accepted and widely studied, each having advantages for particular applications. Although a complete list and comprehensive discussion of each is beyond the scope of this paper, representative examples include set-based design [1–3], surrogate modeling [4–6], evolutionary algorithms [7], and gradient-based size, shape, and topology optimization [8, 9]. Recently, generative design tools employing deep learning are emerging for the purpose of generating design concepts [10–12], and for the purpose of design space exploration [13].

With respect to structural design, gradient based topology optimization is widely employed for a variety of design problems [8, 9, 14, 15] that conventionally begin with a densely meshed solid, or ground structure, which is iteratively eroded by removing elements until optimality is achieved. In contrast, it has been suggested in the literature that design synthesis can be achieved through a *generative* process [16–18] whereby a sparse initial design is altered, increasing in complexity, until optimality is achieved. In this study, the generative viewpoint is adopted; however, unique to this research is formulating the generative structural design synthesis as a finite Markov Decision Process (MDP) that is solved with reinforcement learning (RL).

Hence, the objective of this research is to formulate structural design synthesis as a sequential decision process, whereby a sparsely connected seed configuration is sequentially altered through discrete actions to generate an optimal design. Since the outcome of an alteration on a given configuration is only partially known, the structural design synthesis is mathematically modeled as a finite MDP. A finite MDP is a discrete time stochastic control process, where at a time an agent finds itself in a given state, takes an action, receives an immediate reward, and transitions to a new state. In the context of structural design synthesis, the *states* correspond to specific structural configurations, the *actions* correspond to the available alterations that can be made to a given configuration, and the immediate *rewards* are constructed to be proportional to the improvement in the altered configura-

tion's performance. Since the degree of improvement obtained for a given alteration is unknown at the onset of the process, that is the immediate rewards are unknown, RL is employed to determine an approximately optimal policy by which to alter the seed configuration to synthesize the best design solution with respect to the given design problem. RL is a sub-class of machine learning, that differs from other methods in that an agent continually interacts with its environment, taking actions and receiving immediate rewards with the goal of maximizing the cumulative reward over the long-run.

Although the suggested process being a *generative* approach differs from traditional degenerative approaches, the outcomes of both processes can be comparable depending on how the design problem is posed. However, formulating design synthesis as a generative process could offer some benefits. For example, the best topology is often comprised of few elements and as such will typically be closer to the seed configuration in the generative approach than it is to a fully connected ground structure in a degenerative approach. Thus, the RL agent need learn only to take a few actions that alter the initial seed configuration to generate the best structural design solution. Furthermore, starting with a sparse design translates into computationally efficient model evaluations, in particular early in the optimization process and perhaps throughout depending on the specifics of the best structural design solution. To this end, the suggested approach is applied for the design of planar truss structures, and its utility is investigated through three numerical examples solved using a tabular Q-learning (RL) algorithm.

The remainder of the paper is organized as follows. In the following section, a brief background on finite MDPs and RL is discussed. Thereafter the generative design synthesis methodology is introduced followed by the application of the methodology for the design of planar trusses with the objective of minimizing displacement for a given external force subject to a volume constraint. Following the application section, three numerical examples are presented for the design of planar trusses with different nodal domains and volume constraints to illustrate the structural design synthesis process and solution using a tabular Q-learning algorithm.

BACKGROUND

Finite Markov Decision Process

A finite MDP is defined by a 4 tuple ($\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}$, $\mathcal{R}$), where $\mathcal{S}$ is a finite set of all the 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. 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, $\pi_t(a|s) : \mathcal{S} \rightarrow \mathcal{A}$, 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 expected discounted return:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R \tag{1}$$

where γ is the discount rate. The value of the discount rate determines the importance of a reward received k time steps in the future given that 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 represented by the action-value function, denoted as $q_\pi(s, a)$, and is defined as:

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

where $\mathbb{E}_\pi[\cdot]$ is the expected discounted return of starting at state s, taking an action a, and following a policy π thereafter, also referred to as the *Bellman* equation [19]. The corresponding optimal action-value function is defined as follows:

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

Although it is possible to have more than one optimal policy corresponding to equation (3), a greedy optimal policy that deterministically chooses the action with the maximum action-value in each state $s \in \mathcal{S}$ is:

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

An optimal policy can be determined using a suitable dynamic programming (DP) algorithm, such as generalized policy iteration [20], to first solve equation (3), then by choosing greedy actions in each state according to equation (4). However, when 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 identify the approximately optimal policy.

Reinforcement Learning

Reinforcement learning (RL) is a sub-class of machine learning where an agent learns to take optimal actions that maximize its

reward over the long run by continually interacting with its environment [20]. RL methods maximize the cumulative of a numerical reward signal by mapping different states in a sequential decision process to optimal actions.

Central to RL is the idea of temporal-difference (TD) learning [20]. 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 [20].

Q-learning [21] is an off-policy TD control algorithm where the action-value function is updated according to:

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

In equation (5), $Q(s_t, a_t)$ is the learned action-value function that directly approximates equation (2) regardless of the policy being followed. The bracketed quantity, representing the temporal difference, $[R_{t+1} + \gamma \max_{a \in A(s_{t+1})} Q(s_{t+1}, a_{t+1})]$, is the bootstrapped sample estimate of the quantity in the expectation operator of equation (2). The learning rate, denoted as α_t, controls the weight of the update.

For a learning task, the action-value function is initialized with an arbitrary value, Q_0. 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. The immediate reward if entering a terminal state is zero and 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 converge to an optimal action-value function in a stationary environment with discrete domains [20,21]. However, 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-ϵ) 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.

METHODOLOGY

Design Synthesis as a finite Markov Decision Process

This section describes the formulation of design synthesis as a finite MDP. In the context of structural design, a state, s_t, corresponds to a given structural configuration at time t and an action, $a \in \mathcal{A}(s_t)$, is the action available to make an alteration to the given configuration resulting in a new configuration and hence the next state, s_{t+1}. For now, we defer the discussion of rewards, $\mathcal{R}$, and transition probabilities, $\mathcal{P}$, to the subsequent section since both are problem specific.

The design synthesis process starts with an initial state, s_0, that is a seed configuration which is then altered by a sequence of discrete actions chosen by an agent. The seed configuration is composed of a sparsely connected set of nodes selected from a specified nodal domain. Adapted from Lipson [22], 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, T or D. 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 to the existing configuration without removing an element. Figure 1 provides an illustration of a seed configuration altered by two actions and the resulting new states, that is updated configurations, as a result of the D and T operators.

For the structural design synthesis an action a consists of three components: (1) selection of an available inactive node, (2) selection of an eligible element for that node, and (3) choice of

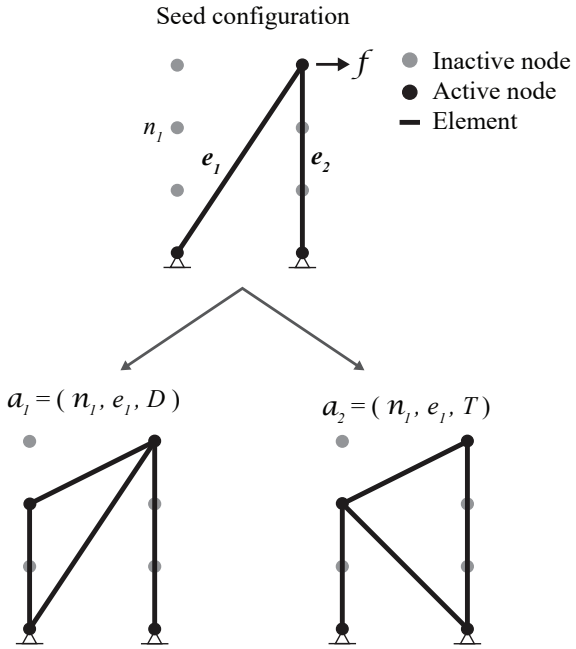

FIGURE 1: Example of a seed configuration altered by two actions, a_1 and a_2, and the associated new states. Specifically T and D operations on element 1, e_1, and node 1, n_1, generate two new truss designs.

a legal operator based on the location of the node relative to the selected element. As such, an action set is defined as:

$$\mathcal{A}(s_t) = \{(n_i, e_j, o_k) \mid n_i \in N_{avail.}(s_t), \\ e_j \in E_{allow}(s_t, n_i), o_k \in O_{legal}(n_i, e_j)\} \quad (6)$$

where, depending on the given configuration at s_t, 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 domain node dictates which elements and operators comprise the set of actions. In particular, a T operator can be applied to an element for any selected available node. However, if an available node is intersected by an existing element, then a D operator cannot be applied to that element. Furthermore, if the selected node is 'inside' the structural configuration then only a single D operator can be applied to any of the immediate surrounding elements. Lastly, elements are connected between the selected node and other active nodes as long as they do not intersect any other existing elements. This restriction prevents redundant structures from forming as a result of different actions on a given configuration. As an example, in Figure 1, an action of $a_1 = (n_1, e_1, D)$ alters the seed configuration by activating node n_1 and generating two elements that connect to the seed configurations without removing e_1. Alternatively, the action $a_2 = (n_1, e_1, T)$ alters the seed configuration by activating node n_1 and generating new elements after removing element e_1.

The process of altering a configuration is repeated until a terminal state s_T is reached. A terminal state is reached if all available nodes in the domain are active, or if an altered configuration is unstable. Other terminal states will exist depending on the design objective and constraints, as discussed in the subsequent section. The described initial states, transition states, and actions are generally applicable for the proposed structural design synthesis approach. However, the rewards, transition probabilities, and additional terminal states vary depending on the specified design objective and constraints. In the following sections, one possible design problem is presented, including specifics regarding the terminal states, transition probabilities, and rewards for the given design objective along with numerical examples for different nodal domains and constraint values.

APPLICATION

Design Problem

The proposed methodology is applied to a design task involving the structural design synthesis of a planar truss given a specified nodal domain and seed configuration. The design goal is to generate a truss topology that minimizes displacement under an externally applied force, 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 topology optimization of truss structures [8, 23]. The displacement of each truss configuration is evaluated using finite element analysis (FEA) as subsequently described.

Finite Element Analysis

The performance of each truss configuration is evaluated using a linear FEA. Element local stiffness matrices, k_{ele}, are transformed, according to a given truss configuration, using the conventional coordinate transformation matrices [24], and then mapped to the global system equilibrium equations: $F_g - K_g u_g = 0$. After assembling the global equilibrium equations, the appropriate boundary conditions are applied at the system level to obtain the system equations: $F_s - K_s u_s = 0$, where F_s is the externally applied load vector, K_s is the partitioned global stiffness matrix of the system, and u_s is the vector of free nodal displacements. Stability of each truss configuration is assessed by checking the condition of its partitioned global stiffness matrix.

Learning Environment

The design synthesis problem is formulated as an undiscounted (γ = 1), episodic, fintite MDP and solved using a tabular Q-learning (RL) algorithm. Each episode starts with an initial state, s_0, represented by a seed configuration, and subsequently the RL agent takes an action, as described in the methodology section, transitioning the seed configuration to a new state s_{t+1}. The RL agent continues to alter the given configuration until one of three terminal states is reached, hence ending the episode. Specifically, a state is considered terminal if there are no available inactive nodes for the RL agent to select, the altered truss structure is unstable, or if the volume constraint is violated. 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.

The goal of the RL agent is to maximize its cumulative rewards over the long run. In the context of this design problem, the RL agent determines an optimal sequence of alterations to the seed configuration that maximize the improvement in the performance of the design with respect to the objective, and hence synthesizes the best design solution, which is a feasible design with the least displacement. The reward signal is constructed so that the best performing design solution is in the optimal policy. Specifically, the immediate reward for this application is proportional to the change in performance of the design in s_t with the altered configuration's in s_{t+1}. In other words, the agent receives a reward R_{t+1} determined by evaluating the performance of the new design in s_{t+1}. If the displacement of the altered configuration in s_{t+1} is less than the configuration's in s_t then the RL agent receives a positive reward equal to the difference, whereas a larger displacement results in a negative reward. A reward of zero is assigned if the agent transitions to a terminal state.

In this study, the RL agent follows an ϵ-greedy policy in which the exploration rate, ϵ_t, decays over a specified number of episodes. The exploration rate is decayed using a cosinusoidal function of $\epsilon_t = A_\epsilon \cos(B_\epsilon \mathrm{e})$, where A_ϵ and B_ϵ are parameters used to adjust 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. For a set of episodes, constituting an experimental run, both the learning and exploration rates (α_t and ϵ_t) are initialized at $\mathrm{e} = 1$ to $A_\alpha = 1$ and $A_\epsilon = 0.9$ respectively.

NUMERICAL EXAMPLES

Three numerical examples are presented in this section to illustrate and investigate the utility of the suggested generative design synthesis approach applied to the design of planar trusses. The following examples consist of different nodal domains and volume constraints. Each example begins with an asymmetric truss seed configuration and an externally applied force, f, at the upper right most node in the domain. Model parameter values are assigned so that the results of the FEA are non-dimensional. As such, a Young's modulus of $E = 10^3$, an area of $A = 1$, and an external force $f = 10$ are assumed for all examples. The length of each element is determined based on its connectivity. The performance of the Q-learning algorithm is assessed with respect to an exhaustive evaluation of all feasible designs.

Example 1

This example is used to illustrate the method on a small domain allowing the feasible states and state directed graph for the problem to be presented. The number of states significantly increase with an increase in the number of nodes, hence precluding compact graphical representation for domains with a modest and large number of nodes. Therefore, the nodal domain and seed configuration shown in Figure 2 are considered. The seed configuration has a volume of 61.62 and a displacement of 0.59 under the specified external force as determined by FEA. A volume constraint of 100 is set for this design problem.

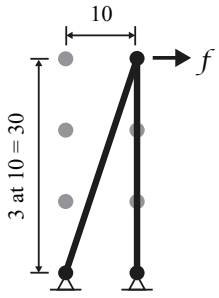

FIGURE 2: Seed configuration: specified nodal domain and seed configuration consisting of five inactive nodes, three active nodes, and two elements.

All truss configurations that the RL agent can transition to, including feasible and terminal states, are illustrated by the state directed graph presented in Figure 3. States that do not exceed the volume constraint, that is feasible states, are shown in blue and states that either exceed the volume constraint or produce an unstable structure, that is terminal states, are shown in grey. Each path branching from the root node indicates a sequence of alterations that the RL agent can make to the seed configuration. For this domain and volume constraint, there are 14 feasible states. The edges and nodes highlighted in green denote the states in the optimal policy as determined by the RL agent.

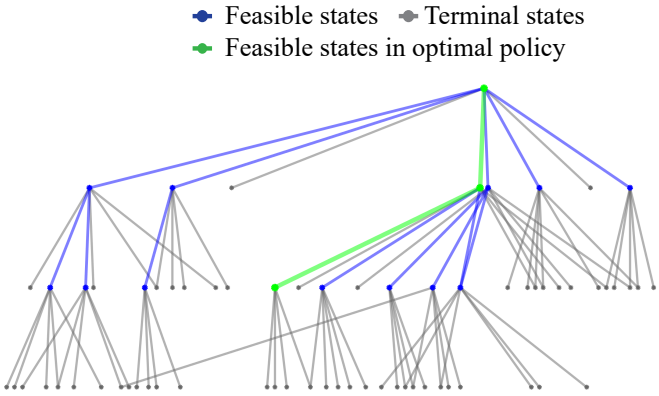

FIGURE 3: State directed graph for the design task with a volume constraint of 100.

The learning and exploration rates for the implemented tabular Q-learning algorithm are initialized as previously described. The learning rate is decayed to a value of 0.1 after 250 episodes by setting $B_\alpha = 0.00585$. The exploration rate is decayed to a value of 0.01 over 230 episodes by setting $B_\epsilon = 0.0068$. Both parameters are then held constant at their respective values for the remaining episodes.

The performance of the Q-learning agent is assessed based on the average of 50 experimental runs, each consisting of 300 episodes. The learning process was paused after every 10 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. Figure 4 presents the performance of the Q-learning agent for this design synthesis example. The maximum reward that the agent can achieve is 0.32, and is indicated by the horizontal grey line. After 150 episodes, the RL agent's accumulated episodic reward approaches the maximum value which indicates that the RL agent learns an optimal policy that maximizes its reward. On average the agent evaluates 13.6 of the total 14 feasible designs per experiment to learn the optimal policy. In comparison to an exhaustive evaluation, which requires the evaluation of all 14 states to determine the best design solution, the RL agent achieves a 2.9% savings. This is to be expected because the agent needs

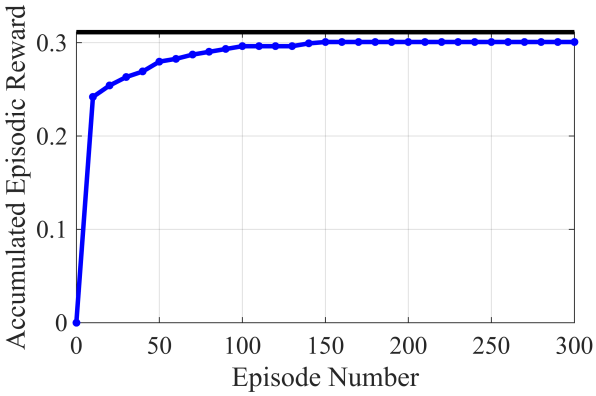

FIGURE 4: The accumulated episodic reward averaged over 50 experimental runs. The RL agent learns the optimal policy and approaches the maximum reward after 150 episodes.

experience to learn and yet there are so few feasible designs that, for this example, little computational savings are gained.

Figure 5 presents an illustration of the feasible altered configurations that the RL agent makes from the learned optimal policy. The design shown for S_2 is the best truss configuration for the given design goal and volume constraint. It has a volume of 92.43 and under the specified external force a displacement of 0.27.

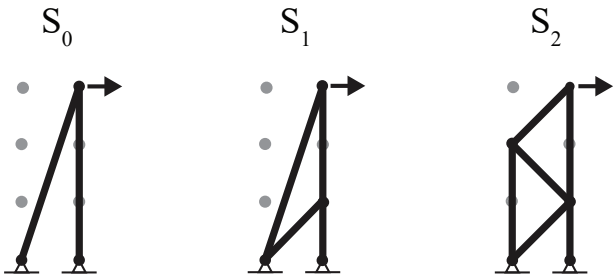

FIGURE 5: Feasible states, designs, visited by the RL agent in the learned optimal policy. The design shown in S_2 represents the optimal truss.

Example 2

To illustrate the effect that the volume constraint has on the set of feasible designs, the optimal policy, and the RL agent's performance, the specified nodal domain in example 1 was repeated with a new volume constraint of 120. The state directed graph for this example is presented in Figure 6. There are a total of 60 feasible states.

The Q-learning algorithm's learning rate, α_t and exploration rate, ϵ_t, are initialized and decayed as described in example 1. Figure 7 presents the performance of the Q-learning agent averaged over 50 experimental runs for the new volume constraint. Similarly, the maximum reward that can be achieved by the agent is 0.32 and the RL agent converges to this value after 170 episodes.

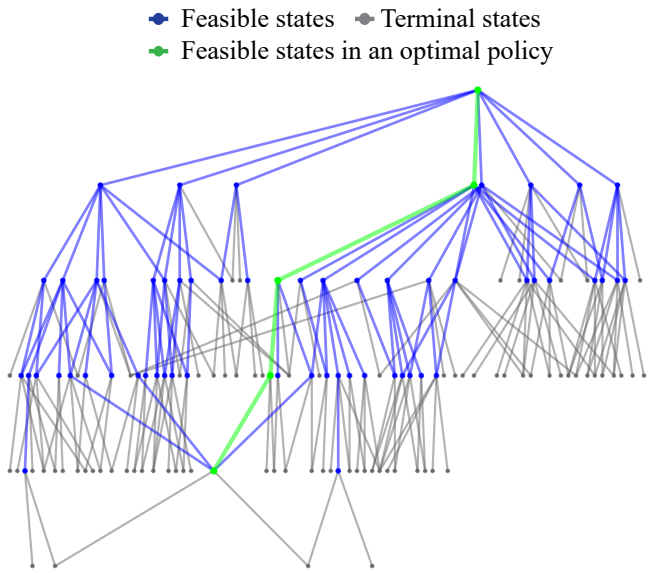

FIGURE 6: State directed graph for a volume constraint of 120.

On average the agent evaluates 51 feasible designs per experiment to converge to the maximum reward and learn an optimal policy. In comparison to an exhaustive evaluation, which requires the evaluation of all 60 feasible states to determine the best design, the RL agent achieves a 15% savings. Increasing the volume constraint for the specified nodal domain in example 1 increased the set of feasible designs while also improving the RL agent's efficiency. The RL agent's increased performance relative to the exhaustive evaluation is in part due to the agent gaining sufficient experience to identify an optimal policy, without

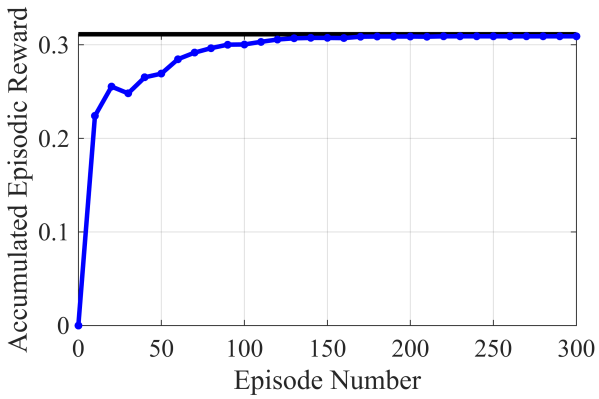

FIGURE 7: The RL agent's averaged performance for a volume constraint of 120.

needing to visit all feasible states.

For this problem there are multiple optimal policies. Figure 6 shows a sample optimal policy and Figure 8 presents an illustration of the feasible altered configurations from the optimal policy. Two additional elements are introduced to the truss design solution shown for S_2 in Figure 5. However, because of the small domain and larger volume constraint, these elements are zero force elements. The design shown for S_4 in Figure 8 has a volume of 112.43 and a displacement of 0.27 under the externally applied force.

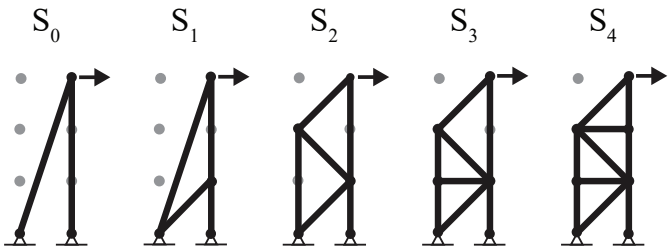

FIGURE 8: Feasible states, designs, visited by the RL agent in a learned optimal policy for a volume constraint of 120. The design shown in S_4 represents the optimal truss.

Example 3

To explore a state space with a greater variability in truss structure configurations, this example applies the generative design synthesis approach on a domain with a larger number of nodes than in the previous examples, albeit still modest. Figure 9 illustrates the specified nodal domain and seed configuration used for this example. The seed configuration has a volume of 66.06 and is determined to have a displacement of 0.1847 under the specified applied load. The volume constraint for this example is set to 160.

Similar to the previous examples, the RL agent's performance is assessed based on the average of 50 experimental runs.

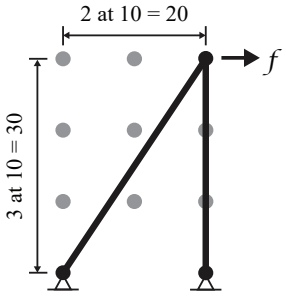

FIGURE 9: Seed configuration: specified nodal domain and seed configuration consisting of eight inactive nodes and three active nodes. Two elements connect the active nodes.

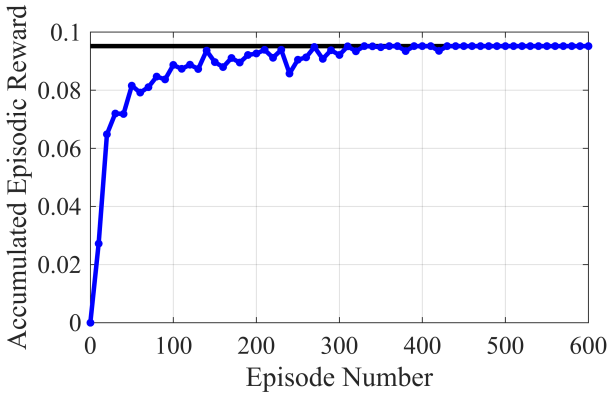

FIGURE 10: Performance of the RL agent averaged over 50 experimental runs.

However for this example, each experimental run consists of 2000 episodes. The learning and exploration rates for the implemented tabular Q-learning algorithm are initialized and decayed as described in the application section. The learning rate is decayed to a value of 0.1 after 1500 episodes by setting $B_\alpha = 0.00097$. The exploration rate is decayed to a value of 0.01 over 1000 episodes by setting $B_\epsilon = 0.00156$. Both the learning and exploration rates are then held constant at their respective values for the remaining episodes.

For the specified volume constraint, there are a total of 1,053 feasible designs. Due to the large number of states, the state directed graph for this example has been omitted. The maximum reward that the agent can achieve is 0.0952. The Q-learning agent converges to the maximum reward after 330 episodes as can be seen in Figure 10, where the performance of the agent is shown for the first 600 episodes since the accumulated reward remained constant for all subsequent episodes. On average the agent evaluated 288 designs per experiment to learn the optimal policy. In comparison to an exhaustive evaluation, which requires the evaluation of all 1,053 feasible states to determine the best design, the RL agent achieves a 73% savings for this example.

The best planar truss configuration that satisfies the volume constraint found in the optimal policy is presented in Figure 11. Generated in the second step, S_2, the best design solution has a volume of 153.01 and a displacement of 0.0895 under the specified external force as determined by FEA. It is worth noting, this design solution is similar to that of the optimal topology found in Stromberg et al. [23], providing some degree of validation, although not comprehensive, of the suggested method.

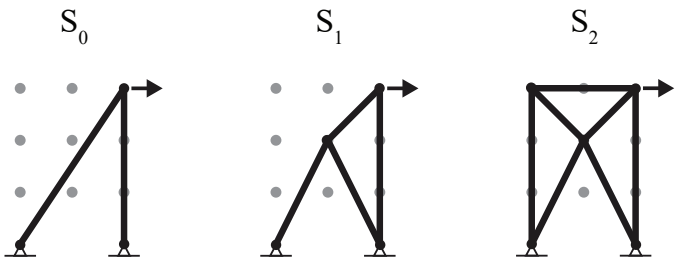

FIGURE 11: Feasible states, designs, visited by the RL agent in the learned optimal policy for a volume constraint of 160. The design shown in S_2 is the optimal truss.

CONCLUSION

This study introduced a methodology whereby structural design synthesis can be mathematically formulated as a finite Markov Decision Process (MDP) and solved by employing reinforcement learning (RL). The utility of the developed methodology was demonstrated through application to a planar truss design problem with the objective of minimizing displacement under a given external force for different nodal domains and volume constraints. The first numerical example served to illustrate, through a small nodal domain, the state space generated using the available finite MDP actions, that the RL agent identifies optimal policy, and that the best performing design solution is in the optimal policy. The second example, for the same domain as the first, demonstrated the effect that the volume constraint has on the size of the set of feasible designs and that some efficiency gains, relative to exhaustive evaluation, are achieved when the set of feasible designs increases. Meaning the RL agent is more efficient when the amount of experience needed to learn the optimal policy is much less than the cardinality of the set of feasible designs. The third example presented a domain with a greater number of nodes and thus a greater number of available feasible states. Furthermore, the third example illustrated that a reasonable efficiency, that is 73% savings over exhaustive evaluation, can be achieved with a tabular implementation of Q-Learning and a modestly sized domain. In other words, the structural design synthesis process, solved with a tabular Q-learning (RL) algorithm, was shown to arrive at the best design solution in less than half the number of designs evaluated by an exhaustive search.

As previously mentioned, the number of feasible states generated significantly increases when the number of nodes increases. As such, in an effort to apply the methodology to more realistic, large scale, structural design problems ongoing research is investigating the implementation of value and policy function approximation methods. Additionally, there is ongoing work to benchmark the suggested method against other widely accepted shortest path and other optimization algorithms.

ACKNOWLEDGMENT

The authors gratefully acknowledge support from the Pennsylvania State University's College of Engineering's 2020 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.

REFERENCES

[1] Ward, A., Liker, J. K., Cristiano, J. J., and Sobek, D. K., 1995. "The second toyota paradox: How delaying decisions can make better cars faster". *Sloan management review,* ***36***(3), p. 43.

[2] Balling, R., 1999. "Design by shopping: A new paradigm?". In Proceedings of the Third World Congress of structural and multidisciplinary optimization (WCSMO-3), Vol. 1, pp. 295–297.

[3] Miller, S. W., Simpson, T. W., Yukish, M. A., Bennett, L. A., Lego, S. E., and Stump, G. M., 2013. "Preference construction, sequential decision making, and trade space exploration". In ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers. Paper no. DETC2013/DAC-13098.

[4] Simpson, T. W., Poplinski, J., Koch, P. N., and Allen, J., 2001. "Metamodels for computer-based engineering design: Survey and recommendations". *Engineering with Computers,* ***17***(2), pp. 129–150.

[5] Forrester, A. I., Sóbester, A., and Keane, A. J., 2007. "Multi-fidelity optimization via surrogate modelling". *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences,* ***463***(2088), pp. 3251–3269.

[6] Viana, F. A., Simpson, T. W., Balabanov, V., and Toropov, V., 2014. "Metamodeling in multidisciplinary design optimization: How far have we really come?". *AIAA Journal,* ***52***(4), pp. 670–690.

[7] Coello, C. A. C., Lamont, G. B., Van Veldhuizen, D. A., et al., 2007. *Evolutionary algorithms for solving multi-objective problems*, Vol. 5. Springer.

[8] Bendsøe, M. P., and Sigmund, O., 1995. *Optimization of structural topology, shape, and material*, Vol. 414. Springer.

[9] Sigmund, O., and Maute, K., 2013. "Topology optimization approaches". *Structural and Multidisciplinary Optimization,* ***48***(6), pp. 1031–1055.

[10] 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". In ASME 2016 International Design Engineering Technical Conferences and Computers and Information in Engi-

neering Conference, American Society of Mechanical Engineers Digital Collection.
[11] Dering, M. L., and Tucker, C. S., 2017. "Generative adversarial networks for increasing the veracity of big data". In 2017 IEEE International Conference on Big Data (Big Data), IEEE, pp. 2595–2602.
[12] Dering, M. L., and Tucker, C. S., 2017. "Implications of generative models in government". In 2017 AAAI Fall Symposium Series.
[13] 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". *Journal of Mechanical Design,* ***142***(7).
[14] Bendsoe, M. P., and Kikuchi, N., 1988. "Generating optimal topologies in structural design using a homogenization method".
[15] Bendsoe, M. P., and Sigmund, O., 2013. *Topology optimization: theory, methods, and applications*. Springer Science & Business Media.
[16] Shea, K., Cagan, J., and Fenves, S. J., 1997. "A shape annealing approach to optimal truss design with dynamic grouping of members".
[17] Vale, C. A., Shea, K., et al., 2003. "A machine learning-based approach to accelerating computational design synthesis". In DS 31: Proceedings of ICED 03, the 14th International Conference on Engineering Design, Stockholm, pp. 183–184.
[18] Vermeer, K., Kuppens, R., and Herder, J. "Kinematic synthesis using reinforcement learning". In ASME 2018 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference.
[19] Bellman, R., 1957. *Dynamic programming*. Princeton University Press.
[20] Sutton, R. S., and Barto, A. G., 2018. *Reinforcement learning: An introduction*. MIT press.
[21] Watkins, C. J., and Dayan, P., 1992. "Q-learning". *Machine learning,* ***8***(3-4), pp. 279–292.
[22] Lipson, H., 2008. "Evolutionary synthesis of kinematic mechanisms". *AI EDAM,* ***22***(3), pp. 195–205.
[23] Stromberg, L. L., Beghini, A., Baker, W. F., and Paulino, G. H., 2012. "Topology optimization for braced frames: combining continuum and beam/column elements". *Engineering Structures,* ***37***, pp. 106–124.
[24] Bathe, K.-J., 2006. *Finite element procedures*. Klaus-Jurgen Bathe.