frontiers | Frontiers in Built Environment

TYPE Original Research
PUBLISHED 23 August 2022
DOI 10.3389/fbuil.2022.899072

Deep deterministic policy gradient and graph convolutional network for bracing direction optimization of grid shells

Chi-tathon Kupwiwat*, Kazuki Hayashi and Makoto Ohsaki

Department of Architecture and Architectural Engineering, Graduate School of Engineering, Kyoto University, Kyoto, Japan

Check for updates

OPEN ACCESS

EDITED BY
Iftikhar Azim,
Shanghai Jiao Tong University, China

REVIEWED BY
Gennaro Senatore,
Swiss Federal Institute of Technology Lausanne, Switzerland
Ge Ou,
The University of Utah, United States

*CORRESPONDENCE
Chi-tathon Kupwiwat,
kupwiwat.chitathon.73c@st.kyoto-u.ac.jp

SPECIALTY SECTION
This article was submitted to Computational Methods in Structural Engineering,
a section of the journal Frontiers in Built Environment

RECEIVED 18 March 2022
ACCEPTED 11 July 2022
PUBLISHED 23 August 2022

CITATION
Kupwiwat C-t, Hayashi K and Ohsaki M (2022), Deep deterministic policy gradient and graph convolutional network for bracing direction optimization of grid shells.
Front. Built Environ. 8:899072.
doi: 10.3389/fbuil.2022.899072

In this paper, we propose a method for bracing direction optimization of grid shells using a Deep Deterministic Policy Gradient (DDPG) and Graph Convolutional Network (GCN). DDPG allows simultaneous adjustment of variables during the optimization process, and GCN allows the DDPG agent to receive data representing the whole structure to determine its actions. The structure is interpreted as a graph where nodes, element properties, and internal forces are represented by the node feature matrix, adjacency matrices, and weighted adjacency matrices. DDPG agent is trained to optimize the bracing directions. The trained agent can find sub-optimal solutions with moderately small computational cost compared to the genetic algorithm. The trained agent can also be applied to structures with different sizes and boundary conditions without retraining. Therefore, when various types of braced grid shells have to be considered in the design process, the proposed method can significantly reduce computational cost for structural analysis.

KEYWORDS

bracing direction optimization, reinforcement learning, deep deterministic policy gradient, graph convolutional network, grid shell structures

1 Introduction

Structural optimization aims to obtain the best design variables that minimize/maximize an objective function under specified constraints (Christensen and Klarbring, 2009). For discrete structures, such as trusses and frames, typically, the design variables are cross-sectional properties, nodal locations and/or nodal connectivity (Ohsaki and Swan, 2002). Finding the best nodal locations is generally called geometry optimization, and the determination of nodal connectivity is called topology optimization. Structural optimization is important in early-stage design of large-span grid shells because their structural performance depends significantly on the shape and topology (Ohsaki, 2010). An optimization problem for grid shells can be formulated to maximize the stiffness against static loads through minimization of the compliance (i.e., elastic strain energy). Examples of such formulation can be found in Refs. (Topping, 1983; Wang et al., 2002; Kociecki and Adeli, 2015).

In topology optimization of grid shells where bracing directions are to be optimized, the optimization problem can be formulated as a combinatorial problem and solved using heuristic approaches such as genetic algorithm (GA) and simulated annealing without utilizing gradient information (Dhingra and Bennage, 1995; Ohsaki, 1995; Kawamura et al., 2002). While this approach allows for simple implementation, it requires many evaluations of the structural response and therefore has a high computational cost especially for structures made of many elements (i.e., many design variables). In addition, the topology optimization problem to minimize compliance can be formulated as mixed-integer programming (MIP) which is practical for small- to medium-size optimization problems due to computational cost (Kanno and Fujita, 2018). Recent advances in MINLP enable to solve very large mixed-integer problems with quadratic and/or bilinear objective function and constraints. However, both heuristic and mathematical programming approaches do not allow the use of knowledge acquired from previously obtained solutions for similar structural configurations.

In recent years, machine learning (ML) approaches have been applied to structural optimization problems. ML can be classified into supervised learning, unsupervised learning, and reinforcement learning (RL). A supervised learning model learns to map (predict or classify) given input instances to specific output domains using sample data for training. Examples of this method for structural optimization can be found in (Berke et al., 1993; Hung et al., 2019; Mai et al., 2021). An unsupervised learning model learns to capture relationships between instances (data). Examples of unsupervised learning are t-distributed stochastic neighbor embedding (t-SNE) (van der Maaten and Hinton, 2008) and k-means clustering (MacQueen, 1967). Jeong and Yoshimura (Jeong and Yoshimura, 2002) also applied an improved unsupervised learning method to multi-objective optimization of plane trusses. Applications of unsupervised learning methods for structural design and structural damage detection can be found in (Eltouny and Liang, 2021; Puentes et al., 2021).

RL is a type of ML that has been developed from optimal control and dynamic programming (Sutton and Andrew, 2018). In RL, a model, or agent, is allowed to interact with an environment. The agent adjusts its policy to take actions according to given reward signals, which are designed to encourage the agent to do actions that change the environment into a desirable state such as winning a game or obtaining solutions to problems. RL has been successfully applied to various problems such as playing arcade games (Mnih et al., 2013) and controlling vehicles (Yu et al., 2019).

Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2016) is a type of RL algorithm that uses two neural networks (NN) (Rosenblatt, 1958; Ivakhnenko, 1968; Goodfellow et al., 2016) as an agent. The DDPG can be used in an environment where multiple agent actions are needed. Kupwiwat and Yamamoto (Kupwiwat and Yamamoto, 2020) studied various RL algorithms, using NNs as agents, and found that DDPG can be effectively applied to the geometry optimization problem of grid shell structures. However, the agents can only observe a constant number of inputs. Therefore, when structural dimensions are changed, it is difficult for the agents to detect the change of structural characteristics.

Hayashi and Ohsaki (Hayashi and Ohsaki, 2021) proposed a combined method of RL and graph representation for binary topology optimization of planar trusses. Graph representation allows an RL agent to observe the whole structure by transforming the structure into graph data consisting of nodes (vertices) and elements (edges) and implementing repetitive graph embedding operations to transmit signals of adjacent nodes and elements for estimating accumulated rewards associated with each action. Zhu et al. (2021) studied the applicability of RL and graph representation for stochastic topology generation of stable trusses which can be further used as initial structures for other topology optimization algorithms.

Graph neural networks are types of NNs, specifically designed for working with graph data. Graph Convolutional Network (GCN) (Kipf and Welling, 2017) is a class of graph neural networks that uses a convolution operator to process graph signals to the output domain. GCN has been successfully applied to problems such as node classification (Kipf and Welling, 2017) and link prediction (Kipf and Welling, 2016).

This paper proposes methods for the bracing directions optimization of grid shell structures for minimizing the strain energy using DDPG and GCN. The RL agent is trained to optimize the bracing directions from initial randomly generated directions. The proposed method is considered a part of the early-stage design of grid shells. The method takes an input the shape of the grid-shell which must be pre-determined. Bracing direction optimization can reduce the structure's strain energy without affecting the appearance of the shape because the braces are typically covered by finishing or ceiling. This paper is organized as follows: Section 2 gives the optimization formulations. Sections 3, 4 introduce existing approaches of GCN and a type of RL named DDPG, respectively. Section 5 explains the novelty of this research consisting of the vectors and matrices utilized in RL and the formulation of the Markov decision process for training the RL. Numerical examples are presented in Section 6 to benchmark the proposed method against the enumeration method and the genetic algorithm in terms of structural performance and computational cost.

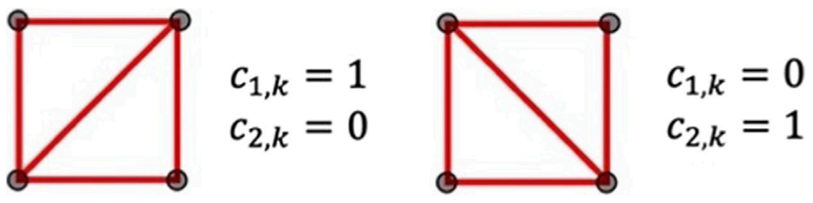

FIGURE 1
Bracing directions in the grid cell k and corresponding values of $c_{1,k}$ and $c_{2,k}$

2 Optimization problems of braced grid shells

2.1 Objective function: Total strain energy

The total strain energy of the structure subjected to static loads is chosen as the objective function to be minimized. The main grid elements are modeled using 3-dimensional beam elements with 12 degrees of freedom (DoFs), whereas the bracing elements are modeled using 3-dimensional truss elements with 6-DoFs. In the local coordinate system, the stiffness matrices of the frame element and the bracing element are denoted as $\mathbf{k}_f \in \mathbb{R}^{12\times12}$ and $\mathbf{k}_e \in \mathbb{R}^{6\times6}$, respectively. These matrices are converted into those with respect to the global coordinate system, and assembled into the global stiffness matrix $\mathbf{K} \in \mathbb{R}^{n_D\times n_D}$, where n_D is the number of DoFs of the structure after assigning the boundary conditions. Every node in the structure is subjected to a point load and every element is subjected to self-weight per unit length. These loads and weights are collected into the load vector $\mathbf{p} \in \mathbb{R}^{n_D}$ with respect to the global coordinates, and the nodal displacement vector $\mathbf{d} \in \mathbb{R}^{n_D}$ is obtained by solving the equilibrium and geometric compatibility equations:

$$\mathbf{Kd} = \mathbf{p} \tag{1}$$

Then, the total strain energy E is computed from

$$E = (1/2)\cdot \mathbf{d}^{\mathrm{T}}\mathbf{Kd} \tag{2}$$

where the superscript T indicates the transpose of a vector or a matrix.

2.2 Bracing direction optimization problem

The combinatorial problem of bracing directions optimization is used to investigate the performance of the proposed method in terms of solution quality and the computational cost. Nodal coordinates and element cross-section size are not included in the variables. Given a grid shell with n_x by n_y square grids and diagonal bracing in each grid cell $k \in \{1,\ldots,n_x n_y\}$, there can be two possible directions for bracing indicated by $c_{1,k}, c_{2,k} \in \{0,1\}$ which correspond to absence and presence of the brace in each direction as illustrated in Figure 1.

Since only one brace should exist in the grid cell k, the summation $c_{1,k} + c_{2,k}$ is always 1. Let $\mathbf{c}_1 \in \mathbb{R}^{n_x n_y}$ and $\mathbf{c}_2 \in \mathbb{R}^{n_x n_y}$, respectively, be vectors consisting of $c_{1,k}$ and $c_{2,k}$ for all bracing elements. The global stiffness matrix of the structure is a function of $\mathbf{c}_1$ and $\mathbf{c}_2$ denoted as $\mathbf{K}(\mathbf{c}_1,\mathbf{c}_2)$. The bracing direction optimization problem to minimize the strain energy is formulated as follows:

$$\text{minimize} \quad E(\mathbf{c}_1,\mathbf{c}_2) = (1/2)\cdot \mathbf{d}^{\mathrm{T}}\mathbf{K}(\mathbf{c}_1,\mathbf{c}_2)\mathbf{d} \tag{3a}$$

$$\text{subject to} \quad c_{1,k},\ c_{2,k} \in \{0,1\},\ \left(k = 1,\ 2,\ \ldots, n_x n_y\right) \tag{3b}$$

$$c_{1,k} + c_{2,k} = 1,\ \left(k = 1,\ 2,\ \ldots, n_x n_y\right) \tag{3c}$$

where $\mathbf{d}$ is an implicit function of $\mathbf{c}_1$ and $\mathbf{c}_2$ obtained by solving Eq. 1.

3 Graph Convolutional Network

Consider a graph consisting of n nodes with g features in each node. The graph data can be represented using a node feature matrix $\mathbf{N} \in \mathbb{R}^{n\times g}$ representing features of each node in the graph, an adjacency matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$ representing the connectivity of structural elements, a weighted adjacency matrix $\mathbf{P} \in \mathbb{R}^{n\times n}$ representing the connectivity of structural elements weighted by element forces, and a degree matrix $\mathbf{D} \in \mathbb{R}^{n\times n}$ representing the number of connections of each node in the graph. Kipf and Welling (Kipf and Welling, 2017) proposed a GCN that can process graph input data which are chosen from $\{\mathbf{N},\mathbf{M},\mathbf{P},\mathbf{D}\}$, and map the inputs to the target domain of the graph for tasks such as node classification or link (connection) prediction (i.e., perform supervise learning by comparing mapped target domain from GCN to the node classification training data). A single GCN computation can be considered as a *layer*. Multiple GCN *layers* can be connected together to create a computational model for an RL agent. A GCN *layer* consists of a normalized form of adjacency matrix $\tilde{\mathbf{M}}$, and converts the input instances $\mathbf{N}$ to the output $\mathbf{O} \in \mathbb{R}^{n\times h}$ that has h embedding spaces (i.e., output for the GCN

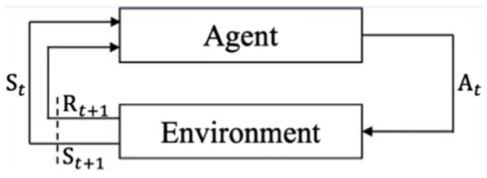

FIGURE 2
Diagram of the MDP

or output for a GCN layer that will be treated as **N** for the next GCN layer). The GCN layer can be formulated as follows:

$$\mathbf{O} = \sigma\left(\tilde{\mathbf{M}}\mathbf{N}\mathbf{w}\right) \quad (4)$$

where σ is a non-linear activation function, and $\mathbf{w} \in \mathbb{R}^{g \times h}$ is the weight matrix (i.e., the convolution filter parameters) (Kipf and Welling, 2017) in the GCN layer which is adjusted during the training. The convolutional filter parameters **w** weight $\tilde{\mathbf{M}}\mathbf{N}$, an aggregated signal of a node and its neighboring nodes. It should be noted that **w** can handle any graph with any number of nodes as long as the nodes have the same number of features.

The normalized adjacency matrix $\tilde{\mathbf{M}}$ can be computed using the following equation:

$$\tilde{\mathbf{M}} = \mathbf{D}^{-1/2}[\mathbf{M} + \mathbf{I}]\mathbf{D}^{-1/2} \quad (5)$$

where $\mathbf{I} \in \mathbb{R}^{n \times n}$ is the identity matrix, and $\mathbf{D}^{-1/2}$ is the inverse of the matrix $\mathbf{D}^{1/2}$ satisfying. $\mathbf{D}^{1/2}\mathbf{D}^{1/2} = \mathbf{D}$

In this paper, we utilize GCN to build an RL agent. The original GCN does not utilize the weighted adjacency matrix. However, for the structural optimization problem, the internal forces (i.e., the forces taken by the structural elements) should be utilized to guide the actions of the RL agent. Therefore, we propose a novel GCN-DDPG architecture that employs weighted adjacency matrices constructed from the internal forces for solving the bracing direction optimization problem. Details of the formulation are given in Section 5.

4 Reinforcement learning and Deep Deterministic Policy Gradient

RL is a type of ML that trains an agent to perform actions in an environment using reward signals. An RL algorithm consists of three main elements: a *policy* that determines the agent behavior, a *reward signal* that defines how good/bad the agent behavior is, according to the policy, and a *value function* that predicts how the agent performs based on the policy (Sutton and Andrew, 2018). The interaction of an agent and the environment is formulated using a Markov Decision Process (MDP) (Bellman, 1954; Bellman, 1957) as follows:

In a discrete step t:
The agent receives a representation of the environment as state S_t.
The agent performs actions A_t.
The agent receives quantitative reward R_{t+1} and next state S_{t+1} from the environment.

The diagram of the MDP can be represented as shown in Figure 2.

DDPG (Lillicrap et al., 2016) is a type of RL policy gradient algorithm that utilizes a parameterized policy function (*Actor*) π_{θ_1} to determine the probability of taking action A_t^i in a state S_t, denoted by $P(A_t^i|S_t)$, and another parameterized value function Q_{θ_2} (*Critic*) to predict the accumulated reward (Q-Value) from the actions of the agent follows:

$$\boldsymbol{\pi}_{\theta_1}(S_t) = P(A_t^i|S_t) \quad (6)$$

$$Q_{\theta_2}(S_t, \boldsymbol{\pi}_{\theta_1}(S_t)) = \sum_{v=1}^{\infty} \gamma^{v-1} R_{t+v} \quad (7)$$

where θ_1 and θ_2 are parameters of policy and value functions to be adjusted during the training, respectively. $\gamma \in [0, 1)$ is a discount factor for the reward.

During training, the value function adjusts its parameter θ_2 to increase the accuracy of its prediction of accumulated reward using a *replay buffer* that stores data of $\{S_t, A_t, R_{t+1}, S_{t+1}\}$, whereas the policy function adjusts its parameter θ_1 to increase the value predicted by the value function, which is equivalent to the obtained rewards, using the gradient ∇J_{θ_1} as described in the following DDPG algorithm. Since using the online policy and value functions will make the learning unstable, Haarnoja et al. (Haarnoja et al., 2018) proposed a *tau update* method that trains a surrogate policy function $\boldsymbol{\pi}'_{\theta'_1}$ and a surrogate value function $\mathbf{Q}'_{\theta'_2}$, and then gradually updates the parameters of these functions into the online functions using a small value of τ $(\tau \ll 1)$ at every *tau update* interval. Note that the agent interacts with the environment to collect data for the replay buffer using the online policy function.

Let $\mathcal{L}(y, \hat{y})$ be a loss function between y which represents the correct value of training data and a predicted value $\hat{y}$. The training algorithm of DDPG is as follows:

DDPG algorithm:
1. Sample u training data $\{S_t, A_t, R_{t+1}, S_{t+1}\}$ from the replay buffer and convert them into a set of vectors $\{\mathbf{S}_t, \mathbf{A}_t, \mathbf{R}_{t+1}, \mathbf{S}_{t+1}\}$.
2. Update the parameters as follows:

$\boldsymbol{\pi}'_{\theta'_1}(\mathbf{S}_t) = \hat{\mathbf{A}}_t$ *# Surrogate policy function* $\boldsymbol{\pi}'_{\theta'_1}$ *decides actions from* $\mathbf{S}_t$

$\boldsymbol{\pi}_{\theta_1}(\mathbf{S}_{t+1}) = \hat{\mathbf{A}}_{t+1}$ *# Online policy function* $\boldsymbol{\pi}_{\theta_1}$ *decides actions from* $\mathbf{S}_{t+1}$

$Q'_{\theta'_2}(\mathbf{S}_t, \mathbf{A}_t) = \hat{\mathbf{Q}}_t$ *# Surrogate value function* $Q'_{\theta'_2}$ *predicts reward from* $\mathbf{S}_t$, $\mathbf{A}_t$

$Q_{\theta_2}(\mathbf{S}_{t+1}, \hat{\mathbf{A}}_{t+1}) = \mathbf{Q}_{t+1}$ *# Online value function* Q_{θ_2} *predicts reward from* $\mathbf{S}_{t+1}$, $\hat{\mathbf{A}}_{t+1}$

```
    $\nabla Q'_{\theta'_2} = \nabla_{\theta'_2} Q'_{\theta'_2}(\mathbf{S}_t, \mathbf{A}_t) \nabla_{\hat{\mathbf{Q}}_t} \mathcal{L}(\mathbf{R}_{t+1} + \mathbf{Q}_{t+1}, \hat{\mathbf{Q}}_t)$ # Gradient of $Q'_{\theta'_2}$ to the loss function
    $\nabla J'_{\theta'_1} = -\mathbb{E}[\nabla_{\theta'_1} \boldsymbol{\pi}'_{\theta'_1}(\mathbf{S}_t) \nabla_{\hat{\mathbf{A}}_t} Q'_{\theta'_2}(\mathbf{S}_t, \hat{\mathbf{A}}_t)|_{\hat{\mathbf{A}}_t = \boldsymbol{\pi}'_{\theta'_1}(\mathbf{S}_t)}]$ # Gradient of $\boldsymbol{\pi}'_{\theta'_1}$ to the Q-Value.
    Update $\boldsymbol{\theta}'_2$ in $Q'_{\theta'_2}$ using $\nabla Q'_{\theta'_2}$
    Update $\boldsymbol{\theta}'_1$ in $\boldsymbol{\pi}'_{\theta'_1}$ using $\nabla J'_{\theta'_1}$
    If tau update interval is reached:
        $\boldsymbol{\theta}_1 = (1-\tau)\boldsymbol{\theta}_1 + \tau\boldsymbol{\theta}'_1$ # Update parameters of $\boldsymbol{\pi}_{\theta_1}$
        $\boldsymbol{\theta}_2 = (1-\tau)\boldsymbol{\theta}_2 + \tau\boldsymbol{\theta}'_2$ # Update parameters of $Q_{\theta_2}$
```

In order to reduce training time of adjusting the weights in each layer of GCN using the gradients, optimizers such as stochastic gradient descent (SGD) (Robbins and Monro, 1951; Kiefer and Wolfowitz, 1952; Ruder, 2016) or Adam (Kingma and Ba, 2015), is used for updating $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$. The exploration of DDPG's policy is activated by adding small Ornstein-Uhlenbeck noise (Uhlenbeck and Ornstein, 1930) into the output value of the policy function.

5 Reinforcement learning for structural optimization

5.1 State

This research utilizes the graph representation to express structural data, such as nodal coordinates, boundary conditions, and internal forces, at each optimization step which is equivalent to a step t in the MDP formulation.

Suppose we have five node features, and the ith row of the node feature matrix $\mathbf{N} \in \mathbb{R}^{n \times 5}$ is represented as $\boldsymbol{n}_i = \{x_i / \max_p x_p \quad y_i / \max_p y_p \quad z_i / z_{\max} \quad k^i_{\text{free}} \quad k^i_{\text{fix}}\}$, in which x_i, y_i, and z_i are the coordinates of node i, $z_{\max}$ is the pre-determined upper-bound value of z_i, $\max_p x_p$ and $\max_p y_p$ are the maximum coordinate values in each axis. Note that the minimum coordinate values are assumed to be 0 for all coordinates. k^i_{free} and k^i_{fix} are determined depending on the boundary condition. $(k^i_{\text{free}}, k^i_{\text{fix}}) = (0, 1)$ if node i is fixed support, and $(k^i_{\text{free}}, k^i_{\text{fix}}) = (1, 0)$ if node i is not supported.

In the adjacency matrix for frame elements $\mathbf{M}_1 \in \mathbb{R}^{n \times n}$, the existence of a frame element e connecting nodes i and j is denoted as $m_{1ij} = m_{1ji} = k^e_{\text{frame}}$ where $m_{1_{ij}}$ indicates (i, j) component of matrix $\mathbf{M}_1$. k^e_{frame} indicates the existence and non-existence of a 12-DoFs frame element e that connects nodes i and j by $k^e_{\text{frame}} = 1$ and 0, respectively. In the adjacency matrix for truss elements $\mathbf{M}_2 \in \mathbb{R}^{n \times n}$, the existence of a truss element e connecting nodes i and j is represented as $m_{2ij} = m_{2ji} = k^e_{\text{truss}}$. k^e_{truss} indicates the existence and non-existence of a 6-DoFs truss element e that connects nodes i and j by $k^e_{\text{truss}} = 1$ and 0, respectively. The combined adjacency matrix for frame and truss elements $\mathbf{M}_3 \in \mathbb{R}^{n \times n}$ is obtained by $\mathbf{M}_3 = \mathbf{M}_1 + \mathbf{M}_2$.

In order to evaluate the efficiency of the structural configuration, this paper proposes weighted adjacency matrices to represent the element internal forces. For the frame element, only a single weighted adjacency matrix $\mathbf{P}_1 \in \mathbb{R}^{n \times n}$ is determined using the ratio between the bending moment and the axial force, which for this type of structure is a useful index that helps minimizing the strain energy. Suppose frame element e connects nodes i and j, the entry p_{1ij} in $\mathbf{P}_1$ is determined as follows:

$$p_{1_{ij}} = k^e_{\text{frame}} b'_{ei} \Big/ \left(a'_e + 1\right) \tag{8a}$$

$$b'_{ei} = \left(|b_{ei}| - b^{\max}_{\text{f}}\right) \Big/ \left(b^{\max}_{\text{f}} - b^{\min}_{\text{f}}\right) \tag{8b}$$

$$a'_e = \left(|a_e| - a^{\max}_{\text{f}}\right) \Big/ \left(a^{\max}_{\text{f}} - a^{\min}_{\text{f}}\right) \tag{8c}$$

where b_{ei} is the bending moment around the horizontal axis on the section at node i, and a_e is the axial force of frame element e. $b^{\max}_{\text{f}}$ and $b^{\min}_{\text{f}}$ are the maximum and minimum absolute values of bending moments at the element ends. $a^{\max}_{\text{f}}$ and $a^{\min}_{\text{f}}$ are the maximum and minimum absolute axial forces of frame elements. For the truss elements, weighted adjacency matrix ($\mathbf{P}_2 \in \mathbb{R}^{n \times n}$) is a normalized form of the truss axial force. The entry $p_{2_{ij}}$ in $\mathbf{P}_2$ corresponding to element e connecting nodes i and j is determined as follows:

$$p_{2_{ij}} = k^e_{\text{truss}} a'_e \tag{9a}$$

$$a'_e = \left(|a_e| - a^{\max}_{\text{q}}\right) \Big/ \left(a^{\max}_{\text{q}} - a^{\min}_{\text{q}}\right) \tag{9b}$$

where a_e is the axial force in the truss element e. $a^{\max}_{\text{q}}$ and $a^{\min}_{\text{q}}$ are the maximum and minimum absolute values of axial forces for truss elements, respectively. All values in these weighted adjacency matrices are in the range of $[0, 1]$, which helps avoiding numerical instabilities during training.

The degree matrices for frame, truss, and combined frame and truss elements are denoted by $\mathbf{D}_1 \in \mathbb{R}^{n \times n}$, $\mathbf{D}_2 \in \mathbb{R}^{n \times n}$, and $\mathbf{D}_3 \in \mathbb{R}^{n \times n}$, respectively. Entries in each degree matrix $\mathbf{D}_u$ $(u = 1, 2, 3)$ are computed using the associated adjacency matrix $\mathbf{M}_u$ $(u = 1, 2, 3)$ as $d_{u_{ij}} = \delta_{ij} \times \sum_{i=1}^{n} m_{u_{ij}}$, where δ_{ij} is the Kronecker delta which is 0 if $i \neq j$ and 1 if $i = j$. The normalized adjacency matrices of frame, truss, and combined frame and truss elements are $\tilde{\mathbf{M}}_1 \in \mathbb{R}^{n \times n}$, $\tilde{\mathbf{M}}_2 \in \mathbb{R}^{n \times n}$, and $\tilde{\mathbf{M}}_3 \in \mathbb{R}^{n \times n}$, respectively, which are computed using Eq. 5. The GCN-DDPG agent for bracing direction optimization uses $\tilde{\mathbf{M}}_1$, $\tilde{\mathbf{M}}_2$, $\tilde{\mathbf{M}}_3$, $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{M}_2$, and $\mathbf{N}$ as a representation of the environment S_t which will be further explained in Section 5.2.

5.2 GCN-DDPG agent

Policy and value functions (i.e., Actor and Critic networks of a GCN-DDPG agent) consist of multiple GCN layers. The policy function takes state data described in Section 5.1 as input to compute the output denoted as $\mathbf{O} \in \mathbb{R}^{n \times h}$ which has the same number of rows as those of the node feature matrix $\mathbf{N} \in \mathbb{R}^{n \times g}$.

This output is used to determine bracing directions, explained in Section 5.3.

The inputs of the value function are state data and the output from the policy function to compute an estimation of the accumulated reward. In the value function, another matrix for prediction of the bracing directions denoted as $\mathbf{M}_o \in \mathbb{R}^{n \times n}$ is internally computed from the output of the policy function $\mathbf{O} \in \mathbb{R}^{n \times h}$, multiplied element-wise with $\mathbf{M}_2$ to exclude non-bracing elements, and normalized within the range [0,1] through dividing by h as $\mathbf{M}_o = (\mathbf{O} \cdot \mathbf{O}^T) \odot \mathbf{M}_2/h$, where $\odot$ is the element-wise multiplication.

When multiple GCN layers are connected, a node feature matrix is replaced with an output from the prior GCN layer. To represent internal forces in the structure, a normalized adjacency matrix in a GCN layer can be replaced with a weighted adjacency matrix. The output of a GCN layer is transformed by the Rectified Linear Unit (ReLU) activation function (Nair and Hinton, 2010) similar to the original GCN (Kipf and Welling, 2017) for all layers of both policy and value functions. ReLU is a nearly linear function that is computationally efficient for gradient-based optimization (i.e., SGD or Adam) (Chigozie et al., 2020). ReLU is applied to all layers of both policy and value functions except the last layer of the policy function which is processed by the Sigmoid activation function for predicting probability-based output (i.e., probability of taking action A_t^i in a state S_t) (Chigozie et al., 2020). Eq. 10a and Eq. 10b represent GCN layers with ReLU and Sigmoid activation functions, respectively, where $\mathbf{N}' \in \mathbb{R}^{n \times g}$ denotes a node feature matrix or an output from a prior GCN layer and $\mathbf{M}' \in \mathbb{R}^{n \times n}$ denotes $\tilde{\mathbf{M}}_1, \tilde{\mathbf{M}}_2, \tilde{\mathbf{M}}_3, \mathbf{P}_1, \mathbf{P}_2$, or $\mathbf{M}_o$ with size $\mathbb{R}^{n \times n}$. Eq. 10c represents multiple computing loops using the same GCN layer.

$$\mu\left(\mathbf{N}', \mathbf{M}'\right) = \mathrm{ReLU}\left(\mathbf{N}'\mathbf{M}'\mathbf{w}_\mu\right) \tag{10a}$$

$$\sigma\left(\mathbf{N}', \mathbf{M}'\right) = \mathrm{Sigmoid}\left(\mathbf{N}'\mathbf{M}'\mathbf{w}_\sigma\right) \tag{10b}$$

$$\mathrm{iter}^{\phi}\left[\mu\left(\mathbf{N}', \mathbf{M}'\right)\right] = \underbrace{\mu\left(\mathbf{M}'\left(\mu\left(\mathbf{M}'\left(\ldots\right)\mathbf{w}_\mu\right)\right)\mathbf{w}_\mu\right)}_{\phi \text{times}} \tag{10c}$$

where $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$, $\mathrm{Sigmoid}(\cdot) = 1/(1 + e^{-(\cdot)})$, and ϕ in $\mathrm{iter}^{\phi}[\,]$ indicates the number of computing loops.

Since the output of a GCN layer is a matrix but the value function output is a scalar representing the estimation of accumulated reward, two operations are used for transforming the output matrix of the last GCN layer of the value function into a scalar value. The first operation is a global sum pooling operation (GSP) (Aich and Stavness, 2018) which transforms the output matrix into a vector by summing up all entries in each column of the output matrix. Let $\mathbf{V} \in \mathbb{R}^{n \times g}$ be a matrix, the GSP operation to transform $\mathbf{V}$ into a vector can be represented as

$$\mathrm{Pool}(\mathbf{V}) = \left[\sum_{i=1}^{n} v_{i,1} \quad \cdots \quad \sum_{i=1}^{n} v_{i,g}\right] \in \mathbb{R}^{1 \times g} \tag{11}$$

TABLE 1 Policy and value functions of GCN-DDPG for bracing direction optimization.

Policy function π	**Value function Q**
Inputs: $\tilde{\mathbf{M}}_1$, $\tilde{\mathbf{M}}_2$, $\tilde{\mathbf{M}}_3$, $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{N}$	Inputs: $\tilde{\mathbf{M}}_1$, $\tilde{\mathbf{M}}_2$, $\tilde{\mathbf{M}}_3$, $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{N}$, $\mathbf{M}_2$, $\mathbf{O}$
Computation:	Computation:
Step 1: $\mathbf{N}_{1.1} = \mu(\mu(\mathbf{N}, \mathbf{P}_1), \tilde{\mathbf{M}}_1)$ $\mathbf{N}_{2.1} = \mathrm{iter}^2[\mu(\mathbf{N}_{1.1}, \mathbf{P}_1)]$	Step 1: $\mathbf{N}_{1.1} = \mu(\mu(\mathbf{N}, \mathbf{P}_1), \tilde{\mathbf{M}}_1)$ $\mathbf{N}_{2.1} = \mathrm{iter}^2[\mu(\mathbf{N}_{1.1}, \mathbf{P}_1)]$
Step 2: $\mathbf{N}_{1.2} = \mu(\mu(\mathbf{N}, \mathbf{P}_2), \tilde{\mathbf{M}}_2)$ $\mathbf{N}_{2.2} = \mathrm{iter}^2[\mu(\mathbf{N}_{1.2}, \mathbf{P}_2)]$	Step 2: $\mathbf{N}_{1.2} = \mu(\mu(\mathbf{N}, \mathbf{P}_2), \tilde{\mathbf{M}}_2)$ $\mathbf{N}_{2.2} = \mathrm{iter}^2[\mu(\mathbf{N}_{1.2}, \mathbf{P}_2)]$
Step 3: $\mathbf{N}_3 = \mathbf{N}_{2.1} + \mathbf{N}_{2.2}$ $\mathbf{O} = \sigma(\mathbf{N}_3, \tilde{\mathbf{M}}_3)$	Step 3: $\mathbf{M}_o = (\mathbf{O} \cdot \mathbf{O}^T) \odot \mathbf{M}_2/h$
	Step 4: $\mathbf{N}_{1.3} = \mu(\mu(\mathbf{O}, \mathbf{M}_o), \tilde{\mathbf{M}}_2)$ $\mathbf{N}_{2.3} = \mathrm{iter}^2[\mu(\mu(\mathbf{N}_{1.3}, \mathbf{M}_o), \tilde{\mathbf{M}}_2)]$
	Step 5: $\mathbf{N}_3 = \mathbf{N}_{2.1} + \mathbf{N}_{2.2} + \mathbf{N}_{2.3}$ $\mathbf{N}_4 = \mu(\mathbf{N}_3, \tilde{\mathbf{M}}_3)$
	Step 6: $\mathbf{N}_5 = \mathrm{Pool}(\mathbf{N}_4)$
	Step 7: $Q = f_{NN}(\mathbf{N}_5)$
Output: $\mathbf{O} \in \mathbb{R}^{n \times h}$	Output: $Q \in \mathbb{R}^{1 \times 1}$

The second operation is to compute the estimation of accumulated reward (i.e., Q-value $\in \mathbb{R}^{1 \times 1}$) from the vector output of the GSP operation using a neural network which consists of approximation functions that have adjustable weight parameters and activation functions (Goodfellow et al., 2016). These approximation functions are connected together so that the output of the prior approximation function is the input of the next approximation function, similarly to how the GCN layers are connected, where every approximation function, except the last one, is called a hidden layer (Goodfellow et al., 2016). Eq. 12 represents an NN with two hidden layers, used in this work, for computing the estimation of accumulated reward from the vector output of the GSP operation $\mathbf{H} \in \mathbb{R}^{1 \times g}$ with adjustable internal weight matrices $\mathcal{W}_1$, $\mathcal{W}_2$, $\mathcal{W}_{out}$, adjustable internal bias vectors $\mathcal{B}_1$ and $\mathcal{B}_2$, and adjustable internal bias scalar $\mathcal{B}_{out}$ as

$$f_{NN}(\mathbf{H}) = \mathcal{W}_{out}\left(\mathrm{ReLU}\left(\mathcal{W}_2\left(\mathrm{ReLU}\left(\mathcal{W}_1\mathbf{H}^T + \mathcal{B}_1\right)\right)^T + \mathcal{B}_2\right)\right)^T + \mathcal{B}_{out} \in \mathbb{R}^{1 \times 1} \tag{12}$$

Table 1 summarizes the computation processes of the policy and value functions of the GCN-DDPG agent for bracing direction optimization. In each column, the 1st row indicates if the computation belongs to the policy or the value function. The 2nd row denotes input data used for the computation. The 3rd row indicates the computation process using GCN layers, GSP operation, and NN in Eqs 10a–12a–Eqs 10a–12.

In the policy function, inputs from state data of frame and truss elements are separately processed in Steps 1 and 2, respectively. Output matrices in Steps 1 and 2 are combined to compute the output of the policy function in Step 3. In the value function, inputs from state data of frame and truss are also processed separately in Steps 1 and 2. Step3 computes the matrix for element direction prediction $\mathbf{M}_o$ which is used for processing

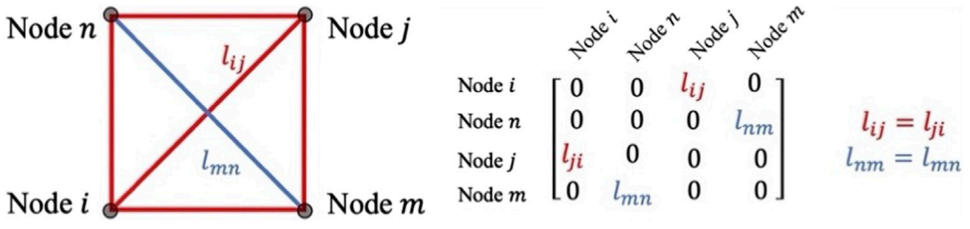
Node n
Node j
Node i
Node m
l_{ij}
l_{mn}
$$\begin{array}{c|cccc} & \text{Node } i & \text{Node } n & \text{Node } j & \text{Node } m \\ \hline \text{Node } i & 0 & 0 & l_{ij} & 0 \\ \text{Node } n & 0 & 0 & 0 & l_{nm} \\ \text{Node } j & l_{ji} & 0 & 0 & 0 \\ \text{Node } m & 0 & l_{mn} & 0 & 0 \end{array}$$
$l_{ij} = l_{ji}$
$l_{nm} = l_{mn}$

FIGURE 3
Bracing directions and associate link predictions in a grid cell.

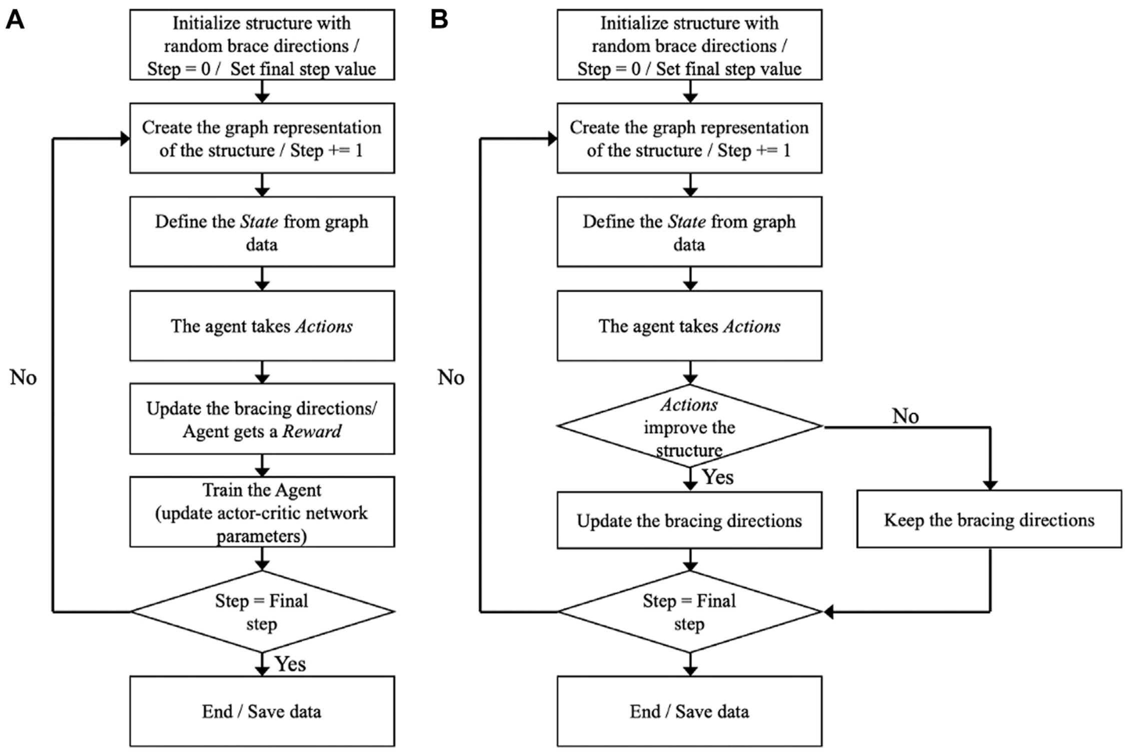
A
Initialize structure with random brace directions / Step = 0 / Set final step value
Create the graph representation of the structure / Step += 1
Define the State from graph data
The agent takes Actions
Update the bracing directions/ Agent gets a Reward
Train the Agent (update actor-critic network parameters)
Step = Final step
No
Yes
End / Save data
B
Initialize structure with random brace directions / Step = 0 / Set final step value
Create the graph representation of the structure / Step += 1
Define the State from graph data
The agent takes Actions
Actions improve the structure
No
Yes
Update the bracing directions
Keep the bracing directions
Step = Final step
No
End / Save data

FIGURE 4
Algorithms of the proposed method; **(A)** Training phase, **(B)** Test phase.

action data in Step 4. Output matrices from the Steps 1, 2, and 4 are combined and re-processed by another GCN layer in Step 5. In Steps 6 and 7, the output matrix is converted into a vector using the GSP operation and the vector is finally converted to the Q-value by the NN. The last row denotes the output of the functions.

5.3 Action

The bracing direction in each grid cell is determined from a dot product of each row of $\mathbf{O} \in \mathbb{R}^{n \times h}$ from the policy function output, similarly to the link prediction using GCN (Kipf and Welling, 2016) for the existence of a connection between two nodes in the graph. In this paper, the structural nodes are represented as graph nodes. Therefore, the prediction of a link or connection between two structural nodes is equivalent to a structural element (bracing element) that connects those nodes. Figure 3 shows a 4-node structure in a grid cell with two possible diagonal braces which connect node i to node j and node n to node m, respectively. The dot product of each output matrix is used to predict the value of l_{ij} and l_{nm}. l_{ij} is equivalent to l_{ji} and l_{nm} is also equivalent to l_{mn} as shown in the matrix in

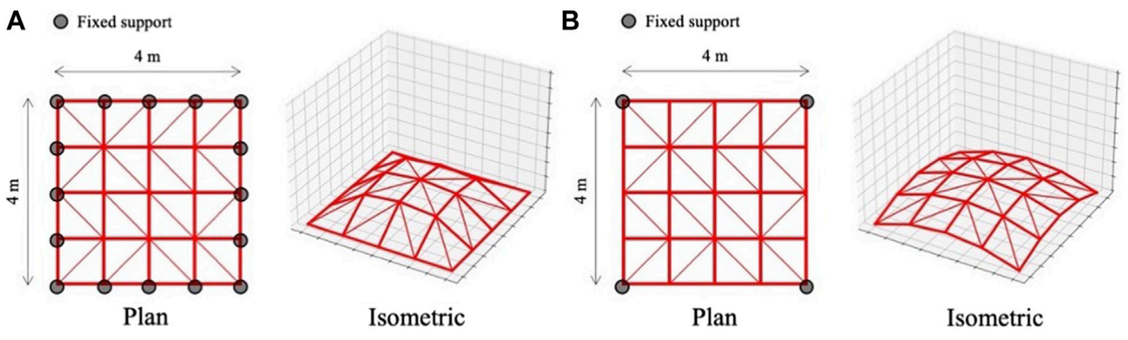

FIGURE 5
Structural models for bracing direction optimization during training phase; **(A)** Support condition 1, **(B)** Support condition 2.

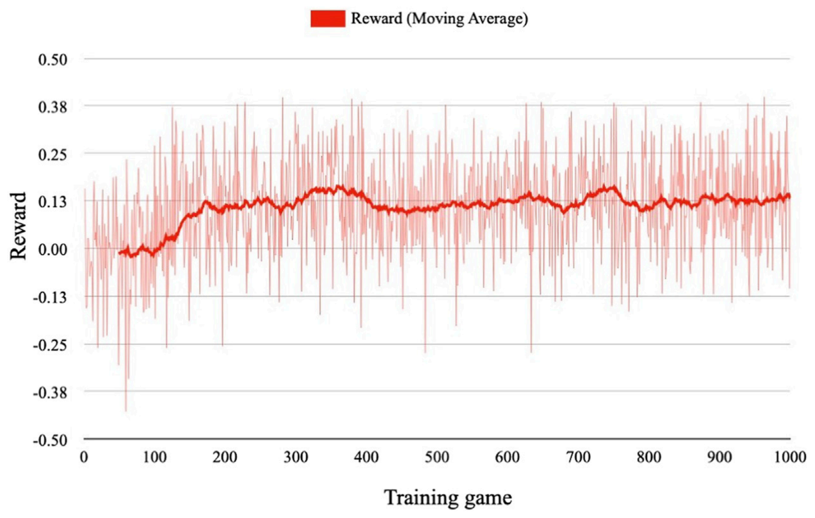

FIGURE 6
Variation of reward and its moving average in training phase.

Figure 3. The bracing direction optimization formulation in Eqs 3a–ca–cEqs 3a–c allows only one brace in each grid cell. Therefore, l_{ij} and l_{nm} are compared to determine the bracing direction in a grid cell.

The number of nodal embedding output dimensions is 100; i.e., $h = 100$. From the output of the policy function $\mathbf{O} \in \mathbb{R}^{n\times 100}$, the brace in grid cell k is determined as

$$\mathrm{A}_t^k = \begin{cases} (c_{1,k},\ c_{2,k}) = (1,\ 0) & (\text{if } l_{ij} > l_{mn}) \\ (c_{1,k},\ c_{2,k}) = (0,\ 1) & (\text{if } l_{ij} < l_{mn}) \end{cases} \tag{13a}$$

$$l_{ij} = l_{ji} = \sum_{u=1}^{100} o_{i,u} o_{j,u} \tag{13b}$$

$$l_{mn} = l_{nm} = \sum_{u=1}^{100} o_{n,u} o_{m,u} \tag{13c}$$

At each step, the agent can change any number of brace directions.

5.4 Reward

In RL, a reward signal is used for training an agent. The reward signal R_{t+1} that the agent receives after executing action A_t in a state S_t at step t is formulated from the change of the strain energy as follows:

$$\mathrm{R}_{t+1} = (E_t - E_{t+1})/E_0 \tag{14}$$

where E_t and E_0 are the strain energy of the structure at step t and the strain energy of the initial structure.

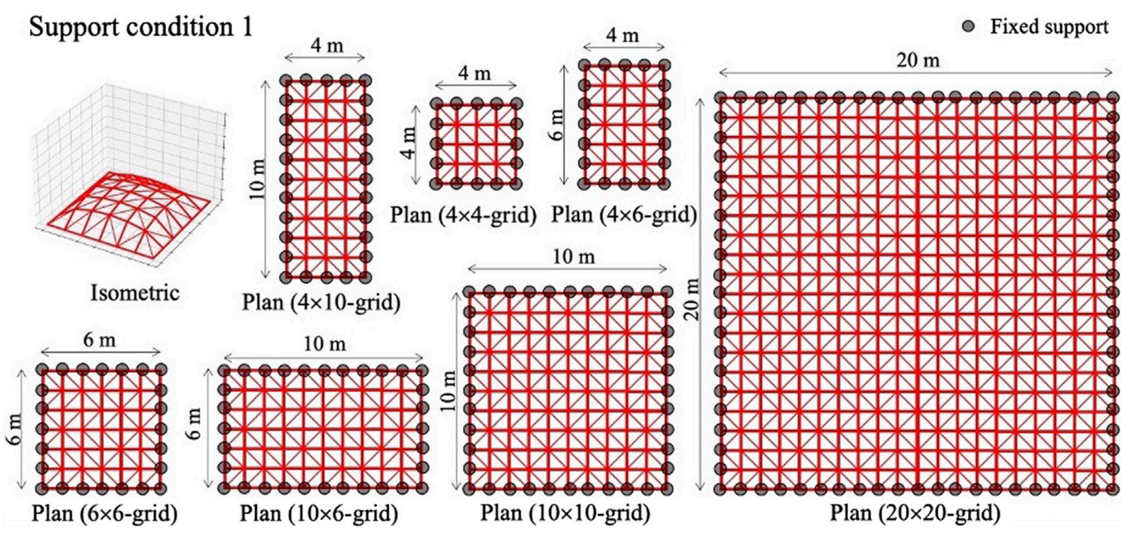

FIGURE 7
Structural models for test phase: Support condition 1.

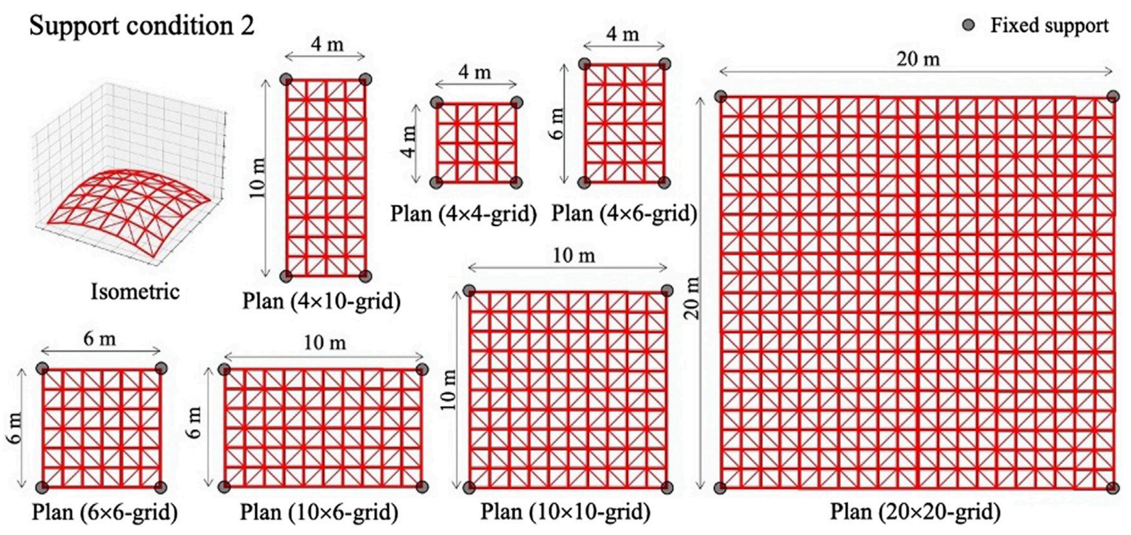

FIGURE 8
Structural models for test phase: Support condition 2.

6 Numerical examples

6.1 General experiment setting and structural model

The agent is trained to optimize the structure in the *training phase* during which the ability to improve performance is assessed. In the *test phase*, the agent performance is evaluated on structural configurations other than those used in training. The method is implemented using Python 3.6 environment. A PC with CPU Intel Core i5-6600 (3.3 GHz, 4 cores) and GPU AMD Radeon R9 M395 2 GB is employed for computation.

Training is carried out on a grid shell structure with 4 × 4 grids and diagonal truss braces. Each grid cell has dimensions of 1.0 m by 1.0 m. To simplify the problem, the 12-DoFs frame element has a hollow cylindrical section with an external diameter of 100 mm and an internal diameter of 90 mm. The 6-DoFs truss element has a solid circular section with a diameter of 43.6 mm. Both elements have Young's modulus of 205 kN/

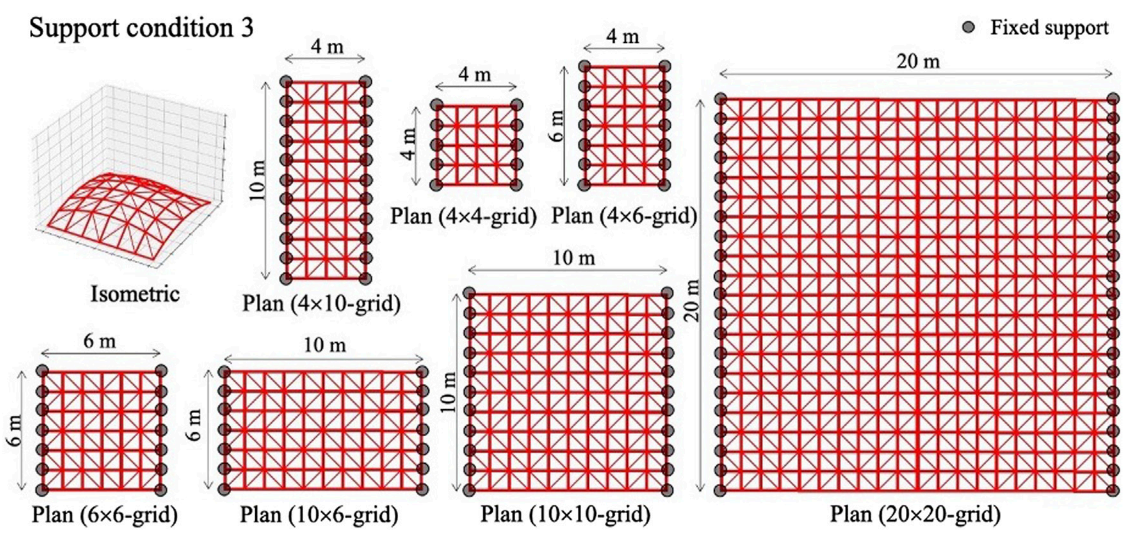

FIGURE 9
Structural models for test phase: Support condition 3.

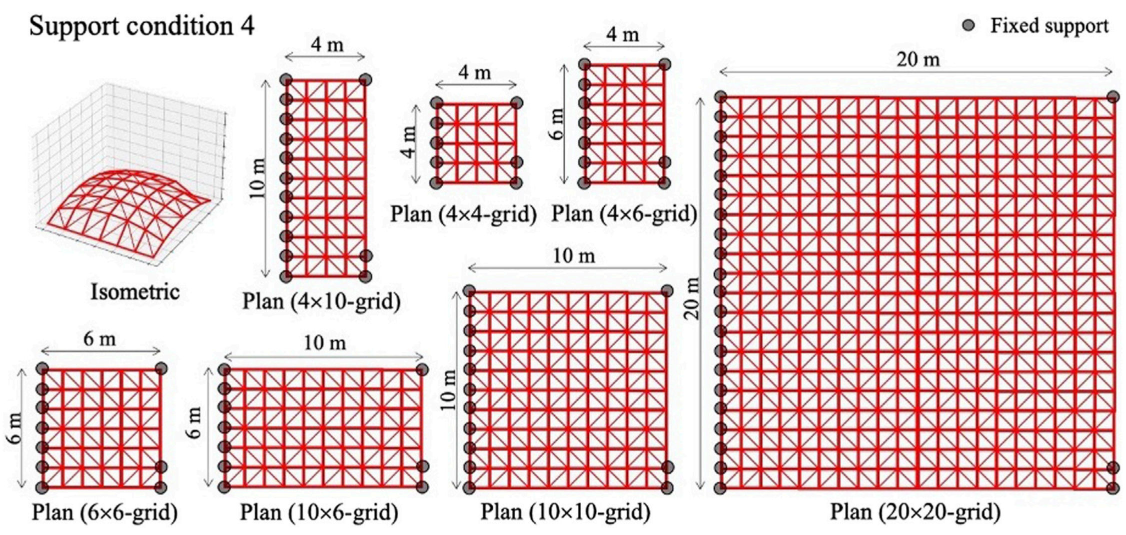

FIGURE 10
Structural models for test phase: Support condition 4.

mm^2 and a similar weight of 12 kg/m. All structural nodes are subjected to a vertical point load of 10 kN.

Results obtained in the test phase using the proposed method are compared with those of the enumeration method (EM) and the genetic algorithm (GA); EM is used for the benchmark when it is feasible to compute all possible solutions, and GA is used for the benchmark when computing all possible solution is not feasible due to the large search space.

The algorithm flowcharts for training and test phases are given in Figure 4. During the optimization, each MDP is denoted as a step. The loop of MDPs or a *game* is terminated when the final step is reached.

TABLE 2 Test results.

Support conditions	Grid size	Min. (N·m)	Mean (N·m)	Std. (N·m)	Reduction (%)
1	4 × 4	4.38	4.53	0.19	16.83
	4 × 6	15.71	16.01	0.38	13.21
	6 × 6	63.15	67.40	2.56	18.10
	4 × 8	72.49	73.72	0.69	5.55
	10 × 6	450.47	459.92	7.89	12.89
	10 × 10	2,242.26	2,288.05	40.79	19.80
	20 × 20	213,715.82	217,120.90	1742.84	10.08
2	4 × 4	77.88	89.35	10.02	19.94
	4 × 6	284.81	290.09	3.98	20.21
	6 × 6	818.14	896.06	46.93	19.71
	4 × 10	2,124.56	2,142.64	17.10	20.76
	10 × 6	4,642.28	4,713.70	37.63	19.79
	10 × 10	17,683.39	17,966.80	181.50	24.36
	20 × 20	1,275,569.06	1,312,975.69	42,317.86	19.66
3	4 × 4	32.11	32.78	0.53	6.43
	4 × 6	70.09	72.40	1.44	5.29
	6 × 6	345.73	356.54	6.92	12.59
	4 × 10	193.91	198.18	2.84	6.69
	10 × 6	2,841.17	2,869.52	18.50	10.95
	10 × 10	8,219.87	8,333.38	101.09	12.28
	20 × 20	589,997.91	596,066.09	4,460.72	9.38
4	4 × 4	40.81	41.34	0.46	17.71
	4 × 6	156.09	161.09	3.61	22.68
	6 × 6	527.56	535.43	5.95	17.30
	4 × 10	1,192.81	1,213.16	11.94	21.76
	10 × 6	3,466.03	3,542.59	47.95	11.63
	10 × 10	12,629.96	12,762.32	109.98	16.49
	20 × 20	920,936.67	931,040.36	8,426.17	16.35

6.2 Training phase

In each game of training, a dome-shaped structural model is initialized with supports assigned to two pre-determined structural shapes indicated in Figure 5. The maximum and minimum nodal height is 1 and 0 m, respectively. Note that structural shapes and size of structural elements are not adjusted during the optimization process. The brace directions are randomly initialized at every game.

The final step in each game is 200. At each step, the directions of braces are adjusted according to the agent actions. The agent surrogate functions are adjusted using the Adam optimizer. The mini-batch size is set to 32 and the learning rates are 10^{-7} and 10^{-6} for policy and value function, respectively. In the value function, the NN in Eq. 12 in Section 5.2 has two hidden layers, each consisting of 200 neurons. The mean square error is used as the loss function, and learning rates are reduced by a factor of $\beta = 0.1$ every 200 games (20,000 steps). Weights and biases of the surrogate functions are synchronized with those of the online functions every 100 steps using $\tau = 0.05$. The GCN-DDPG agent is trained for 1,000 games. In Figure 6, the vertical axis represents the cumulative reward obtained during training, and the horizontal axis is the game number. The thick line shows the moving average of the reward with a window size set to 50. From Figure 6, the cumulative reward increases during the first 200 games and then remains stable around a certain value. Because support and bracing directions are changed at every training game, the fact that the cumulative reward stabilizes indicates that the agent has the learning

TABLE 3 Total computational cost of structural analysis of each method.

Support conditions	Grid size	Size of the global stiffness matrix $n_D \times n_D$	Number of structural analyses (times)		
			GCN-DDPG (test phase)	Benchmarks	
				EM	GA
1	4 × 4	54 × 54	1,000	65,536	5,000
	4 × 6	90 × 90	1,000	—	5,000
	6 × 6	150 × 150	1,000	—	5,000
	4 × 10	198 × 198	1,000	—	5,000
	10 × 6	270 × 270	1,000	—	5,000
	10 × 10	486 × 486	1,000	—	5,000
	20 × 20	2,166 × 2,166	1,000	—	5,000
2	4 × 4	126 × 126	1,000	65,536	5,000
	4 × 6	186 × 186	1,000	—	5,000
	6 × 6	270 × 270	1,000	—	5,000
	4 × 10	306 × 306	1,000	—	5,000
	10 × 6	438 × 438	1,000	—	5,000
	10 × 10	702 × 702	1,000	—	5,000
	20 × 20	2,622 × 2,622	1,000	—	5,000
3	4 × 4	90 × 90	1,000	65,536	5,000
	4 × 6	138 × 138	1,000	—	5,000
	6 × 6	210 × 210	1,000	—	5,000
	4 × 10	234 × 234	1,000	—	5,000
	10 × 6	378 × 378	1,000	—	5,000
	10 × 10	594 × 594	1,000	—	5,000
	20 × 20	2,394 × 2,394	1,000	—	5,000
4	4 × 4	102 × 102	1,000	65,536	5,000
	4 × 6	150 × 150	1,000	—	5,000
	6 × 6	234 × 234	1,000	—	5,000
	4 × 10	246 × 246	1,000	—	5,000
	10 × 6	402 × 402	1,000	—	5,000
	10 × 10	642 × 642	1,000	—	5,000
	20 × 20	2,502 × 2,502	1,000	—	5,000
Total computational cost of structural analysis including the training phase			128,000		140,000

capability to optimize structural configurations of different topology and support conditions.

6.3 Test phase

In the test phase, 4 × 4, 4 × 6, 6 × 6, 4 × 10, 10 × 6, 10 × 10, and 20 × 20-grid shells with pre-determined geometries are employed to investigate the capability of the trained agent on configurations that have not been tested in the training phase. In Figures 7–10, four support conditions denoted as 1, 2, 3, and 4 are considered for each frame model. The bracing directions are initialized randomly.

Similarly to the training phase, the number of steps for the test phase is 100. However, in the test phase, only actions improving the objective function value are accepted at each step. The agent optimizes each structural model 10 times. Table 2 shows the minimum (Min.), mean, and standard deviation (Std.) for the strain energy, and mean energy reduction rate (Reduction) for each structural model.

TABLE 4 Comparison of results obtained by GCN-DDPG (test phase) and benchmarks.

Support conditions	Grid size	Min. (N·m)	Method	Benchmarks (N·m)	Diff
1	4 × 4	4.38	EM/GA	4.38/4.38	0.00
	4 × 6	15.71	GA	15.66	0.00
	6 × 6	63.15		61.55	0.03
	4 × 10	72.49		71.88	0.01
	6 × 10	450.47		422.35	0.07
	10 × 10	2,242.26		2,135.28	0.05
	20 × 20	213,715.82		202,566.99	0.06
2	4 × 4	77.88	EM/GA	77.88/77.88	0.00
	4 × 6	284.81	GA	279.67	0.02
	6 × 6	818.14		777.99	0.05
	4 × 10	2,124.56		2046.13	0.04
	6 × 10	4,642.28		4,342.47	0.07
	10 × 10	17,683.39		16,294.39	0.09
	20 × 20	1,275,569.06		1,170,860.54	0.09
3	4 × 4	32.11	EM/GA	31.59/31.59	0.02
	4 × 6	70.09	GA	66.69	0.05
	6 × 6	345.73		342.38	0.01
	4 × 10	193.91		182.56	0.06
	6 × 10	2,841.17		2,725.89	0.04
	10 × 10	8,219.87		7,697.87	0.07
	20 × 20	589,997.91		553,614.84	0.07
4	4 × 4	40.81	EM/GA	40.77/40.77	0.00
	4 × 6	156.09	GA	150.95	0.03
	6 × 6	527.56		485.53	0.09
	4 × 10	1,192.81		1,140.11	0.05
	6 × 10	3,466.03		3,150.67	0.10
	10 × 10	12,629.96		11,462.31	0.10
	20 × 20	920,936.67		850,282.94	0.08

In the test phase, the strain energy of the structure can be reduced using the proposed method by 5–25%, depending on the structure size and support conditions. For all cases, the minimum and mean of strain energy obtained from 10 tests are very similar, and the standard deviations are low compared with the mean. The trained agent is capable to optimize the bracing directions to reduce the strain energy on configurations that were not tested in the training phase.

6.4 Comparison of computation cost and performance with EM and GA

For the 4×4-grid structure, the global optimal solution can be obtained using the EM which generates 2^k combinations of bracing directions for k grid cells. For structures with a greater number of grids, it is not feasible to use EM. Therefore, the GA is employed to obtain optimal solutions (global optimum cannot be guaranteed). GA is a meta-heuristic method inspired by the process of natural evolution which can be used to solve combinatorial optimization problems. The key operations in GA are *selection* to transfers good solutions from one generation to the next generation, *crossover* to generate new solutions, and *mutation* to modify solutions with a certain probability, which can be helpful to avoid local minima. In this research, bracing directions in each grid cell are represented through binary strings and the GA algorithm is taken from the GA python library named Distributed Evolutionary Algorithms in Python (DEAP) (Fortin et al., 2012). The comparison is made to

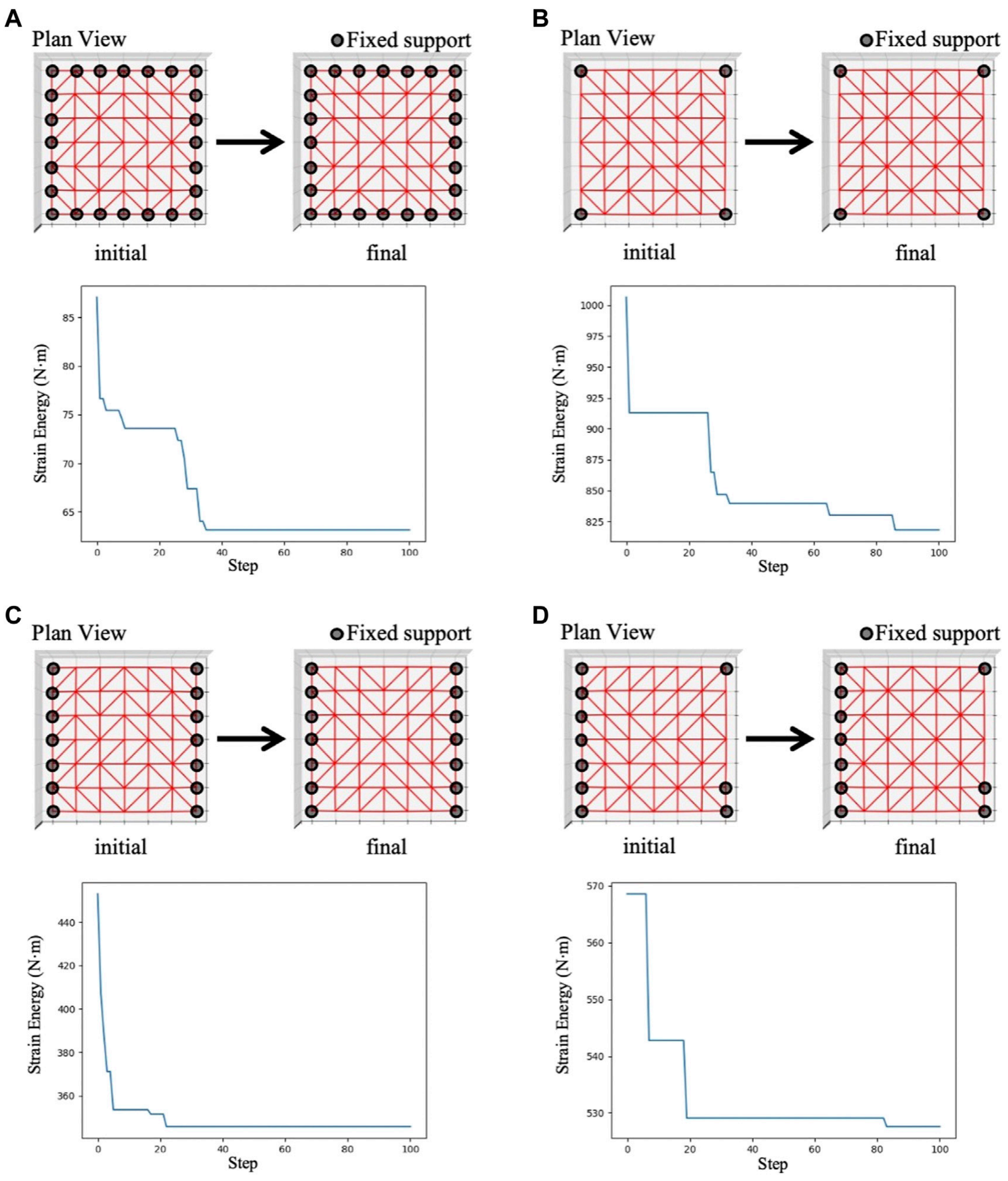

FIGURE 11
GCN-DDPG result of the 6×6-grid structure in the test phase; **(A)** Support condition 1, **(B)** Support condition 2, **(C)** Support condition 3, **(D)** Support condition 4.

benchmark both the performance and computational efficiency of GCN-DDPG and GA for the early-stage design of grid shells where several configurations are to be evaluated. Therefore, the number of population and generation of GA are determined based on feasibility of computational cost. In the following examples, the numbers of population and generations are 50 and 100, respectively.

The number of structural analyses carried out with GCN-DDPG in the training phase is equal to the number of training games multiplied by the number of steps in each training game, which is 100,000. A trained agent can be used in the test phase and in other problems without re-training. The computational cost of GCN-DDPG in the test phase, EM, and GA for each problem, and the size of the global stiffness matrix are shown in Table 3 where the last row indicates the total computational cost. The total number of structural analyses required by GCN-DDPG is less than that required by GA. Since a significant computation time is required for the analysis of large-size structures, the GCN-DDPG is more efficient than the GA when applied to bracing direction optimization of grid shell structures that comprise many elements.

Benchmark results are compared in Table 4. The ratio of the difference between the minimum strain energy solution obtained by RL and that obtained by GA (and EM) is shown in the column labeled 'Diff', which is computed as follows:

$$\text{Diff} = \left(\min\left(\text{Result}_{\text{RL}}\right) - \text{Result}_{\text{GA}}\right)/\text{Result}_{\text{GA}} \qquad (15)$$

From Table 4, the solution quality and the efficiency of the GCN-DDPG agent can be verified. In most cases, results obtained by the GCN-DDPG agent are comparable to those obtained by EM and GA within a margin of 10% difference using less computational cost. The proposed method is useful in early-stage design, which typically requires testing several structures, and therefore an efficient computational process is needed. The RL agent could be further trained using other structural configurations to improve its performance.

Figure 11 shows initial brace topology, final brace topology, and the change of strain energy of GCN-DDPG best results for 6 × 6-grid structural models. Although the structural configurations differ considerably from those used in the training phase in terms of support conditions and size, the agent is capable to minimize the strain energy by adjusting the bracing directions. Therefore, it is possible to train the agent using small-size structural models and use it to optimize structures with different support conditions and sizes.

From Figures 11A,C where the support locations are symmetric, the agent obtains solutions with symmetrical layouts, despite the fact that a symmetrical feature is not explicitly represented.

7 Conclusion

A combined method of DDPG and GCN has been formulated for bracing direction optimization of grid shell structures to minimize the strain energy. The proposed DDPG framework allows the agent to modify the bracing direction in all grid cells simultaneously at each optimization step. The node feature matrix, adjacency matrices, and weighted adjacency matrices are formulated to encode the structural configuration and internal forces as graph representations. The agent is trained using Markov Decision Process (MDP) in the RL framework whereby training data are collected by interacting with the environment. The value function or critic network updates internal weights and biases to minimize the prediction loss for the accumulated reward or Q-value so that it can predict the long-term accumulated reward from state and action. The policy function or actor network updates weights and biases to maximize the equivalent reward calculated by the value function.

Numerical examples show that the trained agent can effectively optimize bracing directions to minimize the strain energy in the test phase. The agent is capable to optimize the bracing direction of structural configurations with size and support conditions different than those in the training phase. The proposed method produces solutions that compare, albeit of marginally lower quality, with those produced through the enumeration method (EM) and the genetic algorithm (GA). However, the trained agent can be employed for additional configurations to those tested in this work. The agent performs well for relatively large structural models without the re-training, thereby significantly reducing the computational cost of optimization. Future work should investigate whether the RL method can be applied without re-training to design significantly larger-size structural configurations (e.g., 200 × 200 grid size) compared to those employed for training. Therefore, The proposed method has good potential to be employed effectively in early-stage design, which typically requires testing several configurations.

Data availability statement

Experiment data from this research are available on the request to corresponding authors.

Author contributions

C-tK: Conceptualization, implementation, writing-original draft, data curation. KH: Conceptualization, writing-review and editing, data curation, resource. MO: Conceptualization, writing-review and editing, data curation, resource.

Funding

This work was supported by MEXT scholarship (Grant Number 180136); and JSPS KAKENHI (Grant Numbers JP 20H04467, JP 21K20461).

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

References

Aich, S., and Stavness, I. (2018). *Global sum pooling: A generalization trick for object counting with small datasets of large images*. Arxiv:1805.11123. [Online]. Available at: https://arxiv.org/abs/1805.11123.

Bellman, R. (1954). The theory of dynamic programming. *Bull. Amer. Math. Soc.* 60 (6), 503–515. doi:10.1090/s0002-9904-1954-09848-8

Bellman, R. (1957). A markovian decision process. *Indiana Univ. Math. J.* 6, 679–684. doi:10.1512/iumj.1957.6.56038

Berke, L., and Hajela, P. (1993). "Application of neural nets in structural optimization," in *Optimization of large structural systems. NATO ASI series (Series E: Applied sciences)*. Editor G. I. N. Rozvany (Dordrecht: Springer), Vol. 231. doi:10.1007/978-94-010-9577-8_36

Chigozie, N., Ijomah, W., Gachagan, A., and Stephen, M. (2020). *Activation functions: Comparison of trends in practice and research for deep learning*. Arxiv:1811.03378 [Online]. Available at: https://arxiv.org/abs/1811.03378.

Christensen, P. W., and Klarbring, A. (2009). *An introduction to structural optimization. Solid mechanics and its applications*, Vol. 153. Dordrecht: Springer. doi:10.1007/978-1-4020-8666-3

Dhingra, A. K., and Bennage, W. A. (1995). Topological optimization of truss structures using simulated annealing. *Eng. Optim.* 24 (4), 239–259. doi:10.1080/03052159508941192

Eltouny, K., and Liang, X. (2021). Bayesian-optimized unsupervised learning approach for structural damage detection. *Computer-Aided. Civ. Infrastructure Eng.* 36, 1249–1269. doi:10.1111/mice.12680

Fortin, F. A., De Rainville, F. M., Gardner, M. A., Parizeau, M., and Gagné, C. (2012). DEAP: Evolutionary algorithms made easy. *J. Mach. Learn. Res. Mach. Learn. Open Source Softw.* 13, 2171–2175.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. The MIT Press. 9780262035613. Available at: https://www.deeplearningbook.org.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). *Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor*. ICML, 1856–1865. Arxiv:1801.01290. Available at: https://dblp.uni-trier.de/db/conf/icml/icml2018.html#HaarnojaZAL18.

Hayashi, K., and Ohsaki, M. (2021). Reinforcement learning and graph embedding for binary truss topology Optimization under Stress and Displacement Constraints. *Front. Built Environ.* 6, 59. doi:10.3389/fbuil.2020.00059

Hung, T. V., Viet, V. Q., and Thuat, D. V. (2019). A deep learning-based procedure for estimation of ultimate load carrying of steel trusses using advanced analysis. *J. Sci. Technol. Civ. Eng. (STCE) - HUCE* 13 (3), 113–123. doi:10.31814/stce.nuce2019-13(3)-11

Ivakhnenko, A. G. (1968). The group method of data handling – a rival of the of stochastic approximation. *Sov. Autom. Control* 13 (3), 43–55.

Jeong, M. J., and Yoshimura, S. (2002). "An evolutionary clustering approach to pareto solutions in multiobjective optimization," in Proceedings of the ASME 2002 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, 141–148. doi:10.1115/DETC2002/DAC-34048

Kanno, Y., and Fujita, S. (2018). Alternating direction method of multipliers for truss topology optimization with limited number of nodes: a cardinality-constrained second-order cone programming approach. *Optim. Eng.* 19 (2), 327–358. doi:10.1007/s11081-017-9372-3

Kawamura, H., Ohmori, H., and Kito, N. (2002). Truss topology optimization by a modified genetic algorithm. *Struct. Multidiscipl. Optim.* 23, 467–473. doi:10.1007/s00158-002-0208-0

Kiefer, J., and Wolfowitz, J. (1952). Stochastic estimation of the maximum of a regression function. *Ann. Math. Stat.* 23 (3), 462–466. doi:10.1214/aoms/1177729392

Kingma, D., and Ba, J. (2015). "Adam: a method for stochastic optimization," in 2014, Published as a conference paper at the 3rd International Conference for Learning Representations (San Diego). Arxiv:1412.6980.

Kipf, T. N., and Welling, M. (2016). *Variational graph auto-encoders*. Arxiv: 1611.07308. Available at: https://arxiv.org/abs/1611.07308.

Kipf, T. N., and Welling, M. (2017). "Semi-supervised classification with graph convolutional networks," in Proceedings of the 5th international conference on learning representations. Arxiv:1609.02907.

Kociecki, M., and Adeli, H. (2015). Shape optimization of free-form steel space-frame roof structures with complex geometries using evolutionary computing. *Eng. Appl. Artif. Intell.* 38, 168–182. ISSN 0952-1976. doi:10.1016/j.engappai.2014.10.012

Kupwiwat, C., and Yamamoto, K. (2020). Fundamental study on morphogenesis of shell structure using reinforcement learning. *Struct. I* 2020, 933–934. Architectural Institute of Japan. Available at: https://ci.nii.ac.jp/naid/200000462858/en/(Access March 17, 2022).

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., et al. (2016). *Continuous control with deep reinforcement learning, ICLR*. Arxiv:1509.02971.

MacQueen, J. (1967). "Some methods for classification and analysis of multivariate observations," in 5-th Berkeley Symposium on Mathematical Statistics and Probability, 281–297.

Mai, T. H., Kang, J., and Lee, J. (2021). A machine learning-based surrogate model for optimization of truss structures with geometrically nonlinear behavior. *Finite Elem. Analysis Des.* 196, 103572. ISSN 0168-874X. doi:10.1016/j.finel.2021.103572

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., et al. (2013). *Playing Atari with deep reinforcement learning*. Arxiv:1312.5602. Available at: https://arxiv.org/abs/1312.5602.

Nair, V., and Hinton, G. E. (2010). *Rectified linear units improve restricted boltzmann machines*. Haifa, 807–814. Available at: https://dl.acm.org/citation.cfm.

Ohsaki, M., and Swan, C. (2002). "Topology and geometry optimization of trusses and frames," in *Recent advances in optimal structural design*.

Ohsaki, M. (1995). Genetic algorithm for topology optimization of trusses. *Comput. Struct.* 57 (2), 219–225. ISSN 0045-7949. doi:10.1016/0045-7949(94)00617-C

Ohsaki, M. (2010). *Optimization of finite dimensional structures*. 1st ed.. Boca Raton, FL: CRC Press. doi:10.1201/EBK1439820032

Puentes, L., Cagan, J., and McComb, C. (2021). Data-driven heuristic induction from human design behavior. *J. Comput. Inf. Sci. Eng.* 21 (2). doi:10.1115/1.4048425

Robbins, H., and Monro, S. (1951). A stochastic approximation method. *Ann. Math. Stat.* 22 (3), 400–407. doi:10.1214/aoms/1177729586

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychol. Rev.* 65 (6), 386–408. doi:10.1037/h0042519

Ruder, S. (2016). *An overview of gradient descent optimization algorithms*. Arxiv: 1609.04747. Available at: https://arxiv.org/abs/1609.04747.

Sutton, R. S., and Andrew, G. B. (2018). *Reinforcement learning, an introduction*. 2nd ed. Cambridge, MA: The MIT Press. 9780262039246.

Topping, B. H. (1983). Shape optimization of skeletal structures: A review. *J. Struct. Eng. (N. Y. N. Y).* 109, 1933–1951. doi:10.1061/(asce)0733-9445(1983)109:8(1933)

Uhlenbeck, G. E., and Ornstein, S. L. (1930). On the theory of the Brownian motion. *Phys. Rev.* 36 (5), 823–841. doi:10.1103/PhysRev.36.823

van der Maaten, L., and Hinton, G. (2008). Visualizing data using t-SNE. *J. Mach. Learn. Res.* 9, 2579–2605. Available at: http://jmlr.org/papers/v9/vandermaaten08a.html (Access March 17, 2022).

Wang, D., Zhang, W. H., and Jiang, J. S. (2002). Truss shape optimization with multiple displacement constraints. *Comput. Methods Appl. Mech. Eng.* 191 (33), 3597–3612. ISSN 0045-7825. doi:10.1016/S0045-7825(02)00297-9

Yu, A., Palefsky-Smith, R., and Bedi, R. (2019). *Deep reinforcement learning for simulated autonomous vehicle control*. Stanford, CA, USA: Stanford University. Available at: http://vision.stanford.edu/teaching/cs231n/reports/2016/pdfs/112_Report.pdf (Access March 17, 2022).

Zhu, S., Ohsaki, M., Hayashi, K., and Guo, X. (2021). Machine-specified ground structures for topology optimization of binary trusses using graph embedding policy network. *Adv. Eng. Softw.* 159, 103032. ISSN 0965-9978. doi:10.1016/j.advengsoft.2021.103032