DEEP DETERMINISTIC POLICY GRADIENT AND GRAPH CONVOLUTIONAL NETWORKS FOR TOPOLOGY OPTIMIZATION OF BRACED STEEL FRAMES

Chi-tathon KUPWIWAT[*], *Yuichi IWAGOE*[**], *Kazuki HAYASHI*[***] and *Makoto OHSAKI*[****]

We propose a method for topology optimization of braced frames under static seismic loads using Deep Deterministic Policy Gradient (DDPG) and Graph Convolutional Network (GCN). The structure is interpreted as a graph where structural elements and element configurations are represented by the node feature matrix and adjacency matrices, respectively. Using this graph representation, the DDPG agent with GCN architecture can observe the properties of the frame, and make the decision to either add braces into the frame or enlarge sections of frame elements by selecting from a list of available sections. During the optimization process, the initial structure that cannot withstand the seismic load is modified by the agent until all constraints are satisfied. The trained agent can be applied to frames of different sizes and can obtain competitive results with less computational cost compared to the genetic algorithm.

Keywords **:** *Topology optimization, Plane building frame, Reinforcement learning, Deep deterministic policy gradient, Graph representation, Graph convolutional network*

1. INTRODUCTION

Optimization of discrete structures, such as frames and trusses, has widely been studied in the field of structural optimization. When nodal connectivity and/or cross-sectional properties are chosen as design variables in the optimization problem, the problem is classified as a topology optimization problem[1]. Optimal solutions of this problem can be obtained with small computational effort when the numbers of members and degrees of freedom (DOFs) are small. However, the problem may become more difficult to solve for real-world structures which have larger numbers of members and DOFs. Moreover, when design variables are chosen from a set of discrete values, the topology optimization is considered as a combinatorial problem which is difficult to solve, because the gradient of the objective and constraint functions with respect to the design variables are not available[2].

Optimization methods can be classified into mathematical programming, in which the gradient with respect to design variables is needed, and heuristic approach, in which the gradient information is not utilized. However, for heuristic approaches, large computational cost is required for computing and comparing structural responses of multiple candidates. The heuristic approach is further classified into a population-based approach such as a genetic algorithm (GA)[3] and a local search approach such as simulated annealing (SA)[4]. In application of the population-based approach to structural optimization problems [Refs.5-8], multiple candidates (i.e., population) for optimal solutions are generated in each optimization step. Prospective candidates are then selected, based on their structural responses, for generating the candidates at the next optimization step, and the optimization process is continued until the termination criteria are satisfied.

Machine learning (ML) has recently been a subject of study for solving engineering problems in various fields including structural design and optimization problems. Vanluchene and Sun[9] applied an ML approach to design reinforced concrete beams. Zheng et al.[10] trained an ML model to predict architect's preference of generated structures based on graphic statics. Mirra and Pugnale[11] used variational autoencoder[12] to design shell structures. For the braced frames, Tamura et al.[13] proposed combining ML, such as binary decision tree or support vector machine, with SA for optimization of brace locations of building frames. Sakaguchi et al.[14] proposed methods for extracting important features, and converting the features of a small frame to those of a large frame. In both studies, the sizes of beams and columns are fixed, and only the types and locations of the braces are optimized.

Types of application of ML are classified into function approximation (regression), classification of solutions (optimal/non-optimal, feasible/infeasible), and learning optimization process by reinforcement learning (RL), which trains the agent to perceive the environment around itself and

* Ph.D. Student, Dept of Architecture and Architectural Engineering, Kyoto University, M. Eng.
** Graduate Student, Dept of Architecture and Architectural Engineering, Kyoto University (Currently, Morikita Publishing Co.,Ltd.), M. Eng.
*** Assistant Prof., Dept of Architecture and Architectural Engineering, Kyoto University, Dr. Eng.
**** Prof., Dept of Architecture and Architectural Engineering, Kyoto University, Dr. Eng.

make decisions to accomplish the given tasks. Thus, the RL agent can be trained to learn an optimal decision process. After the training, the agent can be applied to do tasks in other similar environments. Deep Deterministic Policy Gradient (DDPG)[15] is an RL algorithm using two neural networks (NNs), namely actor and critic networks[16)–18)]. The actor network is trained to maximize the estimated reward, and the critic network is trained to accurately estimate the reward.

Graph is a type of data consisting of vertices (nodes) and edges. A graph can be processed and manipulated using convolutional operators or graph signal processing methods[19),20)]. Graph representation can be effectively utilized for modeling various components of the building. Langenhean et al.[21)] used graphs to represent room types. Abualdenien and Borrmann[22)] used graph representation for patterns of building elements. Vestartas[23)] represented structural elements and joints using a graph. Hayashi and Ohsaki[24)] proposed a combined method of graph representation and RL for binary topology optimization of the trusses. Kupwiwat et al.[25)] proposed a method for binary optimization of braced lattice shells using an RL agent made of Graph Convolutional Network (GCN)[26)] which is a type of NN that works effectively with graph data. However, in Kupwiwat et al. paper, the dot product of the agent output is needed to identify modifications of the element which is too complicated. This paper proposed an improved method where the modifications of the element can be directly obtained from the agent output.

This paper proposes a method for combinatorial optimization of member sizes and brace placements of plane building frames under constraints on static seismic responses using graph representation and a DDPG agent introducing GCN. The agent observes properties of the frame through graph representation and sequentially adjusts the frame by adding braces or enlarging the sizes of beams and columns. During the training, the penalty determined by material and construction cost is given to the agent. It is shown that the optimal design modification can be learned by the proposed method.

The rest of the paper is organized as follows. Section 2 explains the formulations of objective and constraint functions of the combinatorial optimization problem. Sections 3 and 4 introduce the graph representation and DDPG, respectively. Section 5 explains components of the decision-making process to implement RL. In Section 6, numerical examples of the training and test phases are presented. The learnability and applicability of the agent are shown in the training phase, and solutions obtained by the RL agent in the test phase are compared to those obtained by the GA.

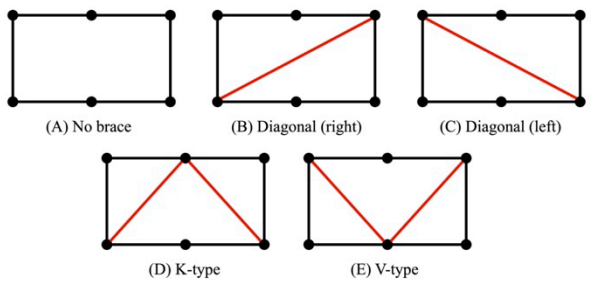


Fig. 1 Five brace types including no brace

2. COMBINATORIAL OPTIMIZATION PROBLEM

2.1 Outline of optimization problem

Seismic performance is an important factor in designing building frames in earthquake-prone areas. In this paper, the seismic load is simplified using the Japanese building code of allowable stress design based on so-called Ai distribution for *short-term loading*, whereas the dead and live loads are building frame is improved by increasing the sizes of beams and columns and placing braces of various types including diagonal braces, K-type, and V-type as shown in Figure 1.

The building frame without braces is initialized with the smallest section index in the lists. During the optimization, braces are placed into the building frame together with adjustment of sizes of beams, columns, and braces until the building frame can satisfy all constraints on the structural responses.

2.2 Static seismic loads

The horizontal seismic force on each floor is computed by distributing shear forces. Horizontal load P_i on the ith floor of the building frame is computed as follows:

$$P_i = \begin{cases} Q_i & (i = N_\mathrm{f}) \\ Q_{i+1} - Q_i & (i = 1, \ldots, N_\mathrm{f} - 1) \end{cases} \tag{1.1}$$

$$Q_i = C_i \sum_{j=i}^{N_\mathrm{f}} W_j \tag{1.2}$$

$$C_i = Z_0 R C_0 \mathcal{A}_i \tag{1.3}$$

$$R = \begin{cases} 1 & (T < T_\mathrm{c}) \\ 1 - 0.2\left(\frac{T}{T_\mathrm{c}} - 1\right)^2 & (T_\mathrm{c} \le T < 2T_\mathrm{c}) \\ 1.6\frac{T}{T_\mathrm{c}} & (2T_\mathrm{c} \le T) \end{cases} \tag{1.4}$$

$$T = 0.003h \tag{1.5}$$

$$\mathcal{A}_i = 1 + \left(\frac{1}{\sqrt{\alpha_i}} - \alpha_i\right)\frac{2T}{1+3T} \tag{1.6}$$

$$\alpha_i = \frac{\sum_{j=i}^{N_\mathrm{f}} W_j}{\sum_{k=1}^{N_\mathrm{f}} W_k} \tag{1.7}$$

$$W_i = w_i a_i \tag{1.8}$$

where Q_i, C_i, W_i, R, and $\mathcal{A}_i$ are the shear force acting on the ith story between the (i-1)th and ith floors, shear coefficient of the ith story, the weight of the ith floor, a value representing soil category and vibration characteristics of the building, and the shear distribution coefficient of the ith floor, respectively. C_0, T, w_i, and a_i are the base shear coefficient, the fundamental natural period of the building, the unit load of the ith floor, and the area of the ith floor, respectively. h, T_c, Z_0, and N_f are the total height of the building, the specific period assigned according to the soil type under the building, the regional seismic coefficient, and the number of floors excluding the base, respectively.

In the following examples, *short-term loadings* in both left and right directions are considered to evaluate the largest response of the building frame.

2.3 Structural response

Internal forces and displacements are computed using linear elastic analysis. The beams and columns are modeled using 2-dimensional beam elements with 6-DOFs, whereas the braces are modeled using 2-dimensional truss elements with 4-DOFs. In the local coordinate system, the stiffness

matrices of the beam or column element and the brace element are denoted as $\mathbf{k}_\mathrm{f} \in \mathbb{R}^{6\times6}$ and $\mathbf{k}_\mathrm{e} \in \mathbb{R}^{4\times4}$, respectively. These matrices are transformed into the global coordinate system and assembled into the global stiffness matrix $\mathbf{K} \in \mathbb{R}^{n_\mathrm{D}\times n_\mathrm{D}}$, where n_D is the number of DOFs of the frame.

The load vectors corresponding to the three cases of *long-term loading*, *long-term loading* with left direction (negative) *short-term loading*, and *long-term loading* with right direction (positive) *short-term loading* are denoted by $\mathbf{p}_\mathrm{L} \in \mathbb{R}^{n_\mathrm{D}}$, $\mathbf{p}_{\mathrm{S}^-} \in \mathbb{R}^{n_\mathrm{D}}$, and $\mathbf{p}_{\mathrm{S}^+} \in \mathbb{R}^{n_\mathrm{D}}$, respectively. Nodal displacement vectors $\mathbf{d}_\mathrm{L} \in \mathbb{R}^{n_\mathrm{D}}$, $\mathbf{d}_{\mathrm{S}^-} \in \mathbb{R}^{n_\mathrm{D}}$, and $\mathbf{d}_{\mathrm{S}^+} \in \mathbb{R}^{n_\mathrm{D}}$ corresponding to the three load cases, respectively, are obtained by solving the following stiffness equation:

$$\mathbf{K}\mathbf{d} = \mathbf{p} \tag{2}$$

where $\mathbf{d} \in \{\mathbf{d}_\mathrm{L}, \mathbf{d}_{\mathrm{S}^-}, \mathbf{d}_{\mathrm{S}^+}\}$ and $\mathbf{p} \in \{\mathbf{p}_\mathrm{L}, \mathbf{p}_{\mathrm{S}^-}, \mathbf{p}_{\mathrm{S}^+}\}$.

The internal forces $\mathbf{f}_i = \left(f_p^x, f_p^y, f_p^\theta, f_q^x, f_q^y, f_q^\theta\right)$ of a beam or column element i can be obtained from p and q end displacements of the element i in the local coordinate system and its corresponding stiffness matrix. See Figure 2 for definitions of forces. Internal forces $\mathbf{f}_i = \left(f_p^x, f_p^y, f_q^x, f_q^y\right)$ of a brace element can be obtained similarly. From these internal forces, internal stress σ_i of a structural element i can be computed as follow:

$$\sigma_i = \frac{|f_p^x|}{A_i} + \frac{\max\left(|f_p^\theta|, |f_q^\theta|\right)}{Z_i} \tag{3}$$

where A_i and Z_i denote the cross-sectional area and the section modulus of element i, respectively, and the second term does not exist for a truss element. Note that this paper uses the assumption of the rigid floor where in-plane stiffness of slab is incorporated by multiplying axial stiffness of the beam element by 10.

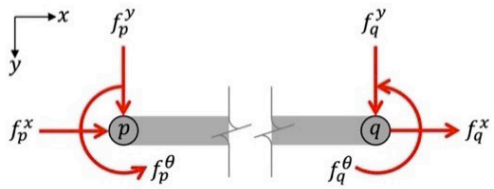


Fig. 2 Internal forces in local coordinates

2.4 Objective function

The optimal section sizes of beams, columns, and braces as well as brace placements of steel frames are to be obtained for minimizing the objective function formulated considering material and construction costs. Let $c_i \in \{0,1,\ldots,n_\mathrm{c}\}$ and $b_i \in \{0,1,\ldots,n_\mathrm{b}\}$ denote the indices of the predetermined cross-sectional properties of columns and beams, respectively. A vector representing the element properties of the building frame which has c_f columns and b_f beams is denoted as

$$\mathbf{A} = \left(c_1, \ldots, c_{c_\mathrm{f}}, b_1, \ldots, b_{b_\mathrm{f}}\right) \tag{4}$$

A vector that represents the type of brace in each domain of a building frame with N_s spans and N_f stories is denoted as

$$\mathbf{B} = \left(r_1, \ldots, r_{N_\mathrm{s}N_\mathrm{f}}\right), \quad r_i \in \{0,1,2,3,4\} \tag{5}$$

where 0, 1, …, 4 correspond to the brace types (A), (B), …, (E) in Figure 1, respectively.

A design variable vector for a building frame can be represented as a combination of $\mathbf{A}$ and $\mathbf{B}$ as

$$\mathbf{X} = \{\mathbf{A}, \mathbf{B}\} \tag{6}$$

The objective function to be minimized is the cost denoted by $V(\mathbf{X})$ which is equivalent to the total structural volume. Note that the cost of braces is multiplied by the cost coefficient B to avoid placing too many braces. The optimization problem of member sizes and brace placements of the steel frame is formulated as follows:

$$\text{minimize} \quad V(\mathbf{X}) \tag{7.1}$$

$$\text{subject to} \quad \sigma_\mathrm{max}^\mathrm{L} \le \hat{\sigma}^\mathrm{L} \tag{7.2}$$

$$\sigma_\mathrm{max}^\mathrm{S} \le \hat{\sigma}^\mathrm{S} \tag{7.3}$$

$$\theta_\mathrm{max} \le \hat{\theta} \tag{7.4}$$

$$\delta_\mathrm{max} \le 1 \tag{7.5}$$

$$\lambda_\mathrm{max} \le 1 \tag{7.6}$$

where $\sigma_\mathrm{max}^\mathrm{L}$, $\sigma_\mathrm{max}^\mathrm{S}$, and θ_max are the maximum absolute value of stresses at the element ends among all members due to *long-term loading*, the largest maximum absolute value of stresses at the element ends among all members due to both directions of *short-term loading*, and the maximum absolute value of inter-story drift angles among all stories, respectively. Their upper bounds are denoted by $\hat{\sigma}^\mathrm{L}$, $\hat{\sigma}^\mathrm{S}$, and $\hat{\theta}$, respectively. Note that the two cases of positive and negative directions of horizontal loads are considered for evaluating the maximum value of responses for the short-term loadings. Constraints on base beams are also measured because they are necessary for adding V-type braces. δ_max is the maximum value of deflection ratio δ_i^b at the centers of all beams defined as

$$\delta_i^\mathrm{b} = \max\left(\frac{300\times|d_i^\mathrm{b}|}{L_i^\mathrm{b}}, 1\right) \tag{8}$$

where d_i^b and L_i^b are the deflection at the center of beam i in the local y coordinate, and the length of beam i, respectively. λ_max is the maximum value of buckling stress ratios λ_i, defined as follows, of all braces:

$$\lambda_i = \begin{cases} \frac{\sigma_i^\mathrm{S}}{\pi^2 E_i I_i/(A_i L_i^2)} & (f_p^x > 0) \\ 0 & (f_p^x \le 0) \end{cases} \tag{9}$$

3. GRAPH CONVOLUTIONAL NETWORK

3.1 Graph representation

In this paper, the building frame structure is represented as a graph which consists of *nodes* representing structural elements (i.e., columns, beams, and braces) and *edges* representing connections between the structural elements as shown in Figure 3. The graph representation is a method to store graph data in vector or matrix forms.

3.2 Graph Convolutional Network

GCN is a type of NN that can map the input of graph representations to the target domain of the graph. The graph consisting of n nodes with u features can be represented using a node feature matrix $\mathbf{N} \in \mathbb{R}^{n\times u}$ to represent the features of each node in the graph. An adjacency matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$ indicates the connectivity between nodes such that each entry $m_{i,j}$ is 1 if there is an edge connecting node *i* and node *j*, or 0 if there is no edge connecting node *i* and node *j*, respectively. A diagonal degree matrix $\mathbf{D} \in \mathbb{R}^{n\times n}$ has each entry $d_{i,i}$ representing the number of edges connected to the node *i*. Note again that the *nodes* in the graph represent structural elements; not the connections of joints between the elements.

A single GCN computation, corresponding to a *layer*, takes these representations as input and computes the output as follows:

$$\mathbf{N}' = \sigma(\widetilde{\mathbf{M}}\mathbf{N}\mathbf{w}) \tag{10}$$

where σ and $\mathbf{w} \in \mathbb{R}^{u \times v}$ are a non-linear activation function and the weight matrix (i.e., the convolution filter parameters with v filters) in the GCN layer which is adjusted during the training, respectively.

$\widetilde{\mathbf{M}}$ is the normalized adjacency matrix computed as

$$\widetilde{\mathbf{M}} = \mathbf{D}^{-1/2}[\mathbf{M} + \mathbf{I}]\mathbf{D}^{-1/2} \tag{11}$$

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

Multiple GCN *layers* can be connected together, in which output $\mathbf{N}' \in \mathbb{R}^{n \times v}$ of the previous GCN *layer* is treated as node input $\mathbf{N}$ of the next GCN *layer*, to create a machine learning model.

In this paper, we utilize GCN to build an RL agent that can determine how to adjust the design of a structure. Let N_a denote the number of actions assigned to each member, which is 1 in this study. The output of the agent is $\boldsymbol{\pi} \in \mathbb{R}^{n \times N_\mathrm{a}}$, representing the probability to choose the element whose cross-sectional area is to be enlarged.

4. REINFORCEMENT LEARNING

4.1 Outline of reinforcement learning

An agent aiming to obtain a high accumulated reward signal is trained to do actions in an environment that give a reward signal according to how the agent's action affects the environment. RL algorithm consists of three main components: a *policy* dictating the agent's behavior, a reward signal, and a *value function* that estimates the accumulated *reward signal*[27]. Interactions between the agent and the environment are represented using a discrete time decision-making process called Markov Decision Process (MDP)[28),29)], in which the next state depends solely on the current state and action. At step t, the agent observes the environment as state S_t and performs action A_t. After that, the agent receives reward signal R_{t+1} and observes the representation of the next state S_{t+1} as shown in Figure 4.

4.2 Deep Deterministic Policy Gradient

DDPG is an RL policy gradient algorithm that determines the probability of taking action A_t^i in a state S_t, denoted by $\boldsymbol{\pi}$, and predicts the accumulated reward (Q-value) from the actions using a policy function (*Actor*) $\boldsymbol{\pi}_{\theta_1}$ and a value function Q_{θ_2} (*Critic*), respectively, both of which are represented as follows:

$$\boldsymbol{\pi}_{\theta_1}(\mathrm{S}_t) = \boldsymbol{\pi} \tag{12}$$

$$\mathrm{Q}_{\theta_2}\left(\mathrm{S}_t, \boldsymbol{\pi}_{\theta_1}(\mathrm{S}_t)\right) = \sum_{v=1}^{\infty} \gamma^{v-1}\mathrm{R}_{t+v} \tag{13}$$

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

The agent interacts with the environment and stores MDP data of $\{\mathrm{S}_t, \mathrm{A}_t, \mathrm{R}_{t+1}, \mathrm{S}_{t+1}\}$ in a storage of training data called *replay buffer*. The value function adjusts its parameters to increase the accuracy of the reward prediction, and the policy function adjusts its parameters to increase the predicted reward, respectively. The *tau update* method[30] is used for stabilizing the learning process by training a surrogate policy function $\boldsymbol{\pi}'_{\theta'_1}$ and a surrogate value function $\mathrm{Q}'_{\theta'_2}$, and then gradually updating the parameters of these functions into the real policy function $\boldsymbol{\pi}_{\theta_1}$ that interact with the environment, and the real value function Q_{θ_2} using a small value of τ ($\tau \ll 1$) at every *tau update* interval.

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

DDPG Algorithm:

1. Sample n_{batch} training data $\{\mathrm{S}_t, \mathrm{A}_t, \mathrm{R}_{t+1}, \mathrm{S}_{t+1}\}$ from the stored replay buffer and changed 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:

$$\boldsymbol{\pi}'_{\theta'_1}(\mathbf{S}_t) = \widehat{\mathbf{A}}_t$$
$$\boldsymbol{\pi}_{\theta_1}(\mathbf{S}_{t+1}) = \widehat{\mathbf{A}}_{t+1}$$
$$\mathrm{Q}'_{\theta'_2}(\mathbf{S}_t, \mathbf{A}_t) = \widehat{\mathbf{Q}}_t$$
$$\mathrm{Q}_{\theta_2}(\mathbf{S}_{t+1}, \widehat{\mathbf{A}}_{t+1}) = \mathbf{Q}_{t+1}$$
$$\nabla \mathrm{Q}'_{\theta'_2} = \nabla_{\theta'_2}\mathrm{Q}'_{\theta'_2}(\mathbf{S}_t, \mathbf{A}_t)\nabla_{\widehat{\mathbf{Q}}_t}\mathcal{L}(\mathbf{R}_{t+1} + \mathbf{Q}_{t+1}, \widehat{\mathbf{Q}}_t)$$
$$\nabla \mathrm{J}'_{\theta'_1} = -\mathbb{E}\left[\nabla_{\theta'_1}\boldsymbol{\pi}'_{\theta'_1}(\mathbf{S}_t)\nabla_{\widehat{\mathbf{A}}_t}\mathrm{Q}'_{\theta'_2}(\mathbf{S}_t, \widehat{\mathbf{A}}_t)|_{\widehat{\mathbf{A}}_t = \boldsymbol{\pi}'_{\theta'_1}(\mathbf{S}_t)}\right]$$

Update $\boldsymbol{\theta}'_2$ in $\mathrm{Q}'_{\theta'_2}$ using $\nabla \mathrm{Q}'_{\theta'_2}$

Update $\boldsymbol{\theta}'_1$ in $\boldsymbol{\pi}'_{\theta'_1}$ using $\nabla \mathrm{J}'_{\theta'_1}$

If *tau update* interval is reached:

$$\boldsymbol{\theta}_1 = (1-\tau)\boldsymbol{\theta}_1 + \tau\boldsymbol{\theta}'_1$$
$$\boldsymbol{\theta}_2 = (1-\tau)\boldsymbol{\theta}_2 + \tau\boldsymbol{\theta}'_2$$

To reduce the time to find optimal weights in each layer of GCN, an optimizer such as stochastic gradient descent (SGD)[31),32),33)] or Adam[34)] is used for updating $\boldsymbol{\theta}'_1$ and $\boldsymbol{\theta}'_2$, using the gradients $\nabla \mathrm{J}'_{\theta'_1}$ and $\nabla \mathrm{Q}'_{\theta'_2}$, respectively. If the agent just exploits its policy to determine the best actions, the agent might miss other better actions. Therefore, Ornstein-Uhlenbeck noise[35)] is added to the output of the policy function to activate the exploration of DDPG's policy function during the training.

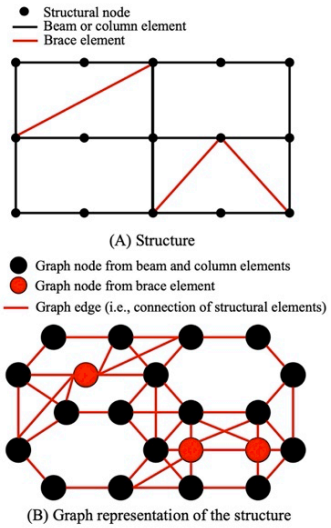


Fig. 3 Braced frame structure represented as a graph

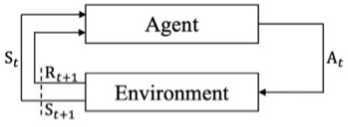


Fig.4 Diagram of the MDP

5. REINFORCEMENT LEARNING FOR OPTIMIZATION

5.1 State

The state of a building frame with all possible n elements is represented as graph data where each node has 31 features obtained from structural configuration and responses. The definitions of $n_{i,j}$ $(j = 1, \dots, 31)$ as follows:

- $n_{i,1}$ 1 if i is in the structure, or 0 if i is not in the structure
- $n_{i,2}$ 0 if i's sectional area is equal to the largest area for i, else 1
- $n_{i,3}$ 1 if i violates stress constraint under $\mathrm{p_{S-}}$, else 0
- $n_{i,4}$ 1 if i violates stress constraint under $\mathrm{p_{S+}}$, else 0
- $n_{i,5}$ 1 if i violates stress constraint under $\mathrm{p_L}$, else 0
- $n_{i,6}$ 1 if the p end of i is rigidly fixed support, else 0
- $n_{i,7}$ 0 if the p end of i is rigidly fixed support, else 1
- $n_{i,8}$ Magnitude of load act on the p end of i in x axis
- $n_{i,9}$ Magnitude of load act on the p end of i in y axis
- $n_{i,10}$ x-axis deformation of the p end of i under $\mathrm{p_{S-}}$
- $n_{i,11}$ x-axis deformation of the p end of i under $\mathrm{p_{S+}}$
- $n_{i,12}$ y-axis deformation of the p end of i under $\mathrm{p_{S-}}$
- $n_{i,13}$ y-axis deformation of the p end of i under $\mathrm{p_{S+}}$
- $n_{i,14}$ p end's position in x-axis coordinate of i
- $n_{i,15}$ p end's position in y-axis coordinate of i
- $n_{i,16}$ 1 if the q end of i is rigidly fixed support, else 0
- $n_{i,17}$ 0 if the q end of i is rigidly fixed support, else 1
- $n_{i,18}$ Magnitude of load act on the q end of i in x axis
- $n_{i,19}$ Magnitude of load act on the q end of i in y axis
- $n_{i,20}$ x-axis deformation of the q end of i under $\mathrm{p_{S-}}$
- $n_{i,21}$ x-axis deformation of the q end of i under $\mathrm{p_{S+}}$
- $n_{i,22}$ y-axis deformation of the q end of i under $\mathrm{p_{S-}}$
- $n_{i,23}$ y-axis deformation of the q end of i under $\mathrm{p_{S+}}$
- $n_{i,24}$ q end's position in x-axis coordinate of i
- $n_{i,25}$ q end's position in y-axis coordinate of i
- $n_{i,26}$ Length of i
- $n_{i,27}$ Cross-sectional area of i
- $n_{i,28}$ Second moment area of i
- $n_{i,29}$ Internal stress in i from $\mathrm{p_{S-}}$
- $n_{i,30}$ Internal stress in i from $\mathrm{p_{S+}}$
- $n_{i,31}$ Internal stress in i from $\mathrm{p_L}$

Note that the values of $n_{i,1}$, $n_{i,3-5}$, and $n_{i,29-31}$ are set as 0 if the brace *i* has not been added to the structure. Both ends of beam and column elements are rigidly fixed supports while those of braces are hinge supports.

5.2 Action

The agent can choose an action of increasing the index (i.e., section) of the existing element or adding a brace. In the case of adding braces, no brace exists in the initial frame, and the number of domains with braces on each story can be only half of the total number of domains on that story. Furthermore, once the braces are added, their domain or type cannot be modified.

The action is interpreted from the output of the agent $\boldsymbol{\pi} \in \mathbb{R}^{n\times 1}$ and the vector that represents feasible action is represented as $\mathbf{g} \in \mathbb{R}^{n\times 1}$ in which the entry g_i is 1 if it is possible to increase the index (i.e., increasing section of an existing element or adding a brace) of the element i and is 0 otherwise. The action of the agent is the Hadamard product of policy function and $\mathbf{g}$ is as follows:

$$\mathbf{g} \circ \boldsymbol{\pi} = \boldsymbol{\pi}' \tag{14}$$

This way, the probability to select an infeasible action exactly becomes 0. Note that if the action of one of the braces that make up K-Type or V-Type is selected, the other member will also be changed to the same index.

If the *i*th component of $\boldsymbol{\pi}'$ has the maximum value among all components, then the section index of element *i* is increased by 1, and its section properties are changed accordingly. In order to simplify the structural design problem, the following rules are also applied:

1. Beams on the same floor shall have the same index. If a beam's section index is increased, those of other beams on the same floor will also be increased.
2. Upper column shall have a smaller cross-sectional area than its lower columns. If a column has a larger section index than its lower columns, the section index of the lower columns will be increased.

5.3 GCN-DDPG Agent

The agent's policy and value functions are made of multiple connected GCN layers defined by Eqs. $(15a) - (15c)$ where the Rectified Linear Unit (ReLU)[36] transforms the GCN's output to increase the robustness of the model, while the Sigmoid activation function[37] obtains probability-based output for the policy function (i.e., probability of taking action A_t^i in a state S_t).

$$\mu(\mathbf{V}, \widetilde{\mathbf{M}}) = \mathrm{ReLU}(\widetilde{\mathbf{M}}\mathbf{V}\mathbf{w}_\mu) \tag{15a}$$

$$\mathrm{iter}^{\phi}[\mu(\mathbf{V}, \widetilde{\mathbf{M}})] = \underbrace{\mu(\widetilde{\mathbf{M}}(\mu(\widetilde{\mathbf{M}}(\dots)\mathbf{w}_\mu))\mathbf{w}_\mu)}_{\phi \text{ times}} \tag{15b}$$

$$\sigma(\mathbf{V}, \widetilde{\mathbf{M}}) = \mathrm{Sigmoid}(\widetilde{\mathbf{M}}\mathbf{V}\mathbf{w}_\sigma) \tag{15c}$$

where $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$, ϕ in $\mathrm{iter}^{\phi}[\,]$ indicates the number of computing loops, $\mathrm{Sigmoid}(\cdot) = 1/(1 + \mathrm{e}^{-(\cdot)})$, and $\mathbf{V}$ denotes a node feature matrix or an output from a prior GCN layer. Equations $(15a)$ and $(15b)$ represent a GCN layer and multiple computing loops using the same GCN layer with ReLU activation function, respectively. Equation $(15c)$ represents a GCN layer with Sigmoid activation function.

In the value function, two matrix operations are used for transforming the output matrix of the last GCN layer into a scalar value representing the estimation of accumulated reward. The first operation is a global sum pooling operation (GSP: $\mathbb{R}^{n\times u} \to \mathbb{R}^{1\times u}$)[38] which transforms an output matrix of the last GCN layer into a vector by summing up all entries in each column of the output matrix. The GSP operation to transform a matrix $\mathbf{V}$ into a vector is represented as

$$\mathrm{GSP}(\mathbf{V}) = \left[\sum_{i=1}^{n} v_{i,1} \quad \cdots \quad \sum_{i=1}^{n} v_{i,u}\right] \in \mathbb{R}^{1\times u} \tag{16}$$

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 an NN which are approximation functions with adjustable weight parameters and activation functions. Equation (17) represents an NN used in this paper for computing the accumulated reward from the vector output of the GSP operation $\mathrm{GSP}(\mathbf{V})$. In NN, $\boldsymbol{\mathcal{W}}_1$, $\boldsymbol{\mathcal{W}}_2$, and $\boldsymbol{\mathcal{W}}_{\mathrm{out}}$ denote adjustable internal weight matrices while $\boldsymbol{\mathcal{B}}_1$, $\boldsymbol{\mathcal{B}}_2$, and $\boldsymbol{\mathcal{B}}_{\mathrm{out}}$ denote adjustable internal bias

vectors in each NN layer.

$$\mathrm{f_{NN}}\big(\mathrm{GSP}(\mathbf{V})\big) = \boldsymbol{\mathcal{W}}_{\mathrm{out}}\Big(\mathrm{ReLU}\big(\boldsymbol{\mathcal{W}}_2\big(\mathrm{ReLU}(\boldsymbol{\mathcal{W}}_1\mathrm{GSP}(\mathbf{V})^{\mathrm{T}} + \boldsymbol{\mathcal{B}}_1)\big) + \boldsymbol{\mathcal{B}}_2\big)\Big) + \boldsymbol{\mathcal{B}}_{\mathrm{out}} \in \mathbb{R}^{1\times 1} \quad (17)$$

Table 1 shows the computation method of the policy and value functions of the GCN-DDPG agent in this paper. The left column shows the policy function's inputs, computation method using Eqs. $(15)-(17)$, and output while the right column shows those of the value function.

Table 1 Policy and value functions of the agent

Policy function $\boldsymbol{\pi}$	Value function Q
Inputs: $\mathbf{N}$, $\widetilde{\mathbf{M}}$	Inputs: $\mathbf{N}$, $\widetilde{\mathbf{M}}$, $\boldsymbol{\pi}$
Computation:	Computation:
$\mathbf{N}_1 = \mu(\mathbf{N}, \widetilde{\mathbf{M}})$	$\mathbf{N}_{1.1} = \mu(\mathbf{N}, \widetilde{\mathbf{M}})$
$\mathbf{N}_2 = \mathrm{iter}^2[\mu(\mathbf{N}_1, \widetilde{\mathbf{M}})]$	$\mathbf{N}_{1.2} = \mathrm{iter}^2[\mu(\mathbf{N}_{1.1}, \widetilde{\mathbf{M}})]$
$\boldsymbol{\pi} = \sigma(\mathbf{N}_2, \widetilde{\mathbf{M}})$	$\mathbf{N}_{2.1} = \mu(\boldsymbol{\pi}, \widetilde{\mathbf{M}})$
	$\mathbf{N}_{2.2} = \mathrm{iter}^2[\mu(\mathbf{N}_{2.1}, \widetilde{\mathbf{M}})]$
	$\mathbf{N}_2 = \mathbf{N}_{1.2} + \mathbf{N}_{2.2}$
	$\mathbf{N}_3 = \mu(\mathbf{N}_2, \widetilde{\mathbf{M}})$
	$\mathbf{N}_4 = \mathrm{GSP}(\mathbf{N}_3)$
	$\mathrm{Q} = \mathrm{f_{NN}}(\mathbf{N}_4)$
Output: $\boldsymbol{\pi} \in \mathbb{R}^{n\times 1}$	Output: $\mathrm{Q} \in \mathbb{R}^{1\times 1}$

5.4 Reward

In RL, a reward signal is used for training an agent. In this paper, the reward signal at step t is formulated based on aforementioned objective and constraint functions Eqs. $(7.1)-(7.6)$ so that the agent can adjust the structure to satisfy all the constraints while reducing the increment ΔV of the cost as

$$\mathrm{R}_{t+1} = -\Delta V \cdot [4 - (\Delta\sigma^{\mathrm{L}}_{\max} + \Delta\sigma^{\mathrm{S}}_{\max} + \Delta\theta_{\max} + \Delta\delta_{\max})] \quad (18)$$

where $\Delta\sigma^{\mathrm{L}}_{\max}$, $\Delta\sigma^{\mathrm{S}}_{\max}$, $\Delta\theta_{\max}$, and $\Delta\delta_{\max}$ denote reward components associated with internal stresses from *long-term loading*, internal stresses from *long and short-term loadings*, inter-story drift angle, and beam center deflection, respectively.

To compute $\Delta\sigma^{\mathrm{L}}_{\max}$ and $\Delta\sigma^{\mathrm{S}}_{\max}$, the ratio of internal stresses and their upper bounds of each member, computed from *long-term loading* and the combinations of *long-term* and *short-term loadings*, are aggregated into vectors associated with each load case. Note if the member is the brace that has ratio of buckling stress larger than the ratio of *short-term loadings*, the ratio of buckling stress will be used as the aggregated value instead. These vectors are then normalized using *p*-norm $(\|\cdot\|_p)$ which computes the *p*th root of the sum of the *p*-powers of the absolute values of the components computed as

$$\|\mathbf{x}\|_p = \sqrt[p]{\sum_{i=1}^{n} |x_i|^p} \quad (19)$$

where p is a parameter to be specified. Particularly, the *p*-norm for $p = 1$ and ∞ are expressed as

$$\|\mathbf{x}\|_1 = |x_1| + |x_2| + \cdots + |x_n| \quad (20.1)$$

$$\|\mathbf{x}\|_\infty = \max\{|x_1|, |x_2|, \ldots, |x_n|\} \quad (20.2)$$

Thus, *p*-norm allows scalar representation of a vector as an intermediate value between the maximum and average values. Since maximum stress usually depends on the section sizes of multiple members, it often happens that reduction of the maximum stress leads to an increase of the second one, and accordingly, an increase of the value of maximum stress. Therefore, the *p*-norm is used for computing reward components associated with stress constraints to incorporate the effect of section sizes of several members with large stresses. On the contrary, the maximum values of inter-story drift angle and beam center deflection mainly depend on the sizes of the specific members, and the constraints can be satisfied by simply reducing the maximum values. Therefore, the *p*-norm is not used for computing reward components associated with these constraints. Reward components associated with each constraint are formulated as

$$\Delta\sigma^{\mathrm{L}}_{\max} = \begin{cases} \dfrac{\|\boldsymbol{\sigma}^{\mathrm{L}}\|_p^t - \|\boldsymbol{\sigma}^{\mathrm{L}}\|_p^{t+1}}{\|\boldsymbol{\sigma}^{\mathrm{L}}\|_p^{t+1}} & (\text{if } \sigma^{\mathrm{L},t}_{\max} > \hat{\sigma}^{\mathrm{L}}) \\ 0 & (\text{else}) \end{cases} \quad (21.1)$$

$$\Delta\sigma^{\mathrm{S}}_{\max} = \begin{cases} \dfrac{\|\boldsymbol{\sigma}^{\mathrm{S}}\|_p^t - \|\boldsymbol{\sigma}^{\mathrm{S}}\|_p^{t+1}}{\|\boldsymbol{\sigma}^{\mathrm{S}}\|_p^{t+1}} & (\text{if } \sigma^{\mathrm{S},t}_{\max} > \hat{\sigma}^{\mathrm{S}} \text{ or } \lambda^t_{\max} > 1) \\ 0 & (\text{else}) \end{cases} \quad (21.2)$$

$$\Delta\theta_{\max} = \begin{cases} \dfrac{\theta^t_{\max} - \theta^{t+1}_{\max}}{\theta^{t+1}_{\max}} & (\text{if } \theta^t_{\max} > \hat{\theta}) \\ 0 & (\text{else}) \end{cases} \quad (21.3)$$

$$\Delta\delta_{\max} = \begin{cases} \dfrac{\delta^t_{\max} - \delta^{t+1}_{\max}}{\delta^{t+1}_{\max}} & (\text{if } \delta^t_{\max} > 1) \\ 0 & (\text{else}) \end{cases} \quad (21.4)$$

6. NUMERICAL EXAMPLES

6.1 General experiment setting and structural model

Structural elements used in this experiment have Young's modulus, maximum *short-term*, and maximum *long-term* stress limits of 200 kN/mm^2, 235 N/mm^2, and 235/1.5 = 156.7 N/mm^2, respectively. Lists of section properties for columns, beams, and braces are shown in Tables 2, 3, and 4, respectively. The structure is subjected to the floor-weight of 6700 N/m^2 (i.e., long-term floor load) applied on each beam using the floor depth of 6 m and the load computed as described in Sec. 2.2 using a floor-weight of 5700 N/m^2 (i.e., seismic floor load) where the base shear coefficient C_0, the specific period T_c, and seismic area coefficient Z_0 are 0.2, 0.6, and 1.0, respectively. The cost coefficient *B* for braces is 1.5, and the parameter *p* for the *p*-norm is 10.

The program is implemented using Python 3.6 environment. A PC with a CPU of Intel Core i9-11900 (2.5 GHz, 8 cores) and a GPU of Nvidia GeForce RTX3060 12GB is used for computation.

6.2 Training phase

The term *episode* is defined as the sequence of optimization from the initial state to the terminal state by the agent. In each episode of the *training phase*, the agent is trained to optimize the section sizes of beams and columns as well as the placements and section sizes of braces of 3-span 3-story steel frames with three predetermined span lengths and story heights as indicated by S1, S2, and S3 in Table 5. At the beginning of each episode, the frame

Table 2 List of column sections

Index	Box section	A (cm^2)	I (cm^4)	Z (cm^3)
0	200×200×9	66	3920	392
1	300×300×12	133	18100	1200
2	400×400×16	237	57100	2850
3	500×500×22	404	150000	6010
4	600×600×32	727	392000	13100
5	700×700×35	855	637000	18210
6	800×800×40	1216	1173000	29360

Table 3 List of beam sections

Index	H section	A (cm^2)	I (cm^4)	Z (cm^3)
0	300×200×56	71	11100	756
1	400×300×105	133	37900	1940
2	500×300×125	159	68900	2820
3	600×300×147	187	114000	3890
4	700×300×182	232	197000	5640
5	800×300×188	234	248000	6270
6	900×300×283	360	491000	10800

Table 4 List of brace sections

Index	Diameter (mm)	Thickness (mm)	A (cm^2)	I (cm^4)	Z (cm^3)
0			no brace		
1		6.6	54.08	4.60×10^3	344
2		8.0	65.19	5.49×10^3	411
3	267.4	9.3	75.41	6.29×10^3	470
4		12.7	101.6	8.26×10^3	618
5		15.1	119.7	9.56×10^3	715

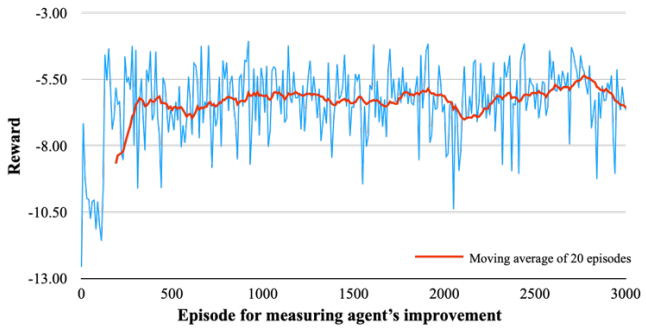


Fig.5 Training phase result (reward)

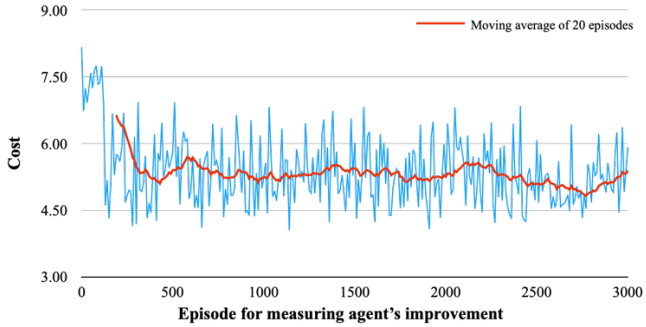


Fig.6 Training phase result (cost)

without braces is initialized using smallest beam and column sections. In order to measure the performance of the agent, 4-span 4-story frames with span lengths of [4.0, 6.0, 6.0, 4.0] (m) and story heights of [4.0, 3.0, 3.0, 3.0] (m), are used for measuring improvement of the reward during the training every 10 episodes.

At each step, the agent's action determines the addition of a brace or increasing the size of existing beam, column, or brace. The surrogate policy and value functions are adjusted by the Adam optimizer using the mini-batch size of 32 and the learning rates of 10^{-3} and 10^{-4} for the policy function and value functions, respectively. In the value function, each layer in NN has 200 cells and the loss function for training NN is the mean squared error loss. Surrogate functions' parameters ($\boldsymbol{\theta'}_1$, $\boldsymbol{\theta'}_2$) are updated to the real policy and value functions' parameters ($\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$) every 100 steps with $\tau = 0.05$. The agent is trained for 3,000 episodes where each episode is terminated when all constraints are satisfied.

The total number of structural analyses in the training phase is 134,995. Figures 5 and 6 show the reward and the final cost of the structure obtained by the agent, respectively, being indicators of the agent's performance during the training phase, where the horizontal axis represents the number of trained episodes and the thick red line in each figure shows the moving average of 20 episodes. According to these figures, the agent has increased the obtained reward, reduced the cost of the structure, and maintained its performance throughout the training, which indicates the learning ability of the agent. Note that since the structural types S1, S2, and S3 are used for training and the performance is measured using a 4-span 4-story frame, the agent might not be able to keep on improving the obtained reward using this structure resulting in the fluctuation of the reward and cost.

6.3 Test phase

In the test phase, the agent saved from the training phase is applied to the frames in Table 6 to verify the generality of the agent, where the material property and section lists are the same as those in the training phase. To prevent the agent to choose unnecessary actions, we introduce another agent output modification vector $\mathbf{h} \in \mathbb{R}^{n\times1}$ that represents the beam or column that satisfies all constraints, including *long-term loading* stress, *long-term and short-term loading* stress, inter-story drift, and deflection constraints. The entry h_i is 0 if the element i is a column or beam that already satisfies every constraint. Therefore, the vector $\mathbf{h}$ prevents the agent from choosing base beams when it is stiff enough, and also other beams and columns, that already satisfies all constraints by assigning 0 for the value in $\mathbf{h}$. The action of agent is modified using the Hadamard product of policy function, using both vectors that represent feasible action $\mathbf{g}$ and $\mathbf{h}$, as follows:

$$\boldsymbol{\pi}' = \mathbf{g} \circ \mathbf{h} \circ \boldsymbol{\pi} \tag{22}$$

The action interpretation of increasing the section index of element and the rules are the same as those of the training phase.

The trained agent is applied to Structures A and B 50 times for each. Table 7 shows the minimum, mean, and standard deviation (Std.) of the objective function and the total number of structural analyses of each structure. Note that the agent can be applied to larger numbers of structures not included in this research as well. Therefore, the number of structural analyses in the training phase is not included in the comparison. Figures 7 and 8 show the

best solutions (Structure A, Structure B) obtained in the test phase where the thickness of line and ▲ indicates rigid support. Numbers on the lines represent section size of column (black), beam (grey), and brace (red).

6.4 Comparison of computational cost and performance with GA

GA is used for evaluating the performance of the proposed RL method utilizing the similar computational cost of structural analyses. GA is a meta-heuristic optimization method where design variables are represented as genes. In each iteration (i.e., generation), candidate solutions (i.e., individuals) are generated by randomly mixing genes (i.e., crossover) from the previous iteration's individuals with some random gene modifications (i.e., mutation). The best fit individuals in view of the objective function are kept for the next generation (i.e., selection).

In this research, each gene has an integer value of [0,9] so that they can be randomly mixed and mutated. The indices $\{J_1, J_2, \dots, J_{N_f}\}$ of column sections in Table 2 along the same vertical axis are defined by N_f genes $\{G_1, G_2, \dots, G_{N_f}\}$. If G_1 is larger than the largest column index (6 in Table 2), J_1 is equal to the largest index. Since the upper column cannot have a

Table 5 Structural configuration for training phase

Structural name	S1	S2	S3
Number of spans	3	3	3
Span length	[4,4,4] (m)	[8,8,8] (m)	[6,6,6] (m)
Number of floors	3	3	3
Floor height	[4,4,4] (m)	[3,3,3] (m)	[4,4,4] (m)

Table 6 Structural configuration for test phase

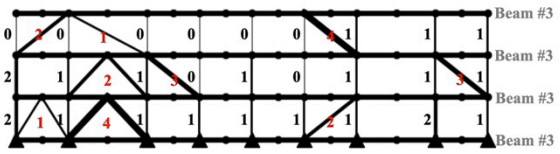
Name	Structure A	Structure B
Number of spans	8	3
Span length	[5.0, 7.5, 5.0, 5.0, 5.0, 5.0, 7.5, 5.0] (m)	[5.0, 5.0, 5.0] (m)
Number of stories	3	7
Story height	[4.0, 4.0, 4.0] (m)	[4.0, 4.0, 3.5, 3.5, 3.5, 3.5, 3.5] (m)

Table 7 RL results in test phase (50 trials)

Name	Structure A	Structure B
Minimum	5.61	5.35
Mean	6.22	5.98
Std.	0.27	0.26
Number of structural analyses	3191	3812

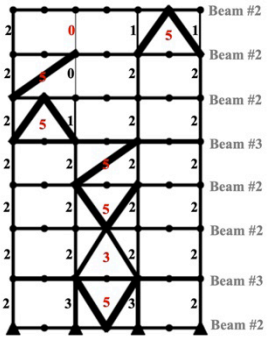
Table 8 GA results (5 random seeds)

Name	Structure A	Structure B
Minimum	6.90	7.34
Mean	7.28	8.05
Std.	0.46	0.58
Population	50	100
Generation	80	50
Number of structural analyses	4000×5	5000×5



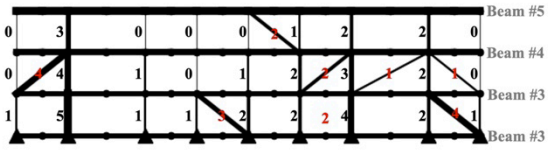
Final cost: 5.61
Maximum short-term stress constraint: 1.00
Maximum long-term stress constraint: 0.85
Maximum deformation constraint: 0.10
Maximum inter-story drift constraint: 0.13

Fig.7 RL best result: Structure A



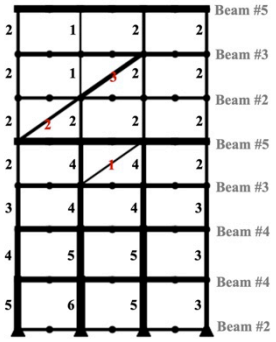
Final cost: 5.35
Maximum short-term stress constraint: 0.97
Maximum long-term stress constraint: 0.82
Maximum deformation constraint: 0.15
Maximum inter-story drift constraint: 0.19

Fig.8 RL best result: Structure B



Final cost: 6.90
Maximum short-term stress constraint: 0.98
Maximum long-term stress constraint: 0.89
Maximum deformation constraint: 0.09
Maximum inter-story drift constraint: 0.11

Fig.9 GA best result: Structure A



Final Cost: 7.34
Maximum Short-term stress constraint: 0.94
Maximum Long-term stress constraint: 0.59
Maximum Deformation constraint: 0.08
Maximum Inter-story drift constraint: 0.19

Fig.10 GA best result: Structure B

larger index than the lower ones, $J_{i\in\{2,\ldots,N_f\}}$ is equal to J_{i-1} if $G_i \in [0,4]$, whereas $J_{i-1} = J_i - 1$ if $G_i \in [5,9]$ (i.e., the gene G_1 indicates the column index while the other genes indicate the reduction of the column indices). Beam sections on the same floor are represented by a single gene; the index of the beam section from Table 3. If the gene has a larger value than the largest beam index, the largest beam index is assigned to the corresponding beam.

The brace type and brace section in each brace domain are represented by two genes where the first gene with the value of {0,1}, {2,3}, {4,5}, {6,7}, and {8,9} represent no brace, right diagonal brace, left diagonal brace, K-type, and V- type, respectively. The second gene with the value of {0,1}, {2,3}, {4,5}, {6,7}, and {8,9} represent indices 1, 2, 3, 4, 5, respectively, of the brace section in Table 4. Also, similar to the proposed RL method, in GA's solutions, the number of domains with braces in each story can be only half of the number of brace domains in that story, which is equal to the number of spans for a regular frame.

In order to obtain solutions that satisfy all constraints with small cost, binary value indicators c_S, c_L, c_θ, and c_δ are introduced for the constraints on maximum *short-term* and buckling stress, *long-term* stress, beam deflection, and inter-story drift, respectively. They The indicator is equal to 1 if the constraint is violated, and 0 if satisfied. The objective function of GA to be maximized is as follows:

$$G(\mathbf{X}) = -V(\mathbf{X}) - w_0\left(c_S\frac{w_1\sigma_{\max}^S}{\hat{\sigma}^S} + c_L\frac{\sigma_{\max}^L}{\hat{\sigma}^L} + c_\theta\frac{\theta_{\max}}{\hat{\theta}} + c_\delta\delta_{\max}\right) \quad (23)$$

where w_0 and w_1 are the weights of the penalty, which prevent the GA to generate solutions that violate the constraints, with the values of 5 and 3, respectively. These values are decided after some trials and errors.

A Python library DEAP[39] is used to implement the GA program. The numbers of populations and generations are specified so that the total number of analyses becomes similar to the total number of analyses used in RL. Since the solution of GA depends on the random seed, we conduct five GA experiments with different random seeds on Structures A and B in Table 6. Table 8 shows the minimum, mean, standard deviation (Std.) of the objective function, population, generation, and the number of structural analyses of each structure using GA. The best solutions of Structures A and B obtained by GA are shown in Figures 9 and 10.

It is observed from Tables 7–8 that the RL agent can obtain more optimal results compared to GA for both test structures shown by the minimum of the objective function. RL agent's results from different trails have similar value of the objective function shown in the Std. value. In computational aspect, the trained RL agent utilizes less structural analyses compared to the GA. Figures 7–10, where the width of each member defined the section index of that member type (i.e., column, beam, or brace), show differences between results obtained by RL and GA. In RL agent's results, beams and columns have smaller cross-sectional sizes compared to those of GA while both RL's results contain more braces than GA's results.

In this research, the brace's cross-sectional areas are smaller than those of beams and columns and increasing sections of beams or columns may lead to the increase of other beams on the same floor or columns on the lower floor as well which eventually lead to a large value of the objective function. From the agent's viewpoint, placing braces could yield a higher reward (lower cost), which is computed from the value of the objective function, in each optimization step (i.e., placing braces, increasing sections), compared to increasing sections of beams or columns. Therefore, the RL agent shows strategies of preferentially placing braces rather than enlarging costly beams and columns. In both test structures, short-term and long-term stress constraints are the most critical constraints in this experiment. In Figures 7–10, maximum short-term stress, maximum long-term stress constraint, maximum deformation, and maximum inter-story drift constraints in RL's results are higher than those of GA except for the maximum long-term stress constraint in structure A. This implied that the trained RL agent yields more efficient solutions that are located near the boundary of the feasible region.

7. CONCLUSIONS

A method for topology optimization of braced steel frames using DDPG and GCN has been proposed for determining brace locations and sizes of structural elements including beams, columns, and braces. The optimization problem is to minimize the cost computed from the volume and type of structural elements under constraints on stress, deformation, and inter-story drift. The problem is a combinatorial problem that is difficult to obtain the optimal solution even using heuristic approaches which requires large computational cost. This research shows that the proposed ML method can find more optimal solutions and requires less computational cost compared to GA.

This paper proposes a topology optimization method that combines graph representation of structural elements and DDPG. The graph representation method that interprets structural elements into graph nodes is utilized for generating state data in the MDP framework. Using this representation, modification of the element can be obtained directly from the RL agent with graph neural networks. The DDPG agent with policy and value functions introducing GCNs is used to determine how the structural design should be adjusted. The agent is trained to modify structures until all constraints are satisfied using the reward computed from both cost and changes of constraints. Results show that the agent can improve its performance by maximizing the accumulative reward.

The numerical examples of 3-story 8-span and 7-story 3-span frames, where short-term and long-term stress constraints are the most critical, show that the trained agent can be applied to structures that differ from those used for training. The trained agent can obtain solutions utilizing a lower cost compared to those of GA by utilizing slenderer beams and columns with more braces compared to solutions from GA which have thicker beams and columns with fewer braces. Note that the reward in this experiment is partly determined from the lists of cross-sections for beams, columns, and braces (i.e., structural volume). Therefore, if the lists of cross sections or the reward function are changed, the RL agent could modify structures differently.

The agent can be used for finding approximate optimal structural configurations in a feasible time and is applicable to problems with larger structural lists or more degrees of freedom than the numerical examples. Application to the three-dimensional model will be investigated in our future study.

Acknowledgement

This study is supported by MEXT scholarship (grant number 180136) and JSPS KAKENHI (grant number JP 20H04467, JP 21K20461, and JP 21K04337).

Disclosure

The authors declare that they have no known competing interests or personal relationships that could have influenced the work reported in this research paper.

REFERENCES

1) P. W. Christensen and A. Klarbring: An introduction to structural optimization, *Solid mechanics and its applications*, Vol.153, Springer, Dordrecht, 2009.
2) M. Ohsaki and C. Swan: Topology and geometry optimization of trusses and frames, *Recent Advances in Optimal Structural Design*, ASCE, 2002.
3) J. H. Holland: Adaptation in natural and artificial systems, *An introductory analysis with applications to biology, control, and artificial intelligence*. U Michigan Press, 1975.
4) P. J. M. van Laarhoven and E. H. L. Aarts: *Simulated Annealing: Theory and Applications*, Reidel, Dordrecht, 1987.
5) W. M. Jenkins: Towards structural optimization via the genetic algorithm, *Comput. Struct.*, Vol.40, pp.1321-1327, 1991.
6) M. Ohsaki: Genetic algorithm for topology optimization of trusses, *Comput. Struct.*, Vol.57(2), pp.219-225, 1995.
7) B. V. H. Topping, A. I. Khan and J. P. Leite: Topological design of truss structures using simulated annealing, *Struct. Engng. Rev.*, Vol.8, pp.301-314, 1996.
8) H. Tagawa and M. Ohsaki: A continuous topology transition model for shape optimization of plane trusses with uniform cross-sectional area, *In Proc. 3rd World Congress of Structural and Multidisciplinary Optimization (WCSMO3)*, pp. 254-2565, 1999.
9) R. D. Vanluchene and R. Sun: Neural networks in structural engineering, *Computer-Aided Civil Infrastructure Eng.*, Vol.5, pp. 207–215, 1990.
10) H. Zheng: Form finding and evaluating through machine learning: the prediction of personal design preference in polyhedral structures, *in The International Conference on Computational Design and Robotic Fabrication*, Singapore Springer, pp. 169–178, 2019.
11) G. Mirra and A. Pugnale: Comparison between human-defined and ai-generated design spaces for the optimisation of shell structures, *Structures*, Vol.34, pp. 2950–2961, 2021.
12) P. D. Kingma and M. Welling: An introduction to variational autoencoders, *Foundations and Trends in Machine Learning*, Vol.12, No.4, pp. 307-392, 2019.
13) T. Tamura, M. Ohsaki and J. Takagi: Machine learning for combinatorial optimization of brace placement of steel frames, *Jpn. Architect. Rev.*, Vol.1, pp.419–430, 2018.
14) K. Sakaguchi, M. Ohsaki and T. Kimura: Machine learning for extracting features of approximate optimal brace locations for steel frames, *Frontiers in Built Environment*, 2020.
15) T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra: Continuous control with deep reinforcement learning, *in ICLR*, 2016.
16) F. Rosenblatt: The perceptron: a probabilistic model for information storage and organization in the brain, *Psycological review*, Vol.65(6), pp.386-408, 1958.
17) A. G. Ivakhnenko: The group method of data handling – a rival of the of stochastic approximation, *Soviet Automatic Control*, Vol.13(3), pp.43-55, 1968.
18) I. Goodfellow, Y. Bengio and A. Courville: *Deep learning*, The MIT Press, 2016.
19) D. K. Duvenaud, D. Maclaurin, J. Aguileraiparraguirre, R. Gomezbombarelli, T. D. Hirzel, A. Aspuruguzik and R. P. Adams: Convolutional networks on graphs for learning molecular fingerprints, *In Proceedings of NIPS*, pp.2224–2232, 2015.
20) D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega and P. Vandergheynst: The emerging field of signal processing on graphs: extending high-dimensional data analysis to networks and other irregular domains, *IEEE SPM*, Vol.30, pp.83–98, 2013.
21) C. Langenhan, M. Weber, M. Liwicki, F. Petzold and A. Dengel: Graph-based retrieval of building information models for supporting the early design stages, *Adv. Eng. Inform*., Vol.27, pp.413–426, 2013.
22) J. Abualdenien and A. Borrmann: PBG: A parametric building graph capturing and transferring detailing patterns of building models, *in Proc. of the CIB W78 Conference 2021 (Luxembourg: International Council for Research and Innovation in Building and Construction)*, 2021.
23) P. Vestartas: Design-to-fabrication workflow for raw-sawn-timber using joinery solver (LausanneSwitzerland: EPFL), *Ph.D. thesis*, 2021.
24) K. Hayashi and M. Ohsaki: Reinforcement learning and graph embedding for binary truss topology optimization under stress and displacement constraints, *Frontiers in Built Environment*, Vol.6, pp.59, 2021.
25) C. Kupwiwat, K. Hayashi and M. Ohsaki: Deep deterministic policy gradient and graph convolutional network for bracing direction optimization of grid shells, *Frontiers in Built Environment*, Sec. Computational Methods in Structural Engineering, 2022.
26) T. N. Kipf and M. Welling: Semi-supervised classification with graph convolutional networks, *in ICLR*, 2017.
27) R. S. Sutton and G. B. Andrew: *Reinforcement learning, an introduction*, The MIT Press, 1998.
28) R. Bellman: A markovian decision process, *Journal of Mathematics and Mechanics*, pp.679-684, 1957.
29) R. Bellman: The theory of dynamic programming, *Bull. Amer. Math. Soc.*, Vol.60, No.6, pp.503-515, 1954.
30) T. Haarnoja, A. Zhou, P. Abbeel and S. Levine: Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor, *Arxiv:1801.01290*, 2018.
31) H. Robbins and S. Monro: A stochastic approximation method, *Ann.*

Math. Statist., Vol.22, No.3, pp.400-407, 1951.

32) J. Kiefer and J. Wolfowitz: Stochastic estimation of the maximum of a regression function, *Ann. Math. Statist.*, Vol.23, No.3, pp.462-466, 1952.

33) S. Ruder: An overview of gradient descent optimization algorithms, *Arxiv:1609.04747*, 2016.

34) D. Kingma and J. Ba: Adam: a method for stochastic optimization, *in ICLR*, 2015.

35) G. E. Uhlenbeck and S. L. Ornstein: On the theory of the brownian motion, *Phys. Rev.*, Vol.36, Issue 5, pp.823-841, 1930.

36) V. Nair and G. E. Hinton: Rectified linear units improve restricted boltzmann machines, *Haifa*, pp. 807–814, 2010.

37) N. Chigozie, W. Ijomah, A. Gachagan and M. Stephen: Activation functions: comparison of trends in practice and research for deep learning, *Arxiv:1811.03378*, 2020.

38) S. Aich and I. Stavness: Global sum pooling: a generalization trick for object counting with small datasets of large images, *Arxiv:1805.11123*, 2019.

39) F. A. Fortin, F. M. De Rainville, M. A. Gardner, M. Parizeau and C. Gagné: DEAP: Evolutionary algorithms made easy, *Journal of Machine Learning Research*, *Machine Learning Open Source Software*, Vol.13, pp.2171-2175, 2012.