

Reinforcement Learning for Optimal Design of Skeletal Structures

Kazuki Hayashi

2021

Contents

List of Abbreviations	7
Nomenclature	9
1 Introduction	15
1.1 Background of overall study	15
1.1.1 Structural optimization	15
1.1.2 Machine learning for structural engineering	16
1.1.3 Optimization of trusses	20
1.1.4 Cross-section optimization of steel frames	22
1.2 Objective	23
1.3 Advantages of the proposed method	24
1.4 Thesis structure	24
2 Basics of reinforcement learning	25
2.1 Characteristic of reinforcement learning	25
2.2 Markov decision process	27
2.3 Reinforcement learning	29
2.3.1 Value function	29
2.3.2 Policy	31
2.3.3 Bellman equation	32
2.3.4 Dynamic programming and Monte Carlo methods	33
2.3.5 Temporal difference learning	34
2.4 Surrogate modelling	36
2.4.1 Curse of dimensionality	36
2.4.2 Neural network	36
2.4.3 Deep Q-network	39
2.5 Conclusion	40
3 Hybrid method of graph embedding and reinforcement learning for training an optimal design agent of skeletal structures	43
3.1 Conversion of an optimization problem into a reinforcement learning task	43
3.2 Edge features estimated by graph embedding	45
3.3 Q-learning using the embedded features	50
3.4 Training workflow	52
3.5 Conclusion	53

4	Case study 1: Topology optimization of trusses	55
4.1	Topology optimization problem of planar trusses considering stress constraint	55
4.2	Conversion to a reinforcement learning task	56
4.3	Training setting	57
4.4	Training result	58
4.4.1	Training history and performance for 4×4 -grid truss	58
4.4.2	Investigation of generalization performance 1: 3×2 -grid truss	70
4.4.3	Investigation of generalization performance 2: 6×6 -grid truss	80
4.4.4	Comparison of efficiency and accuracy with genetic algorithm	89
4.5	Generation of initial solutions by a trained agent for simultaneous optimization of geometry and topology of trusses using force density method	92
4.5.1	Force density method for obtaining nodal locations	92
4.5.2	Optimization problem	95
4.5.3	Sensitivity analysis	97
4.5.4	Workflow	98
4.5.5	Numerical examples	98
4.6	Conclusion	100
5	Case study 2: Cross-section optimization of steel frames	103
5.1	Discrete cross-section optimization problem of planar steel frames under constraints on stress, displacement and column over-strength factor	103
5.1.1	Allowable stress and displacement design	103
5.1.2	“Strong column-weak beam” design	106
5.1.3	Optimization problem	107
5.2	Conversion to a reinforcement learning task	109
5.3	Training setting	111
5.4	Training result	112
5.4.1	Training history and performance for 3×5 -grid frame	112
5.4.2	Investigation of generalization performance 1: 5×3 -grid frame	123
5.4.3	Investigation of generalization performance 2: 4×6 -grid frame	125
5.4.4	Comparison of efficiency and accuracy with particle swarm optimization	128
5.5	Conclusion	130
6	Conclusion	133
6.1	Summary	133
6.2	Future outlook	135
A	GPU implementation for accelerating the training	137
	Bibliography	139

List of presentations related to this study	151
List of other presentations	153
Acknowledgement	155

List of Abbreviations

AEC	Architecture, Engineering and Construction
AI	Artificial Intelligence
API	Application Programming Interface
BIM	Building Information Modeling
CAD	Computer-Aided Design
CNN	Convolutional Neural Network
COF	Column Overstrength Factor
CPU	Central Processing Unit
DOF	Degrees Of Freedom
DP	Dynamic Programming
DQN	Deep-Q Network
FDM	Force Density Method
GA	Genetic Algorithm
GE	Graph Embedding
GPGPU	General-Purpose computing on GPU
GPU	Graphics Processing Unit
GS	Ground Structure
GUI	Graphical User Interface
IGPU	Integrated GPU
LP	Linear Programming
MC	Monte Carlo
MDP	Markov Decision Process
ML	Machine Learning
NLP	Non-Linear Programming
NN	Neural Network
PSO	Particle Swarm Optimization
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RL+GE	Proposed hybrid method of RL and GE
RMSProp	Root Mean Square Propagation
SA	Simulated Annealing
SGD	Stochastic Gradient Descent
SQP	Sequential Quadratic Programming
SVM	Support Vector Machine
TD	Temporal Difference

Nomenclature

a	Action
a_t	Action taken at step t
$a^{(i)}$	Action indexed as i
\bar{A}	Cross-sectional area assigned to existing truss members
A_i	Cross-sectional area of member i
A_i^s	Distribution of the seismic shear force coefficient along with the build height
\mathbf{A}	Cross-sectional area vector
$b_k^{(i)}$	Bias of k -th unit in layer i of the NN
b_i	Penalty coefficient for the violation of i -th constraint in metaheuristic algorithms
c	Smaller positive number for function smoothing in the FDM
C_B	Base shear coefficient
C_i	Penalty function with respect to the i -th constraint
C_{ij}	(i, j) element of the connectivity matrix
C_i^s	Shear force coefficient of story i
\mathbf{C}	Connectivity matrix for a directed graph
\mathbf{C}_A	Non-oriented connectivity matrix
\mathbf{C}_1	Matrix to identify the nodes that each member leaves
\mathbf{C}_2	Matrix to identify the nodes that each member enters
$d_{i,j}$	Elongation of member i in load case j
E	Elastic modulus of the structural material
\mathbf{E}	Force density matrix
$\tilde{\mathbf{E}}$	Matrix assembling the force density matrices in x -, y - and z -directions
$\tilde{\mathbf{E}}_{\text{fix}}$	Sub-matrix of $\tilde{\mathbf{E}}$ concerning indices of fixed coordinates
$\tilde{\mathbf{E}}_{\text{free}}$	Sub-matrix of $\tilde{\mathbf{E}}$ concerning indices of free coordinates
$\tilde{\mathbf{E}}_{\text{link}}$	Sub-matrix of $\tilde{\mathbf{E}}$ concerning the other indices
f_b	Allowable bending stress
f_c	Allowable compressive stress
f_t	Allowable tensile stress
F_d	Design strength of the material
F_{yb}	Design strength of the beam element material
F_{yc}	Design strength of the column element material
G_1	Clipping and scaling function of reward setting 1 in frame optimization
G_2^+	Clipping and scaling function of reward setting 2 for increasing cross-sections
\mathbf{h}_1	Aggregated data of a member with respect to member inputs

\mathbf{h}_2	Aggregated data of a member with respect to node inputs
$\mathbf{h}_3^{(t_u)}$	Weighted self-member feature of a member at step t_u
$\mathbf{h}_4^{(t_u)}$	Aggregated data of a member with respect to member features at step t_u
$\hat{\mathbf{h}}_1$	Aggregated data of all members with respect to member inputs
$\hat{\mathbf{h}}_2$	Aggregated data of all members with respect to node inputs
$\hat{\mathbf{h}}_3^{(t_u)}$	Weighted self-member feature of all members at step t_u
$\hat{\mathbf{h}}_4^{(t_u)}$	Aggregated data of all members with respect to member features at step t_u
H	Height of the structure
I_i	Moment of inertia of member i
I_i^y	Moment of inertia of member i along the weak axis
I_i^z	Moment of inertia of member i along the strong axis
J_i	Cross-section index chosen from a list of standard cross-sections
\mathbf{J}	Set of cross-section indices chosen from a list of standard cross-sections
\mathbf{K}	Global stiffness matrix of the structure
\mathbf{K}_i	Local stiffness matrix with respect to member i
L	Loss function to be minimized for training parameters
L_i	Length of member i
L_e	Length of the eliminated member
M_{pb}	Full plastic moment of beams
M_{pc}	Full plastic moment of columns
$M_{i,j}$	Bending moment at the j -th tip of member i
n_a	Number of actions
n_d	Number of DOFs of the structure
n_e	Number of members whose cross-sections have been changed by the action
n_{ep}	Number of training episodes
n_f	Number of the size of each member feature
n_{fm}	Number of the size of each member input
n_{fn}	Number of the size of each node input
n_l	Number of layers in the NN
n_L	Number of loading conditions in the structural analysis
n_m	Number of members
n_n	Number of nodes
n_o	Number of outputs of the NN
n_p	Number of particles/solutions generated at each step of metaheuristic algorithms
n_s	Number of states
n_{sf}	Number of stories of the building frame
n_{sy}	Synchronization frequency of updating the trainable parameters
n_t	Stopping criterion of metaheuristic algorithms
$n_{u,i}$	Number of units in layer i of the NN
N_i	Axial force of member i
o_k	Linear combination of inputs of the k -th unit in the NN
\mathbf{p}	Nodal load vector in the stiffness method
$\bar{\mathbf{p}}_i$	Load vector assigned to member i
\mathbf{p}_j	Load vector with respect to load case j

\bar{P}_i	Load value to be specified in the FDM
$P_{ss'}^a$	Probability of transitioning from state s to s' by taking action a
\mathbf{P}	Nodal load vector in the FDM
\mathbf{P}_{fix}	Nodal load vector corresponding to fixed nodal coordinates \mathbf{X}_{fix}
\mathbf{P}_{free}	Nodal load vector corresponding to free nodal coordinates \mathbf{X}_{free}
\mathbf{P}_x	x -directional nodal load vector
\mathbf{P}_y	y -directional nodal load vector
\mathbf{P}_z	z -directional nodal load vector
q_i	Force density of member i
$q_{i,x}$	x -directional component of the force density of member i
$q_{i,y}$	y -directional component of the force density of member i
$q_{i,z}$	z -directional component of the force density of member i
\mathbf{q}	Force density vector
Q	Action value
Q^π	Action value based on policy π
Q^*	Optimal action value
Q_i	Shear force of i -th story
r	Reward
r_t	Reward obtained at step t
$r_{ss'}^a$	Reward obtained when transitioning from state s to s' by action a
R_i	i -th reaction force in a structural system
R_t	Return at step t
R^s	Vibration characteristics of the building
\mathbf{R}	Reaction force vector
s	State
s_t	State at step t
s'	Next state
$s^{(i)}$	State indexed as i
t	Step in the episode
t_c	Elapsed CPU time
t_u	Step number of updating the member feature
T	Terminal step of the episode
T_f	Primary natural period
T_u	Terminal step number of updating the member feature
\bar{u}	Upper bound displacement
$u_{i,j}$	Nodal displacement of DOF i for load case j
\mathbf{u}	Nodal displacement vector
\mathbf{u}_j	Nodal displacement vector with respect to load case j
$\bar{\mathbf{u}}_i$	Nodal displacement vector of member i
\mathbf{v}_k	Input from node k in the graph
$\mathbf{v}_{i,j}$	Input from the j -th end of member i
$\hat{\mathbf{v}}$	Set of node inputs in the current state of the graph
$\hat{\mathbf{v}}'$	Set of node inputs in the next state of the graph
V	State value

V_{init}	Initial structural volume
V_s	Structural volume
V^π	State value based on policy π
V^*	Optimal state value
\bar{V}	Structural volume after applying a trained RL for the FDM optimization
$w_{k,j}^{(i)}$	j -th weight of k -th unit in layer i of the NN
\mathbf{w}_i	Input from member i in the graph
$\hat{\mathbf{w}}$	Set of member inputs in the current state of the graph
$\hat{\mathbf{w}}'$	Set of member inputs in the next state of the graph
W_i	Weight of story i
W_T	Total structural weight
\mathbf{x}	Nodal x coordinates
\mathbf{x}_{fix}	Nodal x coordinates fixed to move
\mathbf{x}_{free}	Nodal x coordinates free to move
\mathbf{X}_{fix}	Nodal coordinates fixed to move
\mathbf{X}_{free}	Nodal coordinates free to move
\mathbf{X}_k	Nodal coordinate vector of node k
$y_k^{(i)}$	Output of the k -th unit in layer i of the NN
\mathbf{y}	Nodal y coordinates
\mathbf{y}_{fix}	Nodal y coordinates fixed to move
\mathbf{y}_{free}	Nodal y coordinates free to move
z_k	Final output of the k -th unit of the NN
\bar{z}_k	Target value of k -th unit's output of the NN
\mathbf{z}	Nodal z coordinates
\mathbf{z}_{fix}	Nodal z coordinates fixed to move
\mathbf{z}_{free}	Nodal z coordinates free to move
Z	Coefficient to scale the shear force coefficient C_i^s
Z_i	Section modulus of member i
Z_i^y	Section modulus of member i along the weak axis
Z_i^z	Section modulus of member i along the strong axis
Z_{pb}	Plastic section modulus of beams
Z_{pc}	Plastic section modulus of columns
$Z_{\text{p},i}$	Plastic section modulus of member i
α	Learning rate
β	COF at a node
β_k	COF at node k
γ	Discount rate
$\Delta_{i,x}$	x -directional component of the unitary directional vector of member i
$\Delta_{i,y}$	y -directional component of the unitary directional vector of member i
$\Delta_{i,z}$	z -directional component of the unitary directional vector of member i
ϵ	Small number $\in [0, 1]$ used for ϵ -greedy policy
η	Factor $\in (0, 1)$ to adjust the inheritance of past updates in RMSProp
η_c	Axial force ratio
θ	Trainable parameters of the NN

$\tilde{\theta}$	Previous trainable parameters of the NN
θ^*	Ideal trainable parameters of the NN
$\theta_k^{(i)}$	Trainable parameters of k -th unit in layer i of the NN
$\Delta\theta_k^{(i)}$	Momentum of updating trainable parameters $\theta_k^{(i)}$
θ	Trainable parameters used in the proposed GE method
θ_1	Trainable parameters to weight member inputs
θ_2	Trainable parameters to weight node inputs
θ_3	Trainable parameters to weight node inputs
θ_4	Trainable parameters to weight self-member features
θ_5	Trainable parameters to weight features of neighbor members
θ_6	Trainable parameters to weight features of neighbor members
θ_7	Trainable parameters to compute action values from member features
θ_8	Trainable parameters to compute action values from member features
θ_9	Trainable parameters to compute action values from member features
Θ	Set of trainable parameters $\{\theta_1, \dots, \theta_9\}$
$\tilde{\Theta}$	Set of previous trainable parameters
λ	Effective slenderness ratio
Λ	Critical slenderness ratio
μ_i	Member feature of member i in the graph
$\mu_i^{(t_u)}$	Member feature of member i at step t_u
$\hat{\mu}$	Set of member features in the graph
$\hat{\mu}^{(t_u)}$	Set of member features at step t_u
$\hat{\mu}_{\Sigma}^{(T_u)}$	Matrix containing the sum of the member features
ν	Coefficient of momentum $\in (0, 1)$ for updating the NN parameters
π	Policy
π_p	Probability of transition based on policy π
$\bar{\sigma}$	Upper bound stress
$\sigma_{i,j}$	Axial stress of truss member i for load case j
σ_i^a	Axial stress of beam member i
$\sigma_{i,j}^b$	Bending stress of at the j -th tip of beam member i
τ	Factor in computing full plastic moments
$\varphi^{(i)}$	Activation function assigned to layer i of the NN
$\Phi_{i,j}$	Set of indices of members connecting to the j -th end of member i
ψ_i	Ratio of the weight of story i to the total structural weight
$\omega_k^{(i)}$	Running average of the magnitudes of recent gradients for $\theta_k^{(i)}$
Ω_a	Set of indices of available actions
Ω_b	Set of indices of beam members
Ω_c	Set of indices of column members
Ω_d	Set of indices of DOFs of existing nodes
Ω_m	Set of indices of existing members
Ω_R	Set of indices of reactions to be specified in the FDM
Ω_{β}	Set of indices of middle-layer nodes

Chapter 1

Introduction

1.1 Background of overall study

1.1.1 Structural optimization

Structural design of buildings requires complex decision making that considers costs and human activities and is not limited to simple strength and displacement calculations. Furthermore, there are a wide variety of contents to be integrated for structural design, such as materials, member cross-sections and their layout, and the overall shape. For these difficult requirements, major decisions have usually been made by architects without the strong involvement of engineers during the initial conceptual design phase, which significantly influences the cost of the project [1]. Structural engineers and other consultants conventionally work to realize the conceptual design, in which there are few opportunities for them to greatly improve the initial design [2]. In addition, the structural design is conventionally determined by the experience and intuition of structural engineers at first, and then its performance is evaluated through numerical analysis with a computer. However, this conventional process is inefficient due to a great deal of iterative work to change and evaluate the structural design, and it becomes difficult to design structures with complex shapes and mechanical properties.

Structural optimization is a technology that applies the optimization theory to the design of building structures, and mechanically and mathematically determines a rational structure by making full use of computational power. To implement an optimization approach, it is necessary to determine which variables to change during the optimization and formulate a numerical index called objective function that the structure should maximize (or minimize) and constraints that the structure must satisfy. After formulating the optimization problem consisting of the above elements, the solution can be searched using an optimization algorithm. By applying structural optimization to the architectural design process, not only can designers and computers collaborate efficiently to reach the desired structure, but also can design complex architectural shapes that are difficult to reach only with the intuition of architects and structural engineers. Cui and Sasaki [3] proposed a plan for the competition of a new station in Florence, Italy, which has a structure with an organic shape obtained by structural optimization removing unnecessary materials that do not bear stress.

Kimura et al. [4] optimized the structure of a funeral hall with an organic roof shape with the thickness and shape of the roof as design variables, and successfully reduced the total structural weight. Zegard et al. [5] implemented density-based topology optimization to a 3D-printed housing project to obtain a material distribution that achieves both light-weight and strong structure.

Mathematical programming and metaheuristics have mainly been used as optimization algorithms to solve structural optimization problems. Mathematical programming guarantees the local optimality of the solution based on the gradient of the objective and constraint functions and is frequently used for continuous optimization problems, where design variables take continuous values. Simplex method [9] and interior point method [10] are representative mathematical programming methods for linear optimization problems. For non-linear optimization problems, gradient descent method [11] and sequential programming method [12] are widely acknowledged as available algorithms. Although a metaheuristic is categorized as a stochastic optimization method that does not guarantee to satisfy any optimality criterion; however, it is applicable to a variety of optimization problems including discrete optimization problems, where design variables take integer values. There are a number of metaheuristic methods, such as genetic algorithm (GA) [13], simulated annealing (SA) [14] and particle swarm optimization (PSO) [15].

1.1.2 Machine learning for structural engineering

Apart from the above methods, there is an increasing number of studies that attempt structural optimization based on the knowledge acquired by machine learning (ML). ML is a method of analyzing patterns from many samples. One field of ML is supervised learning, in which sets of input-output pairs are given as training data so that the output can be predicted from the input. ML methods capable of approximating a highly nonlinear function with a support vector machine (SVM) or a neural network (NN) have become available as open-source libraries for Python [16, 17] and a toolbox for MATLAB [18], and attempts have been made to predict the complex response and performance of structures using supervised learning. Examples of the prediction target are diverse including the shear strength of reinforced concrete beams [19], the compression strength of a concrete material [20], and the ground vibration induced by blasting [21].

The trained ML model may be used to reduce the computational cost during optimization. Tamura et al. [22] used an SVM and a decision tree to classify desirable and non-desirable brace placements of steel frames and the trained model boosted the efficiency of optimization with SA. Papadrakakis et al. [23] used a multilayer perceptron, which is a simple class of NNs, to predict the objective function value for shape optimization of continuous materials and size optimization of frames. Note that these researches are also categorized as supervised learning.

Reinforcement learning (RL) is also an area of ML, in which rewards for giving feedback to an action taker called agent are defined, and the agent is trained to learn which action to take so as to maximize the cumulative reward. Unlike supervised learning, the training does not require input-output data sets, and the training is performed in an environment where a next state and a reward are sequentially observed by inputting the current state and an action. Assuming that the next state and the reward observed there do not depend on the



(a)



(b)



(c)

Figure 1.1: Examples of architectural works that are designed with the aid of structural optimization. (a) Proposal for the competition of a new station in Florence, Italy [6]. (b) Crematorium in Kawaguchi, Japan [7]. (c) 3D-printed house, the United States [8] (left and middle photos: Carlos Jones; right photo: Jason Richards)

history of past states or actions, but only on the current state-action pair, and such a sequential decision-making process is called a Markov decision process (MDP) [24]. The agent selects an action based on *value*, which is associated with the expected future rewards related to each state and action. By repeatedly performing simulations in the environment set up as described above, the agent learns to calculate more precise values in accordance with actual rewards based on the experience, and accordingly, becomes able to take appropriate actions that lead to obtaining more rewards. Previous researches showed that the trained agents overwhelmed the top players in Go [25], and obtained higher scores than skilled human players in arcade games [26]. There are also a few examples of using RL in the field of structural engineering. Nakamura and Suzuki [27] trained an RL agent to choose initial or tangential stiffness for each step of iterative calculation of nonlinear structural analysis for the purpose of accelerating convergence. Chiba et al. [28] learned the control method of an actuator attached to the steel frame by RL and confirmed response reduction against the earthquake. However, there are few pieces of research that tried to utilize RL for the optimal design of building structures.

RL and structural optimization are similar in that a certain objective function is defined and the solution with the best objective function is searched within the constraints. If the process of solving a structural optimization problem can be transformed into an MDP in the same way that a structural designer makes design changes at each step, a task synonymous with the problem solved by structural optimization becomes trainable by RL. Even when a professional engineer designs a building structure, if only the structural performance is focused on, the design can be changed by using only the current design contents as a judgment material without tracing the history of past design changes. Therefore, it is possible to formulate the design process of building structures as an MDP. Furthermore, the trained RL agents can be used to save the computational cost in the process of structural design, and they are also expected to become a powerful tool to support the decision making of structural designers. In addition, when the global optimality of the solution cannot be proved in the structural optimization problem, it is difficult to prepare sufficient desired outputs to implement supervised learning; on the other hand, RL does not require explicit input and output data as training material and thus it is easier to implement the training.

The remarkable results of ML in recent years largely depend on the improvement of ML algorithms. In particular, a convolutional neural network (CNN) [29] that adds arithmetic processing called convolution to a NN has made prominent achievements in the field of ML since the 2010s. For simplicity, the explanation is focused on the case where raster images consisting of a set of pixels are used as the input in the following. Convolution is a process in which a small matrix called a filter is overlaid on a part of the image and the linear summation of the filter value and the pixel value of the image is computed while sliding the filter over the entire area of the image. In the convolution process, the feature including the position information between pixels can be extracted, and patterns composed of a group of pixels can be recognized by repeating the convolution. Due to this property, CNNs have demonstrated high performance as ML algorithms for a wide range of tasks such as image classification [30, 31], image generation [32], object recognition [33, 34], audio classification [35], and natural language processing [36].

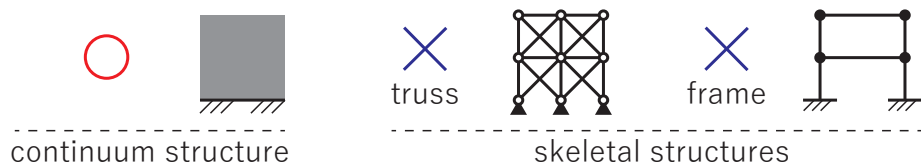


Figure 1.2: Types of structure that CNN can conveniently process and structures that CNN cannot.

Figure 1.2 shows types of structures that CNNs can conveniently handle and structures that CNNs cannot. Here, the skeletal structure is a type of structure consisting of a series of linear members such as trusses and frames. Due to its calculation procedure, the convolution operation is particularly suitable for extracting the feature of continuum structure models with a regular shape rather than skeletal structures, because a well-shaped continuum model can be easily converted into data expressed as a rectangular array. In fact, when glancing over previous studies using a CNN-based ML method for structural optimization problems, most of them are for continuum structures. For example, there are a number of CNN-based topology optimization methods proposed to predict an optimal material density distribution from a continuum structural model with specified boundary conditions [37, 38, 39, 40, 41, 42]. In other studies, the true stress distributions of continuum structures are predicted with a good accuracy using a CNN [43, 44]. Note that CNNs might be applicable to skeletal structures; in Ref. [45], a truss is replaced with a raster image composed of a set of pixels so as to process through a CNN for supervised learning; they intended to predict whether the optimal solution using the selected members as design variables is expected to be a good solution for structural weight minimization of the truss with pure compression. However, if a skeletal structure is pixelated for the CNN, the original data structure in which the member connectivity between nodes is explicitly given is destroyed, and it becomes difficult to consider the connectivity relationship of the skeletal structure.

Feature extraction methods for graphs called graph embedding (GE) have recently been proposed to deal with data structures that are difficult to handle in CNNs [46]. Here, a graph is a general term for data consisting of nodes and edges connecting them, and GE can be defined as an operation that calculates a vector representing some features of the input graph. GE enables to extract features of data with irregular connectivity, preserving the topological structure of the graph. GE-based methods have shown remarkable results for prediction of chemical properties of organic compounds [47, 48] and classification of users with multiple labels in social networking services (SNS) [49].

Attempts have been made to solve optimization problems for graphs by combining RL and GE. Bello et al. [50] applied RL to traveling salesman problems (TSPs) and achieved near-optimal results on two-dimensional Euclidian graphs with up to 100 nodes; however, the method is specially designed for TSP and is difficult to apply to other graph problems conveniently. Dai et al. [51] extracted the feature of each node using GE and formulated an RL method that calculates the value of selecting the node from the extracted feature. They tried to solve typical NP-hard combinatorial optimization problems such as minimum vertex cover, maximum cut, and TSP, and succeeded in obtaining a good solution with

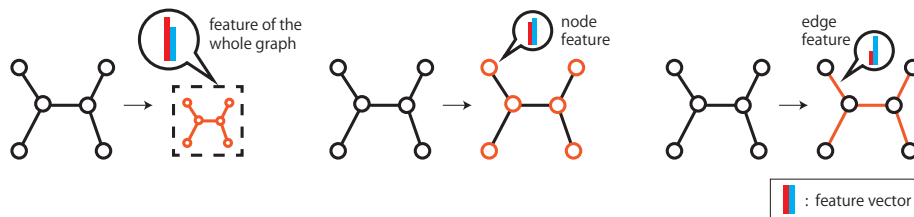


Figure 1.3: Three types of GE: whole graph embedding (left), node embedding (middle), and edge embedding (right).

a small computational load for a large-scale graph with 1000-1200 nodes after training with small graphs with 50-100 nodes.

However, this method is limited to the problem of choosing nodes, and cannot be applied to the problem of choosing edges. There are other methods that combine RL and GE [52, 53, 54]; however, they are also limited to the problem that selects nodes, and very few studies have focused on the problem that aims to extract edge features and select edges. Although Grover and Leskovec [55] and Zhao et al. [56] dealt with the problem of detecting the existence of an edge between two nodes through supervised learning, they predicted the existence of the edge from the node features at both ends of the assumed edge instead of embedding the feature of the assumed edge.

In fact, GE is roughly divided into three types depending on which property of the graph is embedded, as shown in Fig. 1.3. In whole graph embedding, the whole graph is represented with a single vector. In node embedding and edge embedding, each node and edge are encoded with their own vector representation, respectively. Wang et al. [57] pointed out that existing GE methods are node-based and the edge feature cannot be expressed as a low-dimensional vector directly.

When dealing with skeletal structures as a graph, it is common to represent the joints of members with nodes and the members with edges. For example, Furuta et al. [58] replaced the truss structure with a graph as described above and considers the stability conditions. In the initial phase of the design of building structures, the placement and cross-sections of the members are important rather than the design of the joints, to ensure that the joints hold sufficient strength over the members. In other words with the graph theory, it is important to extract the characteristics of edges instead of nodes as features and change the property of the edges when considering initial structural design problems. It might be a peculiarity of the design of building structures that edges are more important than nodes in the problem setting. Hence, for the problem of designing building members, it is desirable to combine methods of GE and RL focusing on how to embed the features of edges instead of nodes. For this reason, among the three major types of GE in Fig. 1.3, edge embedding should be the focus of this research.

1.1.3 Optimization of trusses

There are three major categories in the field of truss optimization, as shown in Fig. 1.4. The first one is size optimization, where the cross-sectional ar-

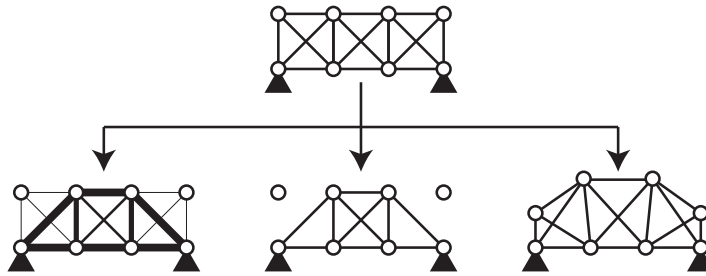


Figure 1.4: Three types of structural optimization for trusses. Size optimization (left). Topology optimization (middle). Geometry optimization (right).

areas of members vary as design variables. Since the 1960s, size optimization is extensively studied for the optimal design under stress constraints. An approximate optimal design can be obtained by the *fully stressed design* [59], where the stress of any member is equal to its upper or lower bound for at least one of the assumed loading conditions.

The method to obtain optimal connectivity of members is called topology optimization, which is a well-established field of research [60, 61]. In particular, the ground structure (GS) method is widely used for the basis of topology optimization, in which a densely connected structure called GS and unnecessary members are eliminated [62]. In topology optimization, the nodal locations do not move during the optimization. The complexity of the GS ranges from Level-1, where all neighboring nodes are connected, to full level, where each node is connected to all the other nodes. Although the full level GS is required in order to obtain the global optimal solution for the given nodes and the loading and support conditions, the combination of node pairs explodes to $n_n(n_n - 1)/2$ as the number of nodes n_n increases. For this reason, a truss with a limited number of members connecting to only adjacent nodes is frequently used as an initial GS, and the GS grids with Level-1 connectivity are used in this study.

However, it becomes difficult to obtain the global optimal topology even for the sparser initial GS stated above when stress constraints of the members are added to the optimization problem under multiple loading conditions, because the stress constraints can be neglected for members whose cross-sectional area is reduced to zero during the optimization process [60, 63, 64, 65]. To overcome this discontinuity, branch-and-bound type techniques are proposed. Sheu and Schmit [66] formulated a two-level optimization problem where the existence of members and nodes varies in the upper-level linear optimization problem and they are fixed in the lower-level nonlinear programming (NLP) problem. Ringertz [67] improved the above method so as to consider the displacement constraints in the lower-level problem; however, only small-scale trusses with 10 to 29 members are studied. Ohsaki and Katoh [68] also adopted a branch-and-bound approach to optimize trusses with constraints on stresses, nodal instability and the intersection of members; however, the number of LP steps and the computational cost increase drastically as the size of the problem increases. Besides branch-and-bound type techniques for trusses with integer design variables, metaheuristic techniques such as GA [69] and SA [70] have been presented. A method heuristically adding nodes and members from a relatively sparse GS

is also proposed [71].

Geometry optimization, or shape optimization, is a method to obtain the optimal shape of the truss geometry by controlling nodal locations. The most basic approach is to directly handle nodal coordinates as design variables; however, extremely short members, called melting or coalescent nodes, may appear if the domain in which the node can move is too large to overlap other domains. The occurrence of melting nodes causes two numerical difficulties; first, the stiffness equation cannot be solved due to the divergence of the member's axial stiffness; second, the speed of convergence becomes slower due to the divergence of the gradient of the member's axial stiffness with respect to the nodal coordinates.

A generalized GS method with variable nodal locations can also be used for optimizing the geometry of a truss. The authors proposed a force density method (FDM) for simultaneous optimization of topology and geometry of trusses [72, 73]. By using the force densities as design variables, the numerical difficulties caused by melting nodes were successfully avoided.

There are few research cases applying ML to truss structures. Hajela and Berke [74] utilized a multi-layer perceptron, a class of NNs that does not employ convolutional layers, for predicting the sensitivity of the design variables for optimization of frames and trusses. Lee et al. [75] also utilized a multi-layer perceptron for predicting nodal displacements and member stresses of 10-bar trusses. Swift and Batill [76] applied the trained multi-layer perceptron to structural optimization problems of trusses. Szewczyk and Hajela [77] adopted a counterpropagation network [78], which is a variant of the multi-layer perceptron, to predict the axial stresses from a set of cross-sectional areas, and utilized the trained network to save the computational cost during the optimization with SA for weight minimization with constraints on the axial stresses. However, in the above cases, the trained model cannot be applied to different trusses without modifying the structure of the NN and their methods are not versatile.

1.1.4 Cross-section optimization of steel frames

The frame is a type of structure consisting of linear members such as columns and beams. In some cases, truss structures in which the joints of the members are pin-jointed without restraining rotation may be included as a part of the frame structures, but here the frame structures are limited to those in which joints are rigidly connected to distinguish themselves from the truss structure. In the design of standard building frames, the frame shape is often determined mainly from the building plan. Therefore, determining the member cross-sections after fixing the shape occupies a large weight in the structural design. Although the fixing condition of the supports is also an important factor to design steel frames, the supports are not the target of the design changes and assumed to be fixed in both translation and rotation for simplicity in this study.

The truss is designed so that the load is transmitted only by the axial force; on the other hand, axial forces, shearing forces, and bending moments are considered as the internal forces in the frame structure. Assuming elastic deformation, the axial stiffness and the shear stiffness are proportional to the cross-sectional area of the member, while the bending stiffness is proportional to the second moment of area of the member. Assuming elastic deformation and the internal forces do not change, the axial and shear stresses are inversely

proportional to the cross-sectional area, and the bending stress is inversely proportional to the section modulus of the member. Therefore, it can be said that the correlation between the member cross-sections and the stresses/displacements in frames is more complicated than that in trusses. For the above reasons, the cross-section optimization problem of frames in which the cross-section of the members are handled as design variables is one of the problems of great practical interest.

If the member cross-sections are treated as continuous variables, the optimization problem can be formulated as an NLP based on the sensitivity analysis, and mathematical programming methods such as the extended Lagrange multiplier method [79] and the sequential quadratic programming method [12] can be used. Kimura et al. [80] used the plate thickness of square steel pipes and the flange-web thickness of I-beams as continuous design variables to formulate an optimization problem that minimizes the horizontal displacement at the top of the frame. However, most of the steel materials used for general buildings are standard products manufactured at a factory. The dimensions of the cross-section of available structural steel members are also standardized for each manufacturing plant, and the cross-sectional dimensions take discrete values. In order to deal with this situation, it is preferable that the design problem is described as a discrete or combinatorial optimization problem in which each member cross-section of the frame is selected from finite standard sizes [81]. In this case, metaheuristics such as GA [13] and SA [14] are often used to solve the optimization problems. In addition, a number of methods have been proposed to obtain the optimal discrete cross-sections by solving a relaxation problem with continuous variables [82, 83].

Tamura et al. [22] is an example of the application of ML in the optimal brace placement design of steel frames, where an area surrounded by the columns and the beams is regarded as one pixel, and convolution and pooling are implemented, which are arithmetic processes used in CNNs to extract features considering the arrangement of adjacent braces. Using the features, they trained a binary decision tree and an SVM whether the brace arrangement is preferable or not. The trained model greatly enhanced the efficiency of the solution search with SA.

However, the above method is designed for the area surrounded by columns and beams, and cannot be applied to design changes of columns and beams themselves. In the case of optimizing the cross-sections of a frame, it is possible to handle the properties of columns and beams more directly by inputting the frame configuration as a graph and applying GE to extract the member features.

1.2 Objective

The objective of this study is to demonstrate simplified structural design processes of skeletal structures through RL-based *intelligent* structural optimization in which the variables are changed based on the knowledge acquired by the algorithm instead of relying on mere repetitive calculations. To achieve this goal, skeletal structures are regarded as graphs consisting of nodes and edges, and GE is applied to extract the member features that capture the structural property from the inputs of nearby nodes and members. Furthermore, in order to implement RL, the feature vector of each member is mapped to a scalar value

indicating the action value of a pre-specified design change associated with the member so that the agent can quantitatively evaluate the design changes.

1.3 Advantages of the proposed method

When using the proposed GE method, the feature quantity can be extracted while explicitly maintaining the connectivity relations of skeletal structures unlike CNNs, and it can be expected to grasp the properties of skeletal structures more accurately.

Considering that conventional GE methods are mostly about extracting node features [57] and the method by Wang et al. [57], which is one of the few methods that can directly extract edge features, can handle node inputs only, the proposed GE method in this study is more suitable for the optimal design of the members of skeletal structures because both node and edge inputs can be handled simultaneously to express the member features. The attributes for nodes and members are both important inputs that characterize the structure. By using the proposed GE method, member features can be extracted without missing information on both nodes and members.

The computational cost to use the trained agent is very low. Although it requires an enormous number of simulations in the training phase to learn the causal relationship between the design change to the members and the change in the structural properties, the agents can efficiently find solutions without the need for repetitive calculations once trained.

Furthermore, the trained agent is versatile, because it can be applied to skeletal structures with different connectivity and the number of nodes and members while maintaining the size of training parameters. In other words, the trained model is re-usable to different structures without re-training if it properly learns the knowledge to change the structural design.

1.4 Thesis structure

The following chapters are organized as follows. Chapter 2 explains the theoretical background of RL, which forms the basis of the proposed method. In Chapter 3, the proposed method combining GE and RL is formulated in detail. RL tasks are problem-specific and must be defined for each problem. In Chapters 4 and 5, two optimization problems are demonstrated as the case studies and converted them into the tasks that can be handled by RL, and then the proposed method is applied to the training of the RL tasks. In Chapter 4, as the first case study, the agents learn the topology optimization problem of trusses through the proposed method. Furthermore, as an application example, the trained RL agents are utilized to generate initial solutions for simultaneous optimization of topology and geometry of trusses by the FDM. In Chapter 5, as the second case study, the proposed method is demonstrated through a discrete cross-section optimization problem for steel frame structures. Chapter 6 is a concluding remark that summarizes the above chapters and findings of the proposed method obtained through the case studies.

Chapter 2

Basics of reinforcement learning

2.1 Characteristic of reinforcement learning

Machine learning (ML) is a term first used by Arthur Samuel [84], and can be defined as the study of computer algorithms to learn tasks without explicit programming by humans. Because of this automatic self-improvement property, ML is regarded as a subset of artificial intelligence (AI). Depending on the approach of the algorithm, ML is roughly divided into three types: supervised learning, unsupervised learning, and reinforcement learning, as shown in Figs. 2.1 and 2.2.

Supervised learning requires a set of data that contains inputs and desired outputs, known as training data. Using them, a mathematical model is trained so that it approximates the output from the input only [85]. For example, Togootokh and Amartuvshin [86] tried to fine-tune one of the state-of-the-art pre-trained image classifier known as VGG19 [87] to recognize Chihuahua and muffin with 1000 pre-labeled images of them.

By contrast, unsupervised learning solely requires inputs and infers the structure in the inputs. One of the most common applications of unsupervised learning is clustering, which aims at finding hidden grouping in the data. van der Maaten and Hinton [88] proposed an unsupervised learning method called t-SNE, a visualization method for high-dimensional data like images by embedding them into a two or three-dimensional space.

Reinforcement learning (RL) seeks to take actions in an environment so as to maximize the cumulative reward. Similarly to unsupervised learning, RL does not require desired outputs as training data; instead, it requires a reward function that evaluates the current state. For instance, Gu et al. [89] utilized RL to train robots to learn a door opening task, in which the robot successfully learned the accurate sequence of the manipulation, which is very difficult to provide explicitly.

The name *reinforcement learning* derives from operant conditioning, a method of learning that improves behaviors through rewards and punishments, as observed in the experiment called *Skinner box* using rats [91, 92]. Figure 2.3 shows what the Skinner box looks like; electrical charges can be given to a rat, and the

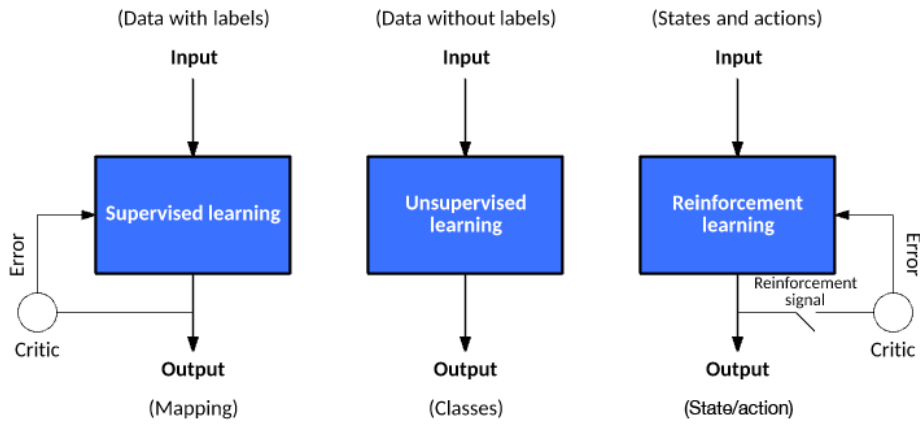


Figure 2.1: Three areas of ML: supervised learning (left), unsupervised learning (middle), and reinforcement learning (right) [90]

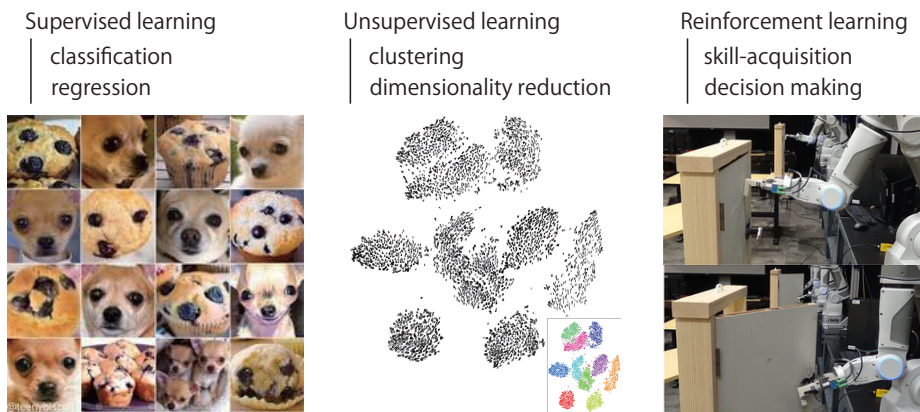


Figure 2.2: Application examples of supervised learning (left) [86], unsupervised learning (middle) [88], and reinforcement learning (right) [89]

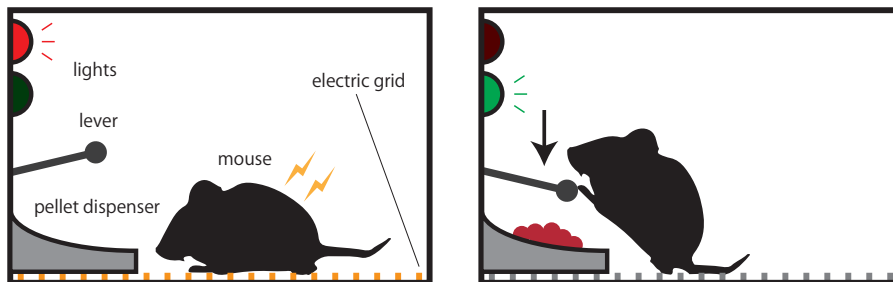


Figure 2.3: Conceptual diagram of a Skinner box [91, 92]

electric shock is turned off and pellets are dispensed when the lever is pressed down. At first, the rat accidentally presses down the lever to stop the electric shock and obtain the food, without noticing the relationship between pressing down the lever and the outcomes. However, the rat gradually learns the relationship while repeating the experience that the electrification can be interrupted and the pellet comes out when the lever is pressed down. Consequently, the behavior of pressing down the lever is *reinforced*. The observed result supports Thorndike’s *law of effect*, which suggests that behaviors that produce a satisfying outcome become more likely to occur and behaviors that produce an unpleasant outcome become less likely to occur Thorndike [93].

The concept of RL takes the same approach to the training of numerical models as this behavioral learning mechanism, characterized in the following two points:

- Selective : trying out different alternatives, comparing and choosing
- Associative : associating the alternatives and situations

Metaheuristics such as GA and SA are selective in the sense that variables are changed to obtain solutions and a good solution is selected among them; however, they are not associative because it merely searches the neighborhood of a good solution stochastically. On the other hand, supervised learning is associative because it gives sets of input and output as the learning material and learns the relationship between them, but not selective because the model does not learn the input-output relationship by changing variables [94].

The mathematical formulation of RL is based on the optimal control theory. In particular, as researches related to RL, Bellman [95, 96] defined the method of solving the Bellman equations for optimal control problems as dynamic programming. He also introduced a discrete stochastic control problem to solve with dynamic programming known as a Markov decision process (MDP) [97], which is explained in Sec. 2.2. Later, a notable variant of dynamic programming called policy iteration was developed [98]. These methods are still the foundational elements of the RL methods even today.

2.2 Markov decision process

RL mainly deals with tasks within the framework of MDP. As a prerequisite, the following elements are necessary to set up an MDP environment:

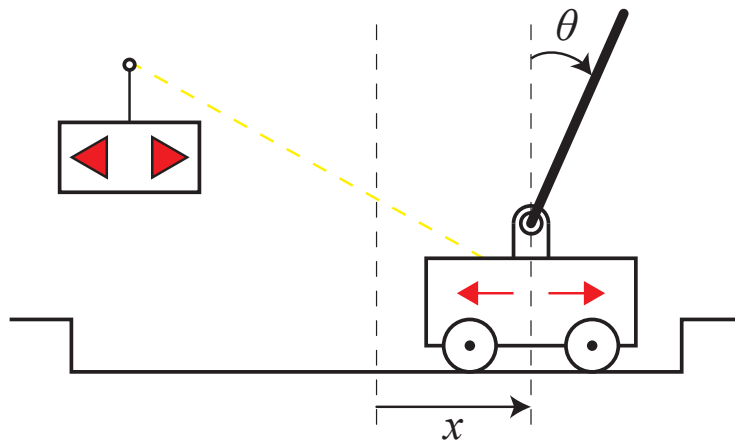


Figure 2.4: Pole balancing problem [99, 100]

- state s : input for the agent to make the decision
- action a : operation that changes the state s to the next state s'
- reward $r(s, s')$: reward immediately provided after transitioning from s to s'
- transition model $\{s, a\} \rightarrow s'$: model that inputs s and a and outputs s'

An example of the above elements are explained through a pole balancing problem [99, 100], which is a typical MDP. As shown in Fig. 2.4, a rigid pole is hinged to a motor-driven cart in a two-dimensional space. The remote controller applies an impact force of fixed magnitude in right or left direction to the cart at discrete time intervals. The cart-pole system is simulated with a detailed physical model considering various parameters such as the coefficient of friction, the length of the pole, the mass of the pole and of the cart. The goal of this problem is to operate the controller so as not to fall the pole and not to move the cart beyond the threshold from the original position without prior knowledge about the system.

The pole balancing problem contains four state variables:

- x : cart position
- θ : pole angle from the vertical axis
- \dot{x} : cart velocity
- $\dot{\theta}$: angle velocity of the pole

From the problem definition, two types of actions can be easily determined: applying an impulsive force to the right and to the left. One of the easiest reward settings is to provide a reward of +1 at every time-step if the pole remains upright [101].

The way an agent behaves is called a policy, which is described in the form of $\pi|s \rightarrow a$ with a probability of selecting an action a in a certain state s . In RL, the goal is to acquire the agent's policy that maximizes the cumulative reward finally obtained. In other words, the rewards must be appropriately set so that the agent achieves the goal by maximizing the rewards. Suppose we

try to acquire a strategy for winning Othello through RL. If more reward is given when taking a corner position, the agent may learn how to occupy the corners without the consideration of winning the game. Although taking corner positions is often advantageous in winning the game, a better reward setting is to give a reward only when winning the game.

A discrete time-step decision-making process is assumed hereafter. When considering how the environment changes at step $t + 1$ with respect to the action taken at step t , it is generally necessary to trace all the state, action, and reward histories from step 0 to t ; that is described as

$$\Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, \dots, r_1, s_0, a_0\} \quad (2.1)$$

The Markov property is one of the characteristics of specific stochastic processes, in which the probabilistic transition from the current state to the next and the expected future rewards does not depend on any past state, action, and reward, but only on the present state and the action selected [24].

$$\Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t\} \quad (2.2)$$

A task that satisfies the Markov property is called an MDP, and an MDP with finite state and action spaces is especially called a finite MDP. Examples of a finite MDP is the task of considering the next move of Othello and chess. In Othello and chess, it is not important to know the past situations and players' moves reaching the current board, and it is enough to know the situation on the current board to decide the next move. In the following, the discussion will proceed on the assumption that all the targeted RL tasks are finite MDPs. In addition, the MDPs in this study are assumed to be episodic having terminal states or the maximum steps, and the sequence of MDP from the initial step $t = 0$ to the terminal step $t = T$ is defined as an *episode* hereafter.

2.3 Reinforcement learning

2.3.1 Value function

The agent observes the next state and the reward from the environment when the action in a certain state is selected based on the policy. At this time, the policy is updated based on a value function: the notion of expected future rewards to be accumulated from the state. If the policy is updated solely using the reward instead of the value function, the agent will acquire a short-sighted policy that only considers immediate rewards. By using the value to update the policy, it is possible to acquire a better policy that considers not only immediate rewards but also expected future rewards.

In considering the value function, it is important to first define the notion of future rewards, known as a return. The simplest way to estimate a return R_t at step t is to compute the sum of *discounted* rewards while following the policy [102] as

$$R_t = \sum_{i=t}^T \gamma^{i-t} r_i \quad (2.3)$$

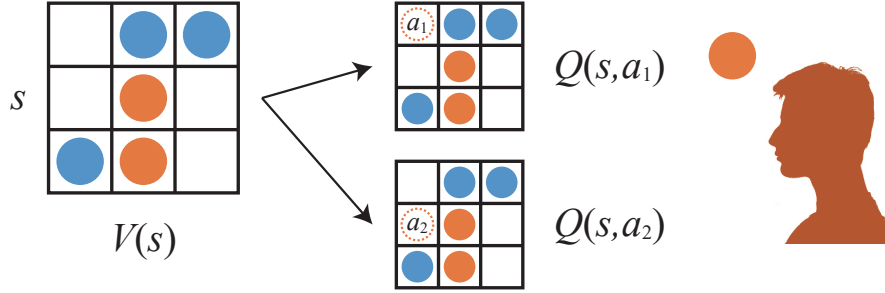


Figure 2.5: State value and action values in the situation of playing three lines

where $\gamma \in [0, 1]$ is a discount factor to adjust the importance of long-term rewards; the immediate reward is only considered and the following future rewards are disregarded when $\gamma = 0$ and the rewards at any future step are treated with the same importance as the immediate reward when $\gamma = 1$. The discount factor with a value less than 1 is particularly convenient when there are a large number of total steps in one episode, because it prevents the return from becoming an enormous value, and when training the agent to obtain relatively short-term rewards.

There are mainly the following two types of value functions: state value $V(s)$ and action value $Q(s, a)$.

- $V^\pi(s)$: expected return by successively following policy π from state s
- $Q^\pi(s, a)$: expected return by taking action a at state s and then successively following policy π

Let $\mathbb{E}_\pi\{R_t\}$ denote the expected return at step t by following policy π . Using $\mathbb{E}_\pi\{R_t\}$, the two values are specifically computed as

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi\{R_t | s_t = s\} \\ &= \mathbb{E}_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\} \end{aligned} \quad (2.4a)$$

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}_\pi\{R_t | s_t = s, a_t = a\} \\ &= \mathbb{E}_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right\} \end{aligned} \quad (2.4b)$$

These values can be intuitively understood supposing playing with three lines, as illustrated in Fig. 2.5. It is possible to judge if a current situation is advantageous or disadvantageous by looking at the board. At this time, the player estimates the state value $V(s)$ of the current state of the board. Simultaneously, the player's next move will change the situation of the board, and will be an important factor for winning or losing. When considering the next move from the current board, the player considers its action value $Q(s, a)$.

2.3.2 Policy

Let $P_{ss'}^a$ and $r_{ss'}^a$ denote the probability and the reward when transitioning from state s to state s' by taking action a , respectively. In order to determine the action that is expected to maximize the return from the current state value $V^\pi(s)$, all possible actions in that state should be listed, and the action that maximizes the state value of the next state should be selected as

$$\pi(s) = \operatorname{argmax}_a P_{ss'}^a (r_{ss'}^a + \gamma V(s')) \quad (2.5)$$

Similarly, once the action values are estimated, the agent's policy that is expected to maximize the return can be determined as

$$\pi(s) = \operatorname{argmax}_a Q(s, a) \quad (2.6)$$

This policy is called *greedy* policy. Equations (2.5) and (2.6) imply the importance of estimating correct state and action values, as a reasonable policy cannot be obtained without an accurate estimation of the values. However, deterministic policies such as the greedy policy are inefficient in exploring better solutions, because such policies persist in exploiting current state-value and action-value estimates and does not attempt other actions that are currently estimated to yield low returns, even though there may be other more profitable actions among them. During the training phase, the ϵ -greedy policy instead of the greedy policy is often used to enhance exploration in the state space, which is given as

$$\pi(s) = \begin{cases} \operatorname{argmax}_{a \in \{1, \dots, n_a\}} Q(s, a) & (\text{if random } [0, 1] > \epsilon) \\ \operatorname{unirand}_{a \in \{1, \dots, n_a\}} a & (\text{else}) \end{cases} \quad (2.7)$$

where `unirand` is an operation to choose a value from a set of values with a uniform probability. The difference of the ϵ -greedy policy from the standard greedy policy is that it chooses an action randomly with a low probability ϵ and this randomness improves exploration. Although both of exploring a space that the RL agent has less experienced and exploiting the agent's current knowledge are very important, they are fundamentally conflicting concepts and thus incompatible at the same time. This is well known as an exploration-exploitation dilemma [103, 104] and also discussed in the area of RL [105, 106]. In order to balance the exploration-exploitation trade-off, ϵ is typically set around 0.1. There are also studies on adaptive ϵ -greedy methods, where the value of ϵ changes during the training [107, 108]. Still, $\epsilon = 0.1$ is adopted for the ϵ -greedy policy in this study to focus on the formulation of the RL method.

RL methods can be classified into two types depending on how the policy is used and updated during the training. One is off-policy methods, which separates the policy to be improved and the policy used for simulation. The other is on-policy methods, which assume the future actions in the simulation are determined based on the policy to be improved; in other words, an on-policy method uses the same policy for improvement and for simulation.

The difference between off-policy and on-policy methods are illustrated in Fig. 2.6. In Fig. 2.6, the agent is trained to find the movement to obtain the reward of +1 at the goal, while avoiding bottom squares with the reward of

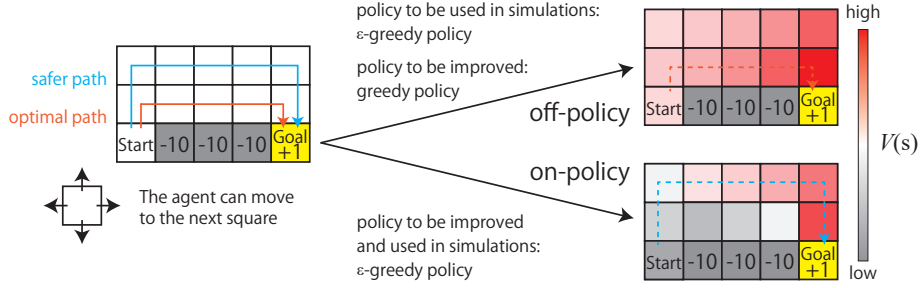


Figure 2.6: Difference of off-policy and on-policy methods. ϵ is fixed at a small positive value.

–10. If an off-policy method is adopted, the state values are updated without considering the randomness caused by the ϵ -greedy policy. In contrast, if an on-policy method is adopted, the state values are updated on the premise that the best action is not always taken; the lower state values are estimated for squares close to the bottom squares with the reward of -10 . In order to converge to the optimal policy using the on-policy method, it is necessary to gradually reduce ϵ to 0 during the training process.

In this study, an RL method is formulated based on an off-policy method. Simulations are performed using the ϵ -greedy policy with $\epsilon = 0.1$ during the training, and the deterministic greedy policy is used when designing the structure using the trained RL agent. Therefore, it is expected that the design using the trained RL agent will not be a redundant design with a margin of structural constraints, but a challenging design in which the amount of materials used is reduced as much as possible within the structural constraints.

2.3.3 Bellman equation

Let $\pi_p(s, a)$ denote the probability to choose an action a at a state s by the current policy. The Bellman equation is a consistency condition for any arbitrary policy π and state s that holds between the value of state s and the value of the possible successor states.

$$\begin{aligned}
V^\pi(s) &= \mathbb{E}_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right\} \\
&= \mathbb{E}_\pi \left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s \right\} \\
&= \sum_a \pi_p(s, a) \sum_{s'} P_{ss'}^a \left(r_{ss'}^a + \gamma \mathbb{E}_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_{t+1} = s' \right\} \right) \\
&= \sum_a \pi_p(s, a) \sum_{s'} P_{ss'}^a (r_{ss'}^a + \gamma V^\pi(s'))
\end{aligned} \tag{2.8a}$$

$$\begin{aligned}
Q^\pi(s, a) &= \mathbb{E}_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right\} \\
&= \mathbb{E}_\pi \left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s, a_t = a \right\} \\
&= \sum_{s'} P_{ss'}^a \left(r_{ss'}^a + \gamma \mathbb{E}_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_{t+1} = s' \right\} \right) \\
&= \sum_{s'} P_{ss'}^a \left(r_{ss'}^a + \gamma \sum_{a'} \mathbb{E}_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_{t+1} = s', a_{t+1} = a' \right\} \right) \\
&= \sum_{s'} P_{ss'}^a \left(r_{ss'}^a + \gamma \sum_{a'} \pi_p(s', a') Q^\pi(s', a') \right)
\end{aligned} \tag{2.8b}$$

where a' is the action taken at the next state s' . Note that the state and action values with respect to the next state s' are discounted by γ both in Eqs. (2.8a) and (2.8b). In order to maximize the return, the policy $*$ satisfying the following equations must be searched.

$$V^*(s) = \max_{\pi} V^\pi(s) \tag{2.9a}$$

$$\begin{aligned}
Q^*(s, a) &= \max_{\pi} Q^\pi(s, a) \\
&= \mathbb{E}_* \{ r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a \}
\end{aligned} \tag{2.9b}$$

The policy $*$ satisfying Eqs. (2.9a) and (2.9b) is called the optimal policy, and $V^*(s)$ and $Q^*(s, a)$ are called the optimal state value and the optimal action value, respectively. For the optimal policy $*$, the Bellman equations (2.8a) and (2.8b) are rewritten as

$$\begin{aligned}
V^*(s) &= \max_a \mathbb{E}_* \{ r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s \} \\
&= \max_a \sum_{s'} P_{ss'}^a (r_{ss'}^a + \gamma V^*(s'))
\end{aligned} \tag{2.10a}$$

$$\begin{aligned}
Q^*(s, a) &= \mathbb{E}_* \{ r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a \} \\
&= \sum_{s'} P_{ss'}^a \left(r_{ss'}^a + \gamma \max_{a'} Q^*(s', a') \right)
\end{aligned} \tag{2.10b}$$

Equation (2.10a) expresses that the state value under the optimal policy is equal to the expected return by taking the best actions from state s . Similarly, Eq. (2.10b) expresses that the action value under the optimal policy is equal to the expected return by taking the best actions after taking action a at state s .

2.3.4 Dynamic programming and Monte Carlo methods

A dynamic programming (DP) method [109, 98] and a Monte Carlo (MC) method [110, 94], which are the theoretical foundations of RL methods, are briefly explained.

DP is a method to compute state values using the Bellman equation and improve the policy based on the computed state values. Given that the number of states and actions are n_s and n_a , respectively. Although the number of possible deterministic policies is $n_a^{n_s}$, it is theoretically guaranteed to reach the optimal policy in polynomial time when using DP. Therefore, the convergence of dynamic programming is much faster than other methods based on enumeration.

However, when evaluating the policy, the value of all states must be calculated at each step, in which the computational cost is too high. In addition, since the current value of all states must be stored for updating the policy, a memory corresponding to the number of states is required. To make matters worse, DP can only be applied when the probability of state transitions $P_{ss'}^a$ is known for all the possible combinations of s , s' and a . In general tasks, $P_{ss'}^a$ is often unknown, and thus the applicability of DP is limited.

By contrast, MC does not require the probability of state transitions. MC is an algorithm that relies on the repetition of sampling the observation of states, actions, and rewards through the simulation of the RL task. Typically, the state value of s_t is updated using the following equation:

$$V(s_t) \leftarrow V(s_t) + \alpha (R_t - V(s_t)) \quad (2.11)$$

where $\alpha \in (0, 1]$ is the learning rate to determine to what extent the observation overrides old action values; the convergence of the learning is slow but stable when α is closer to 0, and fast but might be unstable due to divergence or oscillation of the estimated action values when α is closer to 1. As seen in Eq. (2.11), the estimation of state values is updated so as to minimize the error to R_t , the return to be obtained from step t . Since only the actually obtained rewards through the simulations are used to update the state values without estimating subsequent state values, MC is a stable method so that the correct state values can be estimated. Meanwhile, Eq. (2.11) means that the state value is updated once all the rewards from the current state to the terminal state are observed. In other words, the policy cannot be updated until the final outcome of the episode is known, which is disadvantageous in terms of learning efficiency for tasks with a large number of steps per episode.

2.3.5 Temporal difference learning

Temporal difference(TD) learning [111] is a class of RL methods that has the advantages of DP and MC. In TD learning, a typical update scheme for the state values is described as

$$V(s_t) \leftarrow V(s_t) + \alpha (r_{t+1} + \gamma V(s_{t+1}) - V(s_t)) \quad (2.12)$$

In Eq. (2.12), the estimation of $V(s_t)$ is updated so as to minimize the error to $r_{t+1} + \gamma V(s_{t+1})$. By using the actually observed immediate reward r_{t+1} and the estimated value of the next state $V(s_{t+1})$, the value and the policy can be updated every step like DP. Moreover, Eq. (2.12) does not require a probabilistic model of state transitions $P_{ss'}^a$ like MC.

For example, consider that the state values are estimated for the following 6 episodes:

$$s^{(1)} \xrightarrow{r=0} s^{(2)} \xrightarrow{r=0} s^{(3)} \quad (2.13a)$$

$$s^{(2)} \xrightarrow{r=0} s^{(3)} \quad (2.13b)$$

$$s^{(2)} \xrightarrow{r=1} s^{(3)} \quad (2.13c)$$

$$s^{(2)} \xrightarrow{r=1} s^{(3)} \quad (2.13d)$$

$$s^{(2)} \xrightarrow{r=1} s^{(3)} \quad (2.13e)$$

$$s^{(2)} \xrightarrow{r=1} s^{(3)} \quad (2.13f)$$

In this example, $V^*(s^{(2)})$, the optimum state value for $s^{(2)}$, is naturally expected to be $2/3$. Then how about $V^*(s^{(1)})$? If MC method is used, $V^*(s^{(1)})$ is estimated to be 0 as the return after observing state $s^{(1)}$ is 0, whereas, $V^*(s^{(1)})$ is estimated to be $\gamma V^*(s^{(2)}) = \gamma 2/3$ in TD learning as the transition probability from state $s^{(1)}$ to state $s^{(2)}$ is 100%. The difference between these two estimates is a good indication of the difference between MC and TD learning methods. In TD learning, not only the episode in which the state $s^{(1)}$ is observed but also the other episodes are associated to update the estimation of $V^*(s^{(1)})$, while MC methods merely focus on the episode in which $s^{(1)}$ is observed. This way, TD learning has the advantage of fast convergence through the learning based on a certainty-equivalence estimate, which assumes that the observed value of the transition ratio from the state $s^{(i)}$ to the state $s^{(j)}$ is the true transition probability, and the average of the rewards observed for a certain transition is assumed to be the true expected reward for the transition.

A SARSA algorithm [112] is an RL method that applies the concept of TD learning. The difference from TD learning is that it is formulated to learn action value estimates instead of state value. Before the learning begins, each action value is initialized with arbitrary values. In SARSA, the action value $Q(s, a)$ is updated as

$$Q(s, a) \leftarrow Q(s, a) + \alpha (r + \gamma Q(s', a') - Q(s, a)) \quad (2.14)$$

As seen in Eq. (2.14), SARSA depends on current state s , current action a , reward observed r , next state s' and next action a' , which is the origin of the name SARSA. SARSA minimizes the error between $Q(s, a)$ and $r + \gamma Q(s', a')$, which is the sum of the observed reward r and the discounted action value of s' and a' determined by the current policy. SARSA is categorized as on-policy method because the tuple (s, a, r, s', a') are obtained from the current policy to be improved.

Q-Learning [102] is a well established off-policy RL method. When reward r and next state s' are observed from state s by taking action a , estimation of action value $Q(s, a)$ is updated by the following equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma \max_a Q(s', a) - Q(s, a) \right) \quad (2.15)$$

In Eq. (2.15), $Q(s, a)$ is updated so as to reduce the error from $r + \gamma \max_a Q(s', a)$, which is the sum of observed reward r and the estimated optimal future state value at s' weighted by γ . Here, the greedy policy is used to estimate the optimal future state instead of the current policy; this is the reason why Q-Learning is an off-policy method.

Q-Learning is guaranteed to converge to the optimal action value $Q^*(s, a)$ for arbitrary state-action pairs under the following conditions: states and actions take discrete values, the action value is associated with each unique state-action pair without the use of function approximation, and all the state-action pairs are repeatedly observed [113, 114].

Note that the choice of γ will greatly affect the meaning of action values. If $\gamma = 0$, the error to reduce is $r - Q(s, a)$, which implies that the action value merely estimates an immediate reward obtained by taking action a at state s . If $\gamma = 1$, the error to reduce is $r + \max_a Q(s', a) - Q(s, a)$, which implies that future and immediate rewards are regarded as equivalently important.

2.4 Surrogate modelling

2.4.1 Curse of dimensionality

In order to store all the possible state-action pairs, a tabular representation is utilized, as shown in Table 2.1. The value in each cell represents the action value corresponding to taking action $a^{(j)}$ at state $s^{(i)}$. When a state-action pair is observed in the simulation, the corresponding cell value is updated using Eq. (2.15). This storing scheme is available when the number of possible state-action pairs $n_s \cdot n_a$ is small enough to store all the pairs in a computer's memory device.

Table 2.1: Tabular representation of action values

	$a^{(1)}$	\dots	$a^{(j)}$	\dots	$a^{(n_a)}$
$s^{(1)}$	$Q(s^{(1)}, a^{(1)})$	\dots	$Q(s^{(1)}, a^{(j)})$	\dots	$Q(s^{(1)}, a^{(n_a)})$
\vdots	\vdots	\ddots	\vdots	\ddots	\vdots
$s^{(i)}$	$Q(s^{(i)}, a^{(1)})$	\dots	$Q(s^{(i)}, a^{(j)})$	\dots	$Q(s^{(i)}, a^{(n_a)})$
\vdots	\vdots	\ddots	\vdots	\ddots	\vdots
$s^{(n_s)}$	$Q(s^{(n_s)}, a^{(1)})$	\dots	$Q(s^{(n_s)}, a^{(j)})$	\dots	$Q(s^{(n_s)}, a^{(n_a)})$

However, a problem arises due to the curse of dimensionality [115]; when the state and action spaces are too large, it is unrealistic to estimate, update and store the exact action values for all the state-action pairs. For example, Othello's state space size is approximately 10^{28} [116] and that of Chess is roughly 10^{43} [117]. For such a case, it is necessary to approximate action values using a surrogate model such as a NN which is explained in the next section. Deep-Q network [26], a novel method for learning to approximate action values using NN, is further described in Sec. 2.4.3.

2.4.2 Neural network

A NN is a technique to approximate an input-output relationship. The concept of NN derives from how neurons in a brain are thought to work. Note that convolution, a pervasive technique for NNs to reduce the dimensionality of rectangular arrays while preserving spatial information between pixels, is not explained here because they are not used in this study.

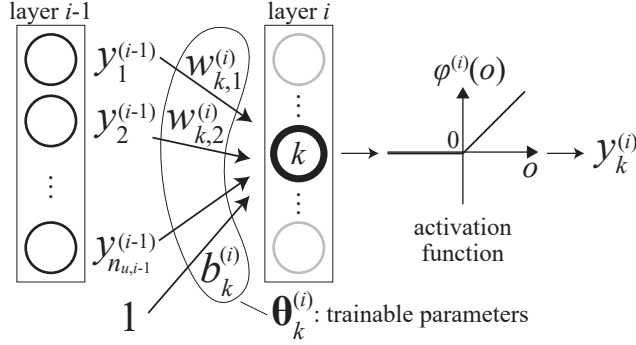


Figure 2.7: Illustration on mathematical operation in each neuron

The NN architecture is defined by the number of layers n_l , the number of units in each layer $n_{u,i}$ ($i = 1, \dots, n_l$), and the choice of activation functions for each layer. Let $w_{k,j}^{(i)}$ denote j -th weight and $b_k^{(i)}$ the bias of k -th unit in layer i , respectively. The unit contains weights $w_{k,1}^{(i)}, \dots, w_{k,n_{u,i-1}}^{(i)}$ and a bias $b_k^{(i)}$, which are the trainable parameters that vary during the training; they are written as $\theta_k^{(i)}$ for brevity.

$y_k^{(i)}$, k -th unit's output in layer i , is expressed as

$$y_k^{(i)} = \varphi^{(i)} \left(\left(\sum_{j=1}^{n_{u,i-1}} w_{k,j}^{(i)} y_j^{(i-1)} \right) + b_k^{(i)} \right) \quad (2.16)$$

Figure 2.7 illustrates how inputs are transmitted through the k -th neuron in layer i . Like a neuron passing electric signals, the unit takes values from units in the previous layer $i - 1$ as inputs, computes a weighted linear sum of them with a bias term, and passes the value through an activation function $\varphi^{(i)}$.

Six representative activation functions are shown in Fig. 2.8. An activation function φ needs to be nonlinear in order to approximate a nonlinear function and differentiable to apply a gradient-based optimization algorithm. ReLU is widely used as an activation function of NNs despite the non-differentiability at $x = 0$, because its computational cost is very low and the gradient does not vanish even when the input is a large positive value.

Let n_o denote the number of outputs from the NN. If the target NN output \bar{z}_k ($k = 1, \dots, n_o$) is known, the loss function L can be determined as the mean squared error between \bar{z}_k and z_k , the k -th output from the NN, as

$$L = \sum_{k=1}^{n_o} (z_k - \bar{z}_k)^2 \quad (2.17)$$

The NN is trained to minimize the loss by modifying the trainable parameters of the units. The simplest way is a gradient descent method where the trainable parameters $\theta_k^{(i)}$ are iteratively modified based on the gradient of the loss function as

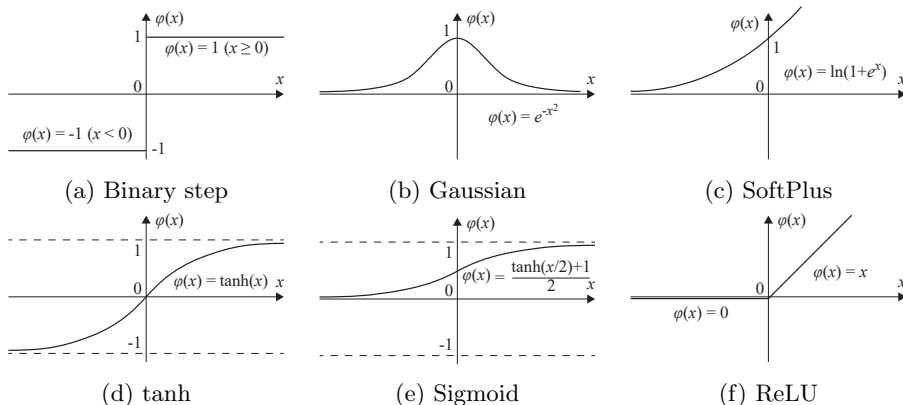


Figure 2.8: Examples of activation functions

$$\theta_k^{(i)} \leftarrow \theta_k^{(i)} - \alpha \frac{\partial L}{\partial \theta_k^{(i)}} \quad (2.18)$$

where α is the learning rate as introduced in Eq. (2.15). This method is well known as stochastic gradient descent (SGD) [118]. In application, $\partial L / \partial \theta_k^{(i)}$ is often obtained by randomly sampling a set of losses called a *mini-batch* and computing the average of them to stabilize the training, referred as the batch gradient descent method [119].

To prevent oscillation during iterative updates with an SGD method, a momentum term can be added to Eq. (2.18) as

$$\theta_k^{(i)} \leftarrow \theta_k^{(i)} + \nu \Delta \theta_k^{(i)} - \alpha \frac{\partial L}{\partial \theta_k^{(i)}} \quad (2.19a)$$

$$\Delta \theta_k^{(i)} \leftarrow \nu \Delta \theta_k^{(i)} - \alpha \frac{\partial L}{\partial \theta_k^{(i)}} \quad (2.19b)$$

where $\nu \in (0, 1)$ is a coefficient of momentum that determines how much the algorithm inherits the previous updates.

Root mean square propagation (RMSProp) [120] is one of the most prevalent variants of SGD, in which the learning rate varies for each of the trainable parameters. The updating scheme of RMSProp is given as

$$\theta_k^{(i)} \leftarrow \theta_k^{(i)} - \frac{\alpha}{\sqrt{\omega_k^{(i)}}} \frac{\partial L}{\partial \theta_k^{(i)}} \quad (2.20a)$$

$$\omega_k^{(i)} \leftarrow \eta \omega_k^{(i)} + (1 - \eta) \left(\frac{\partial L}{\partial \theta_k^{(i)}} \right)^2 \quad (2.20b)$$

where $\eta \in (0, 1)$ is the factor to adjust how much the past update amount is inherited and $\omega_k^{(i)}$ is a running average of the magnitudes of recent gradients for the trainable parameters. The running average is utilized to divide the learning rate α for the trainable parameters in Eq. (2.20a).

To implement one of the update algorithms above, it is necessary to calculate the partial derivatives $\partial L / \partial \theta_k^{(i)}$. For simplicity, only the k -th NN output z_k is focused and the parameters in the k -th unit in the output (i.e. n_1 -th) layer $\theta_k^{(n_1)}$ is derived. The derivatives can be decomposed using the chain rule as

$$\frac{\partial L}{\partial \theta_k^{(n_1)}} = \frac{\partial L}{\partial z_k} \frac{\partial z_k}{\partial o_k} \frac{\partial o_k}{\partial \theta_k^{(n_1)}} \quad (2.21)$$

where o_k is the k -th unit's linear combination of inputs from the previous layer. The first factor of the right-hand side is easily obtained from Eq. (2.17) as

$$\frac{\partial L}{\partial z_k} = \frac{\partial (z_k - \bar{z}_k)^2}{\partial z_k} = 2(z_k - \bar{z}_k) \quad (2.22)$$

The second factor is obtained by partial differentiation of the activation function $\varphi : o \rightarrow z$ as the following equation holds:

$$\frac{\partial z_k}{\partial o_k} = \frac{\partial \varphi(o_k)}{\partial o_k} \quad (2.23)$$

From Eq. (2.23), the activation function needs to be differentiable. The last factor is calculated depending on whether the trainable parameter is a weight or a bias; which is given as

$$\frac{\partial o_k}{\partial w_{k,j}^{(n_1)}} = \frac{\partial \left(\sum_{j=1}^{n_u, n_1-1} \left(w_{k,j}^{(n_1)} y_j^{(n_1-1)} \right) + b_k^{(n_1)} \right)}{\partial w_{k,j}^{(n_1)}} = y_j^{(n_1-1)} \quad (2.24a)$$

$$\frac{\partial o_k}{\partial b_k^{(n_1)}} = \frac{\partial \left(\sum_{j=1}^{n_u, n_1-1} \left(w_{k,j}^{(n_1)} y_j^{(n_1-1)} \right) + b_k^{(n_1)} \right)}{\partial b_k^{(n_1)}} = 1 \quad (2.24b)$$

This way, the derivatives with respect to the parameters can be obtained analytically. The derivatives of the trainable parameters in the previous layers are also obtained in a similar way using the chain rule, which implies that they depend on the derivatives in the subsequent layers. Hence, as opposed to the forwarding process from inputs to outputs, the computation of the gradients is implemented from the output layer to the input layer. For this reason, the gradient descent methods for NNs mentioned above are called *back-propagation* method as a generic term; see more details on [121].

2.4.3 Deep Q-network

A deep Q-network (DQN) [26] is a method combining conventional Q-Learning and deep learning: a method to estimate the relation between inputs and outputs using a deep NN architecture. In a DQN, the agent learns by tuning the NN parameters θ to find the ideal values θ^* so that it can approximate proper action values $Q(s, a; \theta)$ from the input states as

$$Q(s, a; \theta^*) \approx Q^*(s, a) \quad (2.25)$$

where $Q^*(s, a)$ is an optimal action value equal to the highest expected discounted rewards obtained by an optimal sequence of taking actions. Equation (2.15) intended to minimize the difference between the current estimation

of the action value $Q(s, a)$ and the sum of the observed reward and the discounted Q-value thereafter $r + \gamma \max_a Q(s', a)$. Similarly, the NN parameters θ are optimized to minimize the squared error L defined as

$$L(\theta) = \left(r(s') + \gamma \max_a Q(s', a; \theta) - Q(s, a; \theta) \right)^2 \quad (2.26)$$

The optimal target values $r(s') + \gamma \max_a Q(s', a; \theta)$ are substituted by $r(s') + \gamma \max_{\tilde{a}} Q(s', \tilde{a}; \tilde{\theta})$, using $\tilde{\theta}$ from previous θ to stabilize the algorithm, where \tilde{a} is distinguished from a because it is derived from different Q-values [26]. Therefore, the NN parameters are tuned so as to minimize the following equation:

$$\text{minimize } \tilde{L}(\theta) = \left(r(s') + \gamma \max_{\tilde{a}} Q(s', \tilde{a}; \tilde{\theta}) - Q(s, a; \theta) \right)^2 \quad (2.27)$$

Since the loss function is defined this way, the tuning problem (2.27) can be solved with any arbitrary back-propagation method, which is already explained in Sec. 2.4.

Another contribution of the DQN is the use of experience replay [122] for mini-batch updates instead of single updates on the last experience [114]. Experience replay is a technique to store the tuples of agent's experience at each step (s_t, a_t, r_t, s_{t+1}) into a memory. In the process of tuning the NN parameters, a set of random tuples from the memory to create a mini-batch data and compute the averaged loss computed using Eq. (2.27) for each tuple. In order to utilize experience replay, the RL method needs to be off-policy because the parameters of the NN when each sample is stored in the memory and those of the current NN are different. This is the reason why the DQN is based on Q-Learning, which is an off-policy method, rather than on-policy methods like SARSA.

2.5 Conclusion

In this chapter, several conventional RL methods have been briefly summarized, which are the basis of the proposed method to be introduced in Chapter 3. The relationship of this chapter and Chapter 3 is illustrated in Fig. 2.9. RL is characterized as a method for simulating the behavioral learning mechanism for the training process, in which rewards are given according to the selected action and the observed state. The methods explained here aim at estimating the optimal value functions that consider not only immediate rewards but also future rewards. Since the number of action values is equal to the product of the numbers of states and actions, it is necessary to approximate the value functions with a small number of parameters when dealing with a problem with a huge number of states. NNs are a typical example of function approximation methods, and the DQN algorithm is regarded as an important approach that combines NNs and RL. The original DQN utilizes a specific type of NN called the CNN, which specializes in extracting features from rectangular arrays and is not suitable for data with irregular connections. The proposed method explained in the next chapter adopts a GE-based approach instead of NNs for approximating the action values. By introducing GE, the features of data expressed as a graph with irregular connectivity of nodes can be easily extracted.

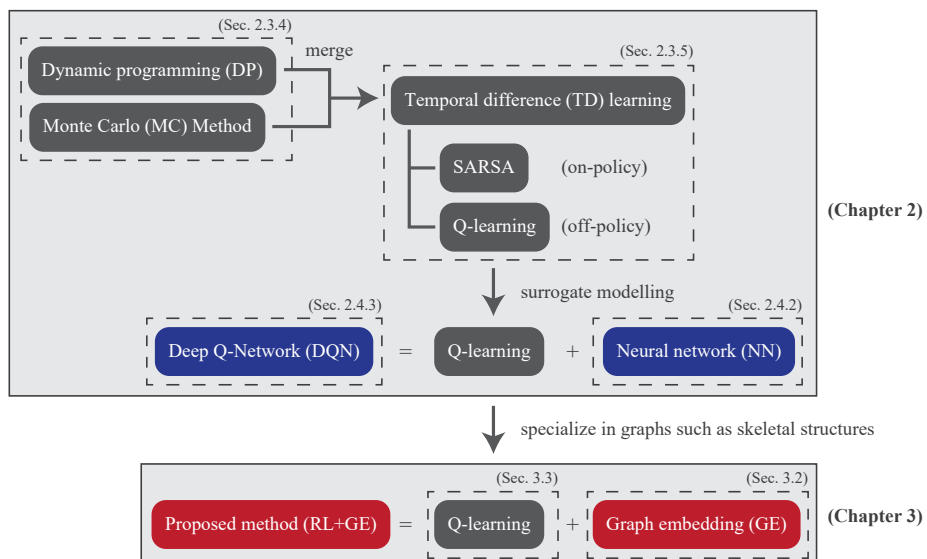


Figure 2.9: The relationship of the contents explained in Chapter 2 and those to be explained in Chapter 3.

Chapter 3

Hybrid method of graph embedding and reinforcement learning for training an optimal design agent of skeletal structures

3.1 Conversion of an optimization problem into a reinforcement learning task

In this section, the common procedure to convert an optimization problem into an RL task is described. As explained in Sec. 2.2, the RL task needs to be defined as a finite MDP. Therefore, state s , action a and reward r , which are the elements of the finite MDP, must be defined. Note that the transition of states by taking an action is deterministic in this study, because the environment for training agents assumes the structural design of skeletal structures, and it is not necessary to consider randomness when changing the design of the structure. Although RL tasks require problem-specific settings and cannot be fully generalized, here the common points are explained regardless of the types of structural optimization problems to be converted into the RL task. More concrete settings are explained in each individual problem in Chapters 4 and 5.

(1) state s

Consider a skeletal structure with n_n nodes and n_m members. In addition, given that each node input, member input, and member feature to be extracted have dimensions n_{fn} , n_{fm} , and n_f , respectively. Let θ denote the trainable parameters that vary during the training. The roll of θ is same as θ in NNs explained in Chapter 2; weighting the inputs for approximating desirable outputs. In the proposed method, a state s is represented as a set of member features $\hat{\mu} = [\mu_1, \dots, \mu_{n_m}] \in \mathbb{R}^{n_f \times n_m}$ embedded from numerical data at nodes $\hat{v} =$

$[\mathbf{v}_1, \dots, \mathbf{v}_{n_n}] \in \mathbb{R}^{n_{fn} \times n_n}$, those at members $\hat{\mathbf{w}} = [\mathbf{w}_1, \dots, \mathbf{w}_{n_m}] \in \mathbb{R}^{n_{fm} \times n_m}$, and trainable parameters θ , which is expressed as

$$s \approx \hat{\mu}(\hat{\mathbf{v}}, \hat{\mathbf{w}}, \theta) \quad (3.1)$$

It is desirable for a state to include numerical information of the skeletal structure as much as possible, such as the geometry of the structure, the property of the nodes and members, and the loading and support conditions, which can be the elements of inputs $\hat{\mathbf{v}}$ and $\hat{\mathbf{w}}$. The dimension of node inputs \mathbf{v}_k ($k \in \{1, \dots, n_n\}$) can take an arbitrary positive integer depending on the problem definition, and the same is true for the dimension of member inputs \mathbf{w}_i ($i \in \{1, \dots, n_m\}$). Note that the values of $\hat{\mathbf{v}}$ and $\hat{\mathbf{w}}$ should be within the range of $[-1, 1]$ in order to enhance the RL agent's performance. This is because the larger values will have a higher contribution to the output error, and the error reduction algorithm for the RL agent will neglect the information from the small-valued variables [123]. Therefore, the best situation is when all inputs are in the same order of magnitude [124]. Several studies also discussed achieving scale-invariance of the inputs [125, 126, 127]. Note that the connectivity of nodes and members is not considered here as it can be expressed by GE described in Sec. 3.2.

For the process in which the total structural volume is gradually increased by the design change, the terminal state is defined as when the structural design reaches a feasible design from an initial infeasible design. Similarly, for the process in which the total structural volume is gradually reduced by the design change, the terminal state is defined as when reaches infeasible design from an initial redundant design. Otherwise, one can define the maximum step instead of the terminal state.

(2) action a

An action in this study is defined as the smallest unit of design changes applied to a member of the structure. One action is assigned to each member, and there are at most n_m types of actions that an agent can take in each step. In practice, in order to reduce the action space to be explored, the action is selected from Ω_a which represents a set of possible actions that excludes clearly inappropriate actions from n_m actions. Here, the excluded actions are those in which the next state associated with the action does not exist obviously, such as an action of removing a member that no longer exists and increasing/reducing a member size exceeding the upper/lower bounds.

(3) reward r

The basic idea of reward settings is to provide larger rewards for good design changes and smaller or negative rewards (i.e. penalties) for bad design changes based on the objective and constraints of the original structural optimization problem. In the same manner as node and member inputs $\hat{\mathbf{v}}$ and $\hat{\mathbf{w}}$, respectively, the rewards should be scaled within the range of $[-1, 1]$ so that the same hyper-parameters of the RL agent can be used across various environments [26, 128].

3.2 Edge features estimated by graph embedding

In this section, skeletal structures are regarded as graphs and the feature of each member is represented as a vector by aggregating numerical data about neighbor nodes and members. Let n_f denote the size of the feature vector of each member, which is to be determined through a careful adjustment with trial-and-error for better performance, because a size that is too small leads to inaccuracy in expressing the features, and a size that is too large requires redundant computation time in training and application of the agent after the training. The trainable parameters $\boldsymbol{\theta}_1 \in \mathbb{R}^{n_f \times n_{fm}}$, $\boldsymbol{\theta}_2 \in \mathbb{R}^{n_f \times n_f}$, $\boldsymbol{\theta}_3 \in \mathbb{R}^{n_f \times n_{fm}}$, $\boldsymbol{\theta}_4 \in \mathbb{R}^{n_f \times n_f}$, $\boldsymbol{\theta}_5 \in \mathbb{R}^{n_f \times n_f}$, and $\boldsymbol{\theta}_6 \in \mathbb{R}^{n_f \times n_f}$ are introduced to weight the inputs to extract features that integratively consider the structural property. Using $\boldsymbol{\theta}_1$ - $\boldsymbol{\theta}_6$, the feature vector of each member $\boldsymbol{\mu}_i \in \mathbb{R}^{n_f}$ ($i = 1, \dots, n_m$) is updated as follows:

$$\boldsymbol{\mu}_i^{(0)} = \mathbf{0} \quad (3.2a)$$

$$\boldsymbol{\mu}_i^{(t_u+1)} = \text{ReLU} \left(\mathbf{h}_1 + \mathbf{h}_2 + \mathbf{h}_3^{(t_u)} + \mathbf{h}_4^{(t_u)} \right) \quad (3.2b)$$

$$\mathbf{h}_1 = \boldsymbol{\theta}_1 \mathbf{w}_i \quad (3.2c)$$

$$\mathbf{h}_2 = \boldsymbol{\theta}_2 \sum_{j=1}^2 \text{ReLU}(\boldsymbol{\theta}_3 \mathbf{v}_{i,j}) \quad (3.2d)$$

$$\mathbf{h}_3^{(t_u)} = \boldsymbol{\theta}_4 \boldsymbol{\mu}_i^{(t_u)} \quad (3.2e)$$

$$\mathbf{h}_4^{(t_u)} = \boldsymbol{\theta}_5 \sum_{j=1}^2 \text{ReLU} \left(\boldsymbol{\theta}_6 \sum_{k \in \Phi_{i,j}} \boldsymbol{\mu}_k^{(t_u)} \right) \quad (3.2f)$$

where $\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3^{(t_u)}$ and $\mathbf{h}_4^{(t_u)}$ are the elements that make up the feature, t_u is the step number of updating the features, $\mathbf{v}_{i,j}$ is the nodal inputs of the j -th ($j \in \{1, 2\}$) end of member i , and $\Phi_{i,j}$ is the set of indices of members connecting to the j -th end of member i . Note that $\Phi_{i,j}$ does not include the index of member i itself. ReLU is one of activation functions as explained in Fig. 2.8, which is defined as

$$\text{ReLU}(x) = \begin{cases} x & (\text{if } x > 0) \\ 0 & (\text{else}) \end{cases} \quad (3.3)$$

To simplify the expression, the ReLU function is applied in Eq. (3.2) to a vector to output a vector with the same size.

The operation in Eq. (3.2) is illustrated in Fig. 3.1. The numerical data of two nodes and members connecting to them are aggregated into the feature vector of a member through a single operation. The aggregated numerical data include member input data \mathbf{h}_1 in Eq. (3.2c), connecting nodes' input data \mathbf{h}_2 in Eq. (3.2d), embedded feature of the member itself $\mathbf{h}_3^{(t_u)}$ in Eq. (3.2e), and embedded features of neighboring members $\mathbf{h}_4^{(t_u)}$ in Eq. (3.2f). Accordingly, $\hat{\boldsymbol{\mu}}^{(t_u)} = \{\boldsymbol{\mu}_1^{(t_u)}, \dots, \boldsymbol{\mu}_{n_m}^{(t_u)}\}$ is the set of feature vectors incorporating connectivity of the truss after $\boldsymbol{\mu}_i^{(t_u)}$ for all the members is computed from Eq. (3.2).

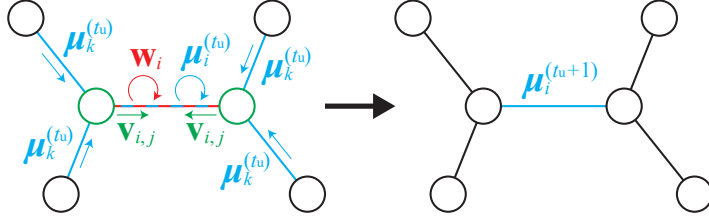


Figure 3.1: The concept of Eq. (3.2). It aggregates numerical data of neighbor nodes and members.

However, the operation of Eq. (3.2) should be iterated more than once in order to capture the features of distant members that are not directly connected. The number of iterations T_u should be carefully selected because it determines the training difficulty and how far distant nodes and members are considered. If T_u is set closer to 1, the computational cost is cheaper and the risk of divergence of trainable parameters $\theta_1, \dots, \theta_6$ during the training with a backpropagation method is lower; however, the extracted features cannot capture the property of distant nodes and members. In contrast, if a large positive integer is assigned to T_u , the extracted features are capable of considering distant nodes and members but the computational cost becomes higher and the risk of divergence of the trainable parameters also becomes higher. In accordance with the previous research of Dai et al. [51], the number of iterations T_u is set to be 4. It should be noted that the embedded feature vector $\mu_i^{(T_u)}$ has the same size n_f regardless of the connectivity. Owing to this property, all the members can be evaluated based on the same measure.

In the following, the operation of Eq. (3.2) is re-formulated utilizing matrix operations to reduce the computational cost. At first, the connectivity (or incidence) matrix $\mathbf{C} \in \mathbb{R}^{n_m \times n_n}$ for a directed graph is defined, such that each element C_{ij} is provided as follows:

$$C_{ij} = \begin{cases} -1 & \text{if member } i \text{ leaves node } j \\ 1 & \text{if member } i \text{ enters node } j \\ 0 & \text{else} \end{cases} \quad (3.4)$$

this matrix contains information about the connectivity between nodes and the orientation of the edges. The non-oriented connectivity matrix $\mathbf{C}_A \in \mathbb{R}^{n_m \times n_n}$ is obtained by taking an absolute value for each element of \mathbf{C} . We further obtain \mathbf{C}_1 and $\mathbf{C}_2 \in \mathbb{R}^{n_m \times n_n}$ to identify the nodes that each member leaves and those that each member enters as

$$\mathbf{C}_1 = \frac{1}{2} (\mathbf{C}_A - \mathbf{C}) \quad (3.5a)$$

$$\mathbf{C}_2 = \frac{1}{2} (\mathbf{C}_A + \mathbf{C}) \quad (3.5b)$$

Using these matrices, the operation of Equation (3.2) for all the members can be integrated as

$$\hat{\mu}^{(0)} = \mathbf{0} \quad (3.6a)$$

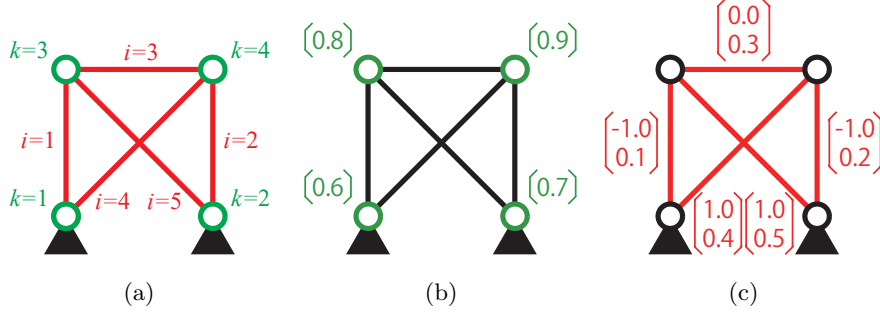


Figure 3.2: An example truss to extract member features. (a) Node and member index. (b) \mathbf{v}_k ($k = 1, \dots, 4$). (c) \mathbf{w}_i ($i = 1, \dots, 5$).

$$\hat{\boldsymbol{\mu}}^{(t_u+1)} = \text{ReLU} \left(\hat{\mathbf{h}}_1 + \hat{\mathbf{h}}_2 + \hat{\mathbf{h}}_3^{(t_u)} + \hat{\mathbf{h}}_4^{(t_u)} \right) \quad (3.6b)$$

$$\hat{\mathbf{h}}_1 = \boldsymbol{\theta}_1 \hat{\mathbf{w}} \quad (3.6c)$$

$$\hat{\mathbf{h}}_2 = \boldsymbol{\theta}_2 (\text{ReLU}(\boldsymbol{\theta}_3 \hat{\mathbf{v}})) \mathbf{C}_A^\top \quad (3.6d)$$

$$\hat{\mathbf{h}}_3^{(t_u)} = \boldsymbol{\theta}_4 \hat{\boldsymbol{\mu}}^{(t_u)} \quad (3.6e)$$

$$\begin{aligned} \hat{\mathbf{h}}_4^{(t_u)} = & \boldsymbol{\theta}_5 \left(\text{ReLU} \left(\boldsymbol{\theta}_6 \left(\mathbf{C}_1 \mathbf{C}_A^\top \hat{\boldsymbol{\mu}}^{(t_u)\top} - \hat{\boldsymbol{\mu}}^{(t_u)\top} \right)^\top \right) \right) \\ & + \text{ReLU} \left(\boldsymbol{\theta}_6 \left(\mathbf{C}_2 \mathbf{C}_A^\top \hat{\boldsymbol{\mu}}^{(t_u)\top} - \hat{\boldsymbol{\mu}}^{(t_u)\top} \right)^\top \right) \end{aligned} \quad (3.6f)$$

Although the operation of Eq. (3.6) yields the same result as repeating Eq. (3.2) for all the members, the former is more computationally efficient than the latter, owing to efficient matrix multiplication algorithms [129, 130, 131, 132].

Example: extraction of member features of a truss

In this example, the above method is demonstrated through a truss with four nodes and five members, as shown in Fig. 3.2(a). Given that the dimension of node and member inputs are $n_{\text{fn}} = 1$ and $n_{\text{fm}} = 2$, and that the node and member inputs are assigned as shown in Figs. 3.2(b) and 3.2(c), respectively. Note that the values here are arbitrarily provided without any structural meaning to focus on demonstrating the mathematical operations of the proposed GE method. The dimension of member features to be extracted is assumed to be $n_f = 3$.

Firstly, $\hat{\mathbf{v}} \in \mathbb{R}^{1 \times 4}$ and $\hat{\mathbf{w}} \in \mathbb{R}^{2 \times 5}$ are expressed as

$$\hat{\mathbf{v}} = [0.6 \quad 0.7 \quad 0.8 \quad 0.9] \quad (3.7a)$$

$$\hat{\mathbf{w}} = \begin{pmatrix} -1.0 & -1.0 & 0.0 & 1.0 & 1.0 \\ 0.1 & 0.2 & 0.3 & 0.4 & 0.5 \end{pmatrix} \quad (3.7b)$$

The connectivity matrix of the truss $\mathbf{C} \in \mathbb{R}^{5 \times 4}$ is obtained as

$$\mathbf{C} = \begin{pmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 1 \\ -1 & 0 & 0 & 1 \\ 0 & -1 & 1 & 0 \end{pmatrix} \quad (3.8)$$

Member 5 is directed from node 2 to node 3; accordingly, -1 is assigned to (5, 2) element, 1 is assigned to (5, 3) element of the matrix, and the other elements in the fifth row are all 0. The non-oriented connectivity matrix $\mathbf{C}_A \in \mathbb{R}^{5 \times 4}$ is obtained by simply computing an absolute value of the connectivity matrix for each element as

$$\mathbf{C}_A = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \quad (3.9)$$

Using \mathbf{C} and \mathbf{C}_A , \mathbf{C}_1 and $\mathbf{C}_2 \in \mathbb{R}^{n_m \times n_n}$ are also obtained as

$$\mathbf{C}_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad (3.10a)$$

$$\mathbf{C}_2 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (3.10b)$$

The values of trainable parameters $\boldsymbol{\theta}_1 \in \mathbb{R}^{3 \times 2}$, $\boldsymbol{\theta}_2 \in \mathbb{R}^{3 \times 3}$, $\boldsymbol{\theta}_3 \in \mathbb{R}^{3 \times 1}$, $\boldsymbol{\theta}_4 \in \mathbb{R}^{3 \times 3}$, $\boldsymbol{\theta}_5 \in \mathbb{R}^{3 \times 3}$, and $\boldsymbol{\theta}_6 \in \mathbb{R}^{3 \times 3}$ are assumed as follows.

$$\boldsymbol{\theta}_1 = \begin{pmatrix} 0.11 & 0.12 \\ 0.13 & 0.14 \\ 0.15 & 0.16 \end{pmatrix} \quad (3.11a)$$

$$\boldsymbol{\theta}_2 = \begin{pmatrix} 0.21 & 0.22 & 0.23 \\ 0.24 & 0.25 & 0.26 \\ 0.27 & 0.28 & 0.29 \end{pmatrix} \quad (3.11b)$$

$$\boldsymbol{\theta}_3 = \begin{pmatrix} 0.31 \\ 0.32 \\ 0.33 \end{pmatrix} \quad (3.11c)$$

$$\boldsymbol{\theta}_4 = \begin{pmatrix} -0.041 & -0.042 & -0.043 \\ 0.044 & 0.045 & 0.046 \\ -0.047 & -0.048 & -0.049 \end{pmatrix} \quad (3.11d)$$

$$\boldsymbol{\theta}_5 = \begin{pmatrix} 0.051 & 0.052 & 0.053 \\ -0.054 & -0.055 & -0.056 \\ 0.057 & 0.058 & 0.059 \end{pmatrix} \quad (3.11e)$$

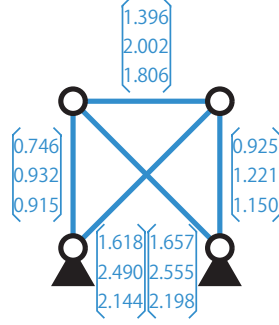


Figure 3.3: Extracted features $\hat{\boldsymbol{\mu}}^{(4)}$ of the example truss

$$\boldsymbol{\theta}_6 = \begin{pmatrix} -0.061 & -0.062 & -0.063 \\ 0.064 & 0.065 & 0.067 \\ -0.067 & -0.068 & -0.069 \end{pmatrix} \quad (3.11f)$$

Using the above values and Eq. (3.6), $\hat{\boldsymbol{\mu}}^{(1)} \in \mathbb{R}^{3 \times 5}$ is computed as

$$\hat{\boldsymbol{\mu}}^{(1)} = \begin{pmatrix} 0.198 & 0.252 & 0.395 & 0.475 & 0.487 \\ 0.220 & 0.282 & 0.450 & 0.546 & 0.560 \\ 0.243 & 0.312 & 0.505 & 0.618 & 0.634 \end{pmatrix} \quad (3.12)$$

Similarly, $\hat{\boldsymbol{\mu}}^{(2)}$, $\hat{\boldsymbol{\mu}}^{(3)}$ and $\hat{\boldsymbol{\mu}}^{(4)} \in \mathbb{R}^{3 \times 5}$ are sequentially computed as

$$\hat{\boldsymbol{\mu}}^{(2)} = \begin{pmatrix} 0.388 & 0.490 & 0.759 & 0.902 & 0.925 \\ 0.449 & 0.580 & 0.935 & 1.145 & 1.174 \\ 0.476 & 0.608 & 0.974 & 1.179 & 1.210 \end{pmatrix} \quad (3.13)$$

$$\hat{\boldsymbol{\mu}}^{(3)} = \begin{pmatrix} 0.571 & 0.715 & 1.093 & 1.282 & 1.314 \\ 0.686 & 0.893 & 1.452 & 1.793 & 1.840 \\ 0.700 & 0.887 & 1.407 & 1.687 & 1.730 \end{pmatrix} \quad (3.14)$$

$$\hat{\boldsymbol{\mu}}^{(4)} = \begin{pmatrix} 0.746 & 0.925 & 1.396 & 1.618 & 1.657 \\ 0.932 & 1.221 & 2.002 & 2.490 & 2.555 \\ 0.915 & 1.150 & 1.806 & 2.144 & 2.198 \end{pmatrix} \quad (3.15)$$

Figure 3.3 describes extracted features $\hat{\boldsymbol{\mu}}^{(4)}$ of the truss. Each column of $\hat{\boldsymbol{\mu}}^{(4)}$ corresponds to the feature of each member. It should be noted again that the member features can be represented by vectors of the same size $n_f = 3$ regardless of the connection relationship of the members. If $\hat{\boldsymbol{\mu}}^{(10)}$ and $\hat{\boldsymbol{\mu}}^{(100)}$ are to be computed, the results are as follows:

$$\hat{\boldsymbol{\mu}}^{(10)} = \begin{pmatrix} 1.624 & 1.899 & 2.630 & 2.767 & 2.826 \\ 2.595 & 3.503 & 5.936 & 7.601 & 7.806 \\ 1.997 & 2.387 & 3.512 & 3.884 & 3.975 \end{pmatrix} \quad (3.16)$$

$$\hat{\boldsymbol{\mu}}^{(100)} = \begin{pmatrix} 37.432 & 30.744 & 2.962 & 2.853 & 2.910 \\ 20.668 & 36.539 & 212.93 & 578.993 & 598.968 \\ 40.205 & 33.136 & 4.184 & 4.205 & 4.295 \end{pmatrix} \quad (3.17)$$

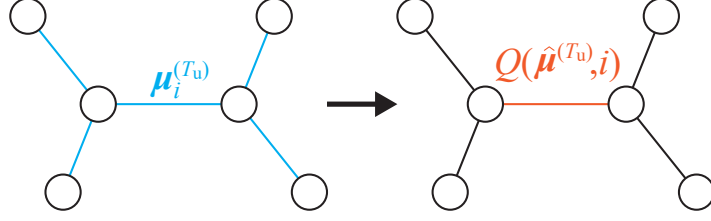


Figure 3.4: The concept of Eq. (3.18). It converts the extracted member features into the action value.

3.3 Q-learning using the embedded features

Using $\hat{\boldsymbol{\mu}}^{(T_u)} = [\boldsymbol{\mu}_1^{(T_u)}, \dots, \boldsymbol{\mu}_{n_m}^{(T_u)}] \in \mathbb{R}^{n_f \times n_m}$, the action value to eliminate member i in the current state is approximated using trainable parameters $\boldsymbol{\theta}_7 \in \mathbb{R}^{2n_f}$, $\boldsymbol{\theta}_8 \in \mathbb{R}^{n_f \times n_f}$, and $\boldsymbol{\theta}_9 \in \mathbb{R}^{n_f \times n_f}$ as follows:

$$Q(\hat{\boldsymbol{\mu}}^{(T_u)}, i) = \boldsymbol{\theta}_7^\top \left(\text{ReLU} \left[\boldsymbol{\theta}_8 \sum_{i=1}^{n_m} \boldsymbol{\mu}_i^{(T_u)}; \boldsymbol{\theta}_9 \boldsymbol{\mu}_i^{(T_u)} \right] \right) \quad (3.18)$$

where $[\cdot; \cdot]$ is a concatenation operator of two vectors or matrices in the column direction. The operation in Eq. (3.18) is illustrated in Fig. 3.4. With Eq. (3.18), the vector that represents the member feature is converted to a scalar that represents the action value to choose the member. The feature $\boldsymbol{\theta}_9 \boldsymbol{\mu}_i^{(T_u)}$ is the local information about member i , although the peripheral nodes and members are taken into consideration. Therefore, the term $\boldsymbol{\theta}_8 \sum_{i=1}^{n_m} \boldsymbol{\mu}_i^{(T_u)}$ is further arranged to capture the global information about the structure by aggregating local information about each member. The term $\boldsymbol{\theta}_8 \sum_{i=1}^{n_m} \boldsymbol{\mu}_i^{(T_u)}$ also plays an important role in estimating appropriate action values for structures of various scales.

As well as $\hat{\boldsymbol{\mu}}^{(T_u)}$, Eq. (3.18) is reformulated so that $\hat{\mathbf{Q}}(\hat{\boldsymbol{\mu}}^{(T_u)}) = [Q(\hat{\boldsymbol{\mu}}^{(T_u)}, 1), \dots, Q(\hat{\boldsymbol{\mu}}^{(T_u)}, n_m)] \in \mathbb{R}^{1 \times n_m}$ is obtained at once, which is given as

$$\hat{\mathbf{Q}}(\hat{\boldsymbol{\mu}}^{(T_u)}) = \boldsymbol{\theta}_7^\top \left(\text{ReLU} \left[\boldsymbol{\theta}_8 \hat{\boldsymbol{\mu}}_\Sigma^{(T_u)}; \boldsymbol{\theta}_9 \hat{\boldsymbol{\mu}}^{(T_u)} \right] \right) \quad (3.19)$$

where $\hat{\boldsymbol{\mu}}_\Sigma^{(T_u)} \in \mathbb{R}^{n_f \times n_m}$ is the matrix in which a column vector $\sum_{i=1}^{n_m} \boldsymbol{\mu}_i^{(T_u)}$ is repeated in the row direction for n_m times.

Since $\hat{\boldsymbol{\mu}}$ is computed using $\{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_6\}$, the set of action values $\hat{\mathbf{Q}}(\hat{\boldsymbol{\mu}})$ is dependent on $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_9\}$. In the following, the training method for tuning the parameters $\boldsymbol{\Theta}$ is described. The parameters $\boldsymbol{\Theta}$ are tuned using a method based on 1-step Q-learning, which is frequently used as an RL method. Here the same equation as Eq. (2.15) is shown below again. When a function approximator is not utilized, the action value is updated using state s , chosen action a , observed next state s' and reward r as

$$Q(s, a) = Q(s, a) + \alpha \left(r(s') + \gamma \max_a Q(s', a) - Q(s, a) \right) \quad (3.20)$$

In Eq. (3.20), the action value is updated so as to minimize the difference between the sum of the observed reward and estimated action value at the next

state $r(s') + \gamma \max_a Q(s', a)$ and the estimated action value at the previous state $Q(s, a)$. Following this scheme, the parameters are trained by solving the following optimization problem [26]:

$$\text{minimize } \tilde{L}(\Theta) = \left(r(s') + \gamma \max_{\tilde{a} \in \Omega_a(s')} Q(s', \tilde{a}; \tilde{\Theta}) - Q(s, a; \Theta) \right)^2 \quad (3.21)$$

where $\Omega_a(s')$ is a set of actions available at state s' . In Eq. (3.21), the training can be stabilized by using previous parameters $\tilde{\Theta}$ obtained during the training for an estimation of action values at the next state s' [26]. $\tilde{\Theta}$ synchronizes with Θ once every n_{sy} episodes, where n_{sy} is an arbitrary positive integer. In the same manner as NNs, the inputs are repeatedly weighted by the trainable parameters and processed through activation functions. Therefore, a back-propagation method can be used for solving Eq. (3.21). Among a number of back-propagation methods proposed so far, RMSProp [120] which is already explained in Eq. (2.20) is adopted as an optimization method in this study.

Example: computation of action values of a truss

In the following, the same truss as Fig. 3.2 is discussed. We suppose $\theta_7 \in \mathbb{R}^6$, $\theta_8 \in \mathbb{R}^{3 \times 3}$ and $\theta_9 \in \mathbb{R}^{3 \times 3}$ as

$$\theta_7 = (0.71 \quad 0.72 \quad 0.73 \quad -0.74 \quad -0.75 \quad -0.76) \quad (3.22a)$$

$$\theta_8 = \begin{pmatrix} 0.081 & 0.082 & 0.083 \\ 0.084 & 0.085 & 0.086 \\ 0.087 & 0.088 & 0.089 \end{pmatrix} \quad (3.22b)$$

$$\theta_9 = \begin{pmatrix} 0.091 & 0.092 & 0.093 \\ 0.094 & 0.095 & 0.096 \\ 0.097 & 0.098 & 0.099 \end{pmatrix} \quad (3.22c)$$

$\hat{\mu}_\Sigma^{(4)}$ is given from $\hat{\mu}^{(4)}$, which is already obtained in Eq. (3.15), as follows:

$$\hat{\mu}_\Sigma^{(4)} = \begin{pmatrix} 6.342 & 6.342 & 6.342 & 6.342 & 6.342 \\ 9.200 & 9.200 & 9.200 & 9.200 & 9.200 \\ 8.211 & 8.211 & 8.211 & 8.211 & 8.211 \end{pmatrix} \quad (3.23)$$

Using the above values and Eq. (3.19), $\hat{\mathbf{Q}}(\hat{\mu}^{(4)}) \in \mathbb{R}^5$ is computed as

$$\hat{\mathbf{Q}}(\hat{\mu}^{(4)}) = (3.812 \quad 3.661 \quad 3.253 \quad 3.029 \quad 2.995) \quad (3.24)$$

Each element of $\hat{\mathbf{Q}}(\hat{\mu}^{(4)})$ corresponds to the action value to choose the member, as illustrated in Fig. 3.5. In this example, choosing the first member is expected to obtain the largest cumulative reward, and choosing the fifth member is expected to obtain the least cumulative reward.

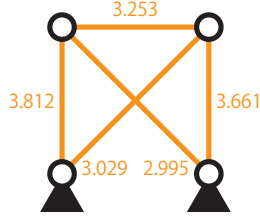


Figure 3.5: Action values $\hat{Q}(\hat{\mu}^{(4)})$ of the example truss

3.4 Training workflow

The whole training workflow is described in Fig. 3.6. At first, parameters Θ and $\tilde{\Theta}$ are randomly initialized with a normal distribution with 0 for the mean and 0.05 for the standard deviation, which is very important to avoid symmetry within each parameter [133]; if the parameters are initialized with the same value and a deterministic learning algorithm is adopted, the gradients of the loss function with respect to the parameters may become the same value, and the parameters with the same gradients will be updated in the same way. The synchronization frequency n_{sy} is fixed as 100 in this study.

An episode is defined as a sequence of design changes to the members from the initial state to the terminal state or step. At the beginning of each episode, structural conditions are randomly provided so that the agent can demonstrate good performance for various conditions. At each step, the agent selects an action to apply the design change to the member of the structure, using an ϵ -greedy policy. Although ϵ may vary depending on the RL task to obtain better results, ϵ is fixed as 0.1 in this study. After taking an action, the agent observes the reward r , the next state s' represented by $\hat{\mathbf{v}}'$ and $\hat{\mathbf{w}}'$, and a set of possible actions at the next state $\Omega_a(s')$. To reduce a required capacity of a storage device, 1000 sets of observed transitions $(\hat{\mathbf{v}}, \hat{\mathbf{w}}, a, \hat{\mathbf{v}}', \hat{\mathbf{w}}', r, \Omega_a(s'))$ are stored at the maximum. When the number of transition steps reaches 1000, the latest transition overrides the oldest one.

Between the steps, the set of trainable parameters Θ is optimized using RMSProp as follows; 32 datasets out of the stored transitions are randomly chosen to create a minibatch and Θ is updated based on the mean squared error of the loss function of each dataset computed by the right-hand side of Eq. (3.21). The size of edge feature n_f , the number of training episodes n_{ep} , the learning rate α , and the discount rate γ are set depending on the RL task, which are mentioned in the case studies in Chapters 4 and 5. Not only n_f as explained in Sec. 3.2, n_{ep} and α are also very important hyper-parameters determined through trial-and-error. When a smaller value is assigned to α , a larger value for n_{ep} is required for the RL agent to perform well; on the other hand, if α is too large, not only will the learning become unstable, but trainable parameters Θ may diverge.

Once in 10 episodes of the training, the performance of Θ is tested for a prescribed condition. The cumulative reward until the terminal state is recorded using the greedy policy without randomness (i.e. the ϵ -greedy policy with $\epsilon = 0$) during the test. If the cumulative reward is larger than the previous best score, Θ at that step is saved. The most recently saved Θ after the training is regarded

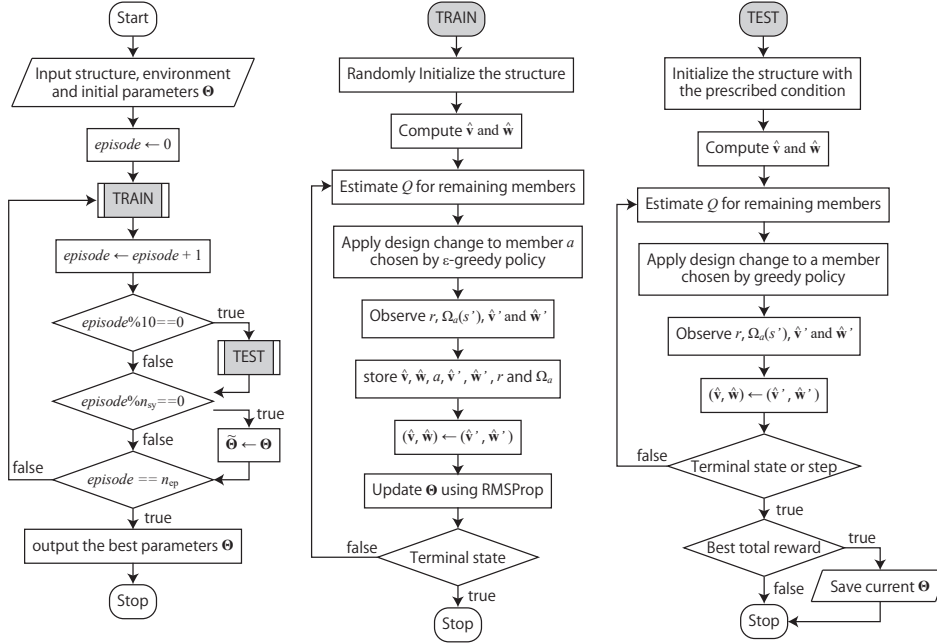


Figure 3.6: Training workflow utilizing RL and GE

as the best parameters.

3.5 Conclusion

In this chapter, the hybrid method of GE and RL has been proposed. The state of the skeletal structure is expressed by member features, which are computed through GE from node and member inputs. The trainable parameters that change during the training play a role in weighting the inputs. By adjusting these parameters, the agent becomes capable of obtaining member features grasping the overall structural properties from the local inputs of nodes and members of the structure. Each member feature is expressed as a vector of the same size for all the members.

Considering that there are few research examples of edge embedding compared to whole graph embedding and node embedding, and that only node inputs can be handled even in the recent edge embedding method [57], it is a major highlight of the proposed edge embedding method that the inputs of both nodes and members can be handled at the same time.

The feature vector is finally converted to a scalar value that represents an action value of applying a pre-specified design change to the member. Since the feature and action value of the members can be calculated using the common trainable parameters regardless of the member connectivity, the trained agent can be applied without re-training to other skeletal structures with different connectivity and numbers of nodes and members.

Chapter 4

Case study 1: Topology optimization of trusses

4.1 Topology optimization problem of planar trusses considering stress constraint

In this section, the topology optimization problem of planar trusses is formulated for minimizing the total structural volume under stress and displacement constraints. The optimization problem becomes the RL task to be handled in Secs. 4.2 - 4.4. Note that the proposed method belongs to a *binary* type approach where the cross-sectional area of each member either keeps its initial value, or takes a very small value.

Displacements of nodes and axial stresses of members are computed for n_L static loading conditions using a standard stiffness method. Consider a planar truss with the total number of degrees of freedom (DOFs) n_d , the number of members n_m and the constant elastic modulus E . Let A_i and L_i denote the cross-sectional area and the length of member i . If only axial stiffness is considered for the member stiffness, the member stiffness matrix with respect to the local coordinates of member i is a matrix $\mathbf{K}_i \in \mathbb{R}^{4 \times 4}$ where (1, 1) and (3, 3) elements are EA_i/L_i , (1, 3) and (3, 1) elements are $-EA_i/L_i$ and the others are 0. Member stiffness matrices of all the members are transformed into the global coordinate system to be assembled to the global stiffness matrix $\mathbf{K} \in \mathbb{R}^{n_d \times n_d}$. Using \mathbf{K} , the nodal displacements are obtained by solving the following stiffness equation [134]:

$$\mathbf{K}\mathbf{u}_j = \mathbf{p}_j \quad (4.1)$$

where $\mathbf{u}_j \in \mathbb{R}^{n_d}$ and $\mathbf{p}_j \in \mathbb{R}^{n_d}$ are the nodal displacement and nodal load vectors corresponding to load case $j \in \{1, \dots, n_L\}$. Let $\sigma_{i,j}$ denote the axial stress of member i for load case j , which is expressed as

$$\sigma_{i,j} = \frac{Ed_{i,j}}{L_i} \quad (4.2)$$

where $d_{i,j}$ is the elongation of member i for load case j , which is simply calculated after obtaining the nodal displacements.

Let $\mathbf{A} = \{A_1, \dots, A_{n_m}\}$ denote the vector of cross-sectional areas. This research aims to obtain the optimal topology of a truss that minimizes the total structural volume $V_s(\mathbf{A})$ under stress constraints. However, some solutions to this problem are trivial; for example, if all the members connecting to supporting nodes are eliminated, the structure cannot resist external loads at all while the stress constraints for existing members are satisfied. For this reason, displacement constraints are further added to avoid apparently infeasible trusses that deform excessively. By assigning the upper-bound stress $\bar{\sigma}$ and the upper-bound displacement \bar{u} , the optimization problem is formulated as follows:

$$\text{minimize } V_s(\mathbf{A}) \quad (4.3a)$$

$$\text{subject to } \max_{i \in \Omega_m, j \in \{1, \dots, n_L\}} \left(\frac{|\sigma_{i,j}(\mathbf{A})|}{\bar{\sigma}} \right) \leq 1 \quad (4.3b)$$

$$\max_{i \in \Omega_d, j \in \{1, \dots, n_L\}} \left(\frac{|u_{i,j}(\mathbf{A})|}{\bar{u}} \right) \leq 1 \quad (4.3c)$$

$$A_i \in \{\bar{A} \times 10^{-6}, \bar{A}\} \quad (i = 1, \dots, n_m) \quad (4.3d)$$

where Ω_m and Ω_d are the sets of indices of existing members and DOFs of existing nodes including loaded nodes and $u_{i,j}$ is the i -th displacement component for load case j . Hereafter, the ratio $|\sigma_{i,j}|/\bar{\sigma}$ is called stress safety ratio, which indicates a safe state if it is small. The cross-sectional areas are chosen from \bar{A} for existing members and a small value $\bar{A} \times 10^{-6}$ instead of 0 for non-existing members to prevent singularity of the stiffness matrix \mathbf{K} in Eq. (4.1).

4.2 Conversion to a reinforcement learning task

The optimization problem Eq. (4.3) is converted to a RL task in this section. First, a state is represented by a set of numerical data at nodes $\mathbf{v} = \{\mathbf{v}_1, \dots, \mathbf{v}_{n_n}\}$ and that at members $\mathbf{w} = \{\mathbf{w}_1, \dots, \mathbf{w}_{n_m}\}$, as described in Tables 4.1 and 4.2 based on the loading and support conditions, the geometry of the truss, and the stress safety ratios. In Table 4.1, the load intensity is defined as the magnitude of the external load applied at the node, which is decomposed into x and y directions; the magnitude in z direction is omitted because only planar trusses are focused in this example. In Table 4.2, the orientation of members is expressed by trigonometric quantities instead of the angle itself so as to avoid discontinuity when the angles are 0 and 2π .

In this case study, the action is defined as eliminating a member from existing ones. By repeating the actions, a sparse topology is obtained from a densely connected initial GS. Note again that the members regarded as removed virtually have a very small cross-sectional area as seen in Eq. (4.3d) to avoid instability in computing nodal displacements. Any state-action pair deterministically leads to a unique next state without random transitions to the other states. The reward is evaluated after each removal of a member. When the truss violates the stress or displacement constraints, a reward of -1 is provided and the removal process is terminated. Otherwise, the reward computed by the following equation is provided:

Table 4.1: Detail of node input \mathbf{v}_k

index	input
1	1 if pin-supported, 0 otherwise
2	load intensity [kN] at the node in x direction (load case 1)
3	load intensity [kN] at the node in y direction (load case 1)
\vdots	\vdots
$2n_L$	load intensity [kN] at the node in x direction (load case n_L)
$2n_L + 1$	load intensity [kN] at the node in y direction (load case n_L)

Table 4.2: Detail of member input \mathbf{w}_i

index	description
1	the cosine of the angle formed by the member with respect to the positive x -direction
2	the sine of the angle formed by the member with respect to the positive x -direction
3	member length [m]
4	1 if remained, 0 if removed
5	stress safety ratio (load case 1)
\vdots	\vdots
$n_L + 4$	stress safety ratio (load case n_L)

$$r = L_e \left(1 - \max_{i \in \Omega_m, j \in \{1, \dots, n_L\}} \frac{|\sigma_{i,j}|}{\bar{\sigma}} \right) \quad (4.4)$$

where L_e is the length of the eliminated member. Equation (4.4) is derived considering the total structural volume to be minimized and the stress safety ratio to be constrained; accordingly, the reward is large when the eliminated member is long and the maximum stress safety ratio of the resulting topology is small.

4.3 Training setting

The detail of the training implementation is explained in this section. The agent is trained using a 72-member truss as shown in Fig. 4.1. The truss has 4×4 grids, and each grid is a 1m square. The bracing members are not connected at the intersection. The upper-bound stress $\bar{\sigma}$ is 200 N/mm² for both tension and compression. The elastic modulus E is 200 kN/mm² and the initial cross-section A_i is 1000mm². At the beginning of each episode, two pin-supports are randomly chosen; one from nodes 1 and 2, and the other from nodes 4 and 5. The number of load cases n_L is 2. A horizontal or vertical load with a fixed magnitude of 1.0 kN is applied at a node randomly chosen from right tip nodes for each loading condition.

Note that the loading conditions may be identical, loaded nodes may be the same but the load directions are different, or the load directions may be

the same but loaded nodes are different between the two load cases. Thus, the training concerns a total of $2 \times 2 \times 20 \times 20 = 1600$ combinations of support and loading conditions, and these combinations are almost equally simulated as long as the number of training episodes is sufficient. The topology that does not satisfy stress or/and displacement constraints is regarded as the terminal state. The episode ends when the terminal state is reached, and then a new episode starts with all the members of the initial GS reset as existing and the support/load conditions re-randomized. The number of training episodes n_{ep} is set to be 5000. In this example, the training is implemented with different learning rates α , discount rates γ and the sizes of the member feature n_f in order to investigate the effect of these values on the training results.

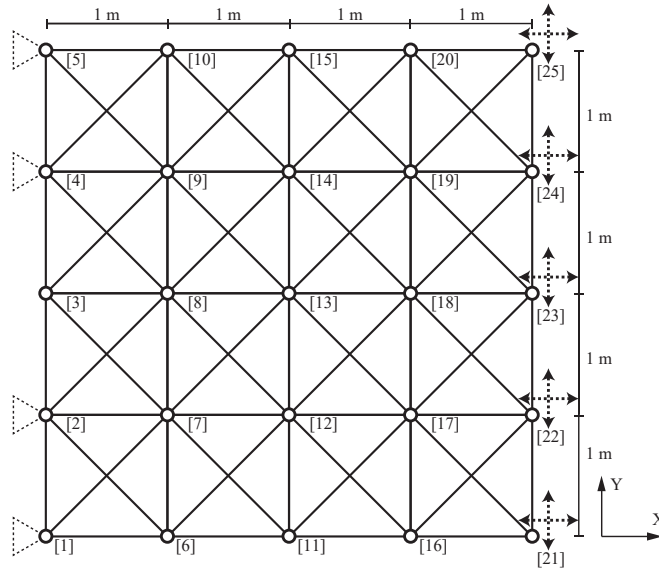


Figure 4.1: 4×4 -grid truss used for training ($V_s = 0.0853 \text{ [m}^3\text{]}$)

During the test, nodes 1 and 5 are pin-supported, and loads are applied at node 23 in positive x and negative y directions separately as different loading conditions, which is denoted as loading condition L1.

4.4 Training result

4.4.1 Training history and performance for 4×4 -grid truss

Figure 4.2 plots the histories of the obtained cumulative rewards in the test simulations recorded at every 10 episodes for the trainings with three different learning rates: $\alpha = 1.0 \times 10^{-5}$, $\alpha = 5.0 \times 10^{-5}$ and $\alpha = 1.0 \times 10^{-3}$ while fixing γ as 0.99 and n_f as 100. When $\alpha = 1.0 \times 10^{-5}$, the cumulative reward earned by the agent have steadily increased as the number of training episodes increases, but the rate of increase was slow. In contrast, the history of obtained cumulative reward fluctuates when $\alpha = 1.0 \times 10^{-3}$. In this example, $\alpha = 5.0 \times 10^{-5}$ is the

best setting because the performance improvement is stable and fast. The score improved significantly in the first 1000 episodes and has remained stable mostly above 35.0 since then. Therefore, α is fixed at 5.0×10^{-5} in the following. For $\alpha = 5.0 \times 10^{-5}$, it took about 8.6 hours for the training through 232810 linear structural analyses with Intel(R) Core(TM) i7-8850H CPU @ 2.60GHz. The maximum test score of 42.9 is obtained at the 1400th episode.

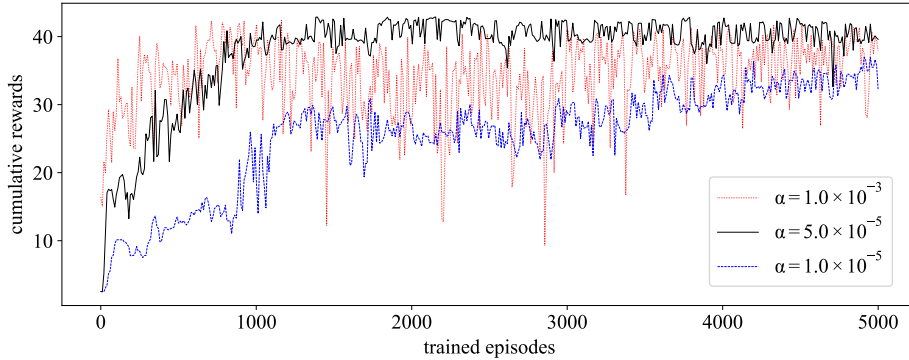


Figure 4.2: Histories of the cumulative reward of each test measured every 10 episodes ($\gamma = 0.99$, $n_f = 100$)

Next, the difference in training results was investigated by varying the size of the extracted feature: $n_f = 3$, $n_f = 10$, $n_f = 100$ and $n_f = 200$, while fixing γ as 0.99 and α as 5.0×10^{-5} . The training histories are shown in Fig. 4.3. $n_f = 3$ is clearly an appropriate setting for this problem; the cumulative reward earned have peaked around 15. Although the training takes a relatively shorter time for $n_f = 10$, the learning is unstable because the cumulative reward obtained for each test varied widely and it required more episodes to obtain the cumulative reward comparable to the other results. The learning histories are very similar between the two cases $n_f = 100$ and $n_f = 200$, but when the feature size is $n_f = 200$, the required training time is more than three times longer than when the feature size is $n_f = 100$. Note that the total size of trainable parameters in Θ is 90 for $n_f = 3$, 720 for $n_f = 10$, 61200 for $n_f = 100$ and 242000 for $n_f = 200$. From Fig. 4.3, $n_f = 100$ is adopted as the best hyper-parameter in the following.

The removal sequence of the members when the maximum score is recorded is illustrated in Fig. 4.4. Trivial members close to the pin-supports or placed upper-right or bottom-right that do not bear stress are first removed, and important members along the load paths are removed afterward. This is evidence that the agent is capable of detecting the load paths among the members, and this capability is owing to the proposed GE method that is capable of extracting member features considering the truss connectivity.

The final truss in the removal process of members presented in Fig. 4.4(b) is the terminal state, where the nodal displacements are excessive due to the unstable connectivity. Therefore, the topology just before the terminal state surrounded by a red square in Fig. 4.4(b) shall be a sub-optimal topology, which is a truss with 12 members and $V_s = 0.0145$. Since the removal of any remaining member will cause a violation of the displacement constraints, there is no unnecessary member in the sub-optimal topology. It forms a very simple truss composed of six pairs of members connecting linearly. Note that the connection

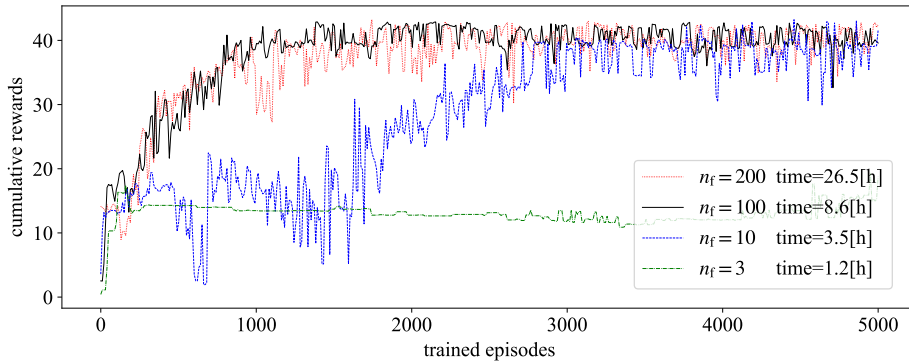
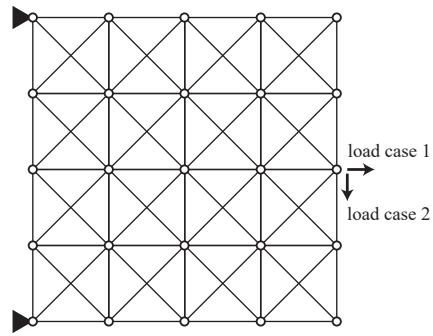


Figure 4.3: Histories of the cumulative reward of each test measured every 10 episodes ($\gamma = 0.99$, $\alpha = 5.0 \times 10^{-5}$)

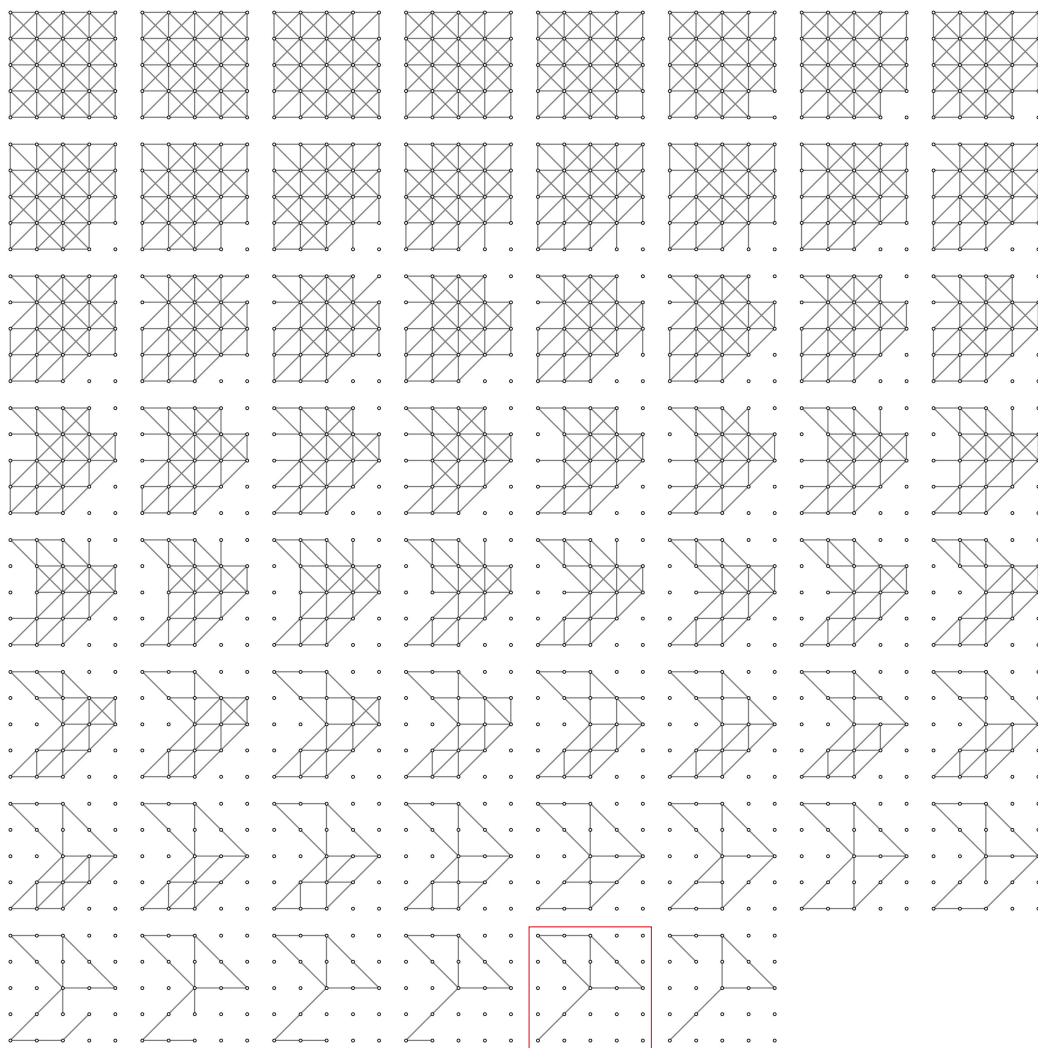
between the members in each pair is an unstable node, and must be fixed to generate a single long member.

In utilizing the trained agent, n_{load} ! different removal sequences can be obtained for a different order of the same set of load cases in node input data \mathbf{v}_k ; for example, exchanging the values at indices 2 and 4 and those at indices 3 and 5 in \mathbf{v}_k maintains the original loading condition but may lead to different actions to be taken during the member removal process because the trained agent estimates different action values due to the exchange. As shown in Fig. 4.5, a different removal sequence of the members is observed with the reverse order of the load cases.

We further investigated the performance of RL agents trained with another discount rate: $\gamma = 0.9$. After training with the same settings except for the discount rate, the RL agent showed other removal sequence of members as shown in Figs. 4.6 and 4.7. From these results, the RL agent trained with $\gamma = 0.9$ acquired a short-sighted policy compared with the agent trained with $\gamma = 0.99$, because the agent apparently first removes the diagonal members and then removes the horizontal and vertical members. In this case, where rewards are given according to the length of the member, removing the diagonal member in the initial steps will certainly give a lot of rewards first, but the cumulative reward will not necessarily increase.

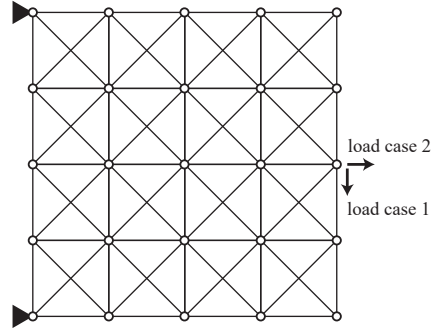


(a)

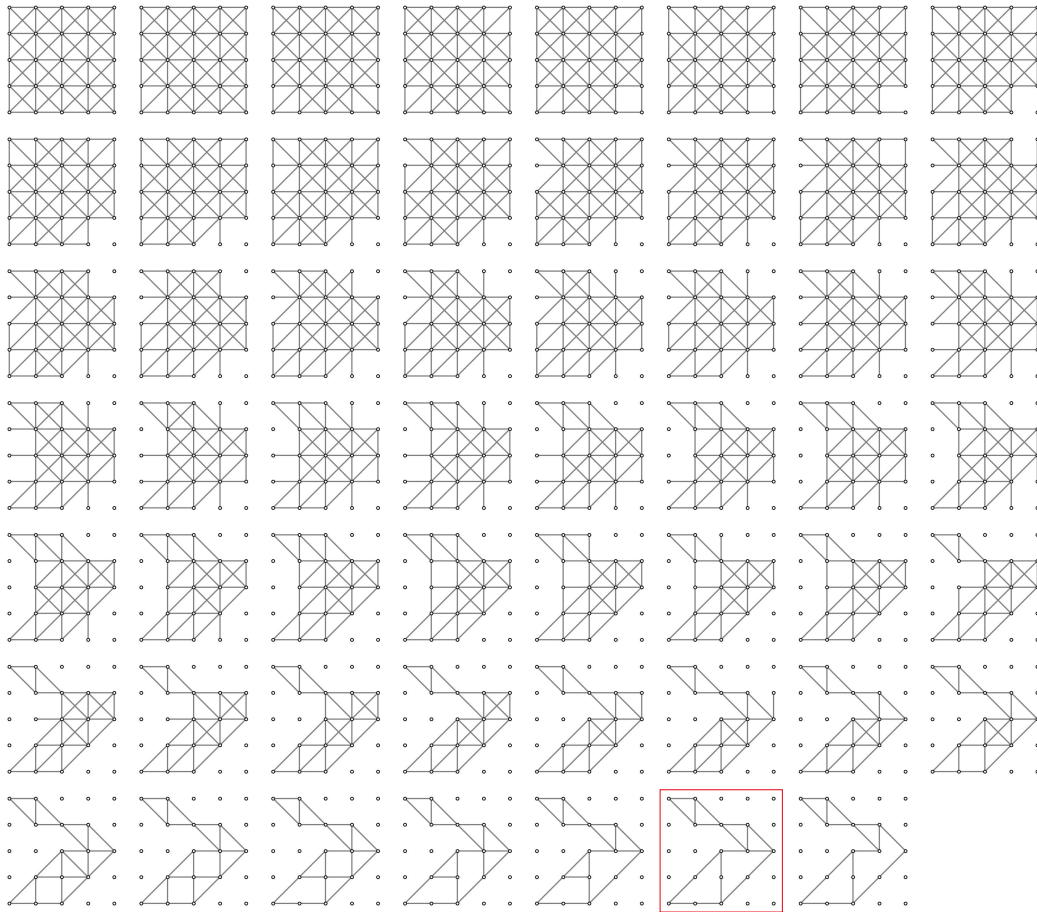


(b)

Figure 4.4: Best scored removal process of members for loading condition L1 of 4×4 -grid truss ($\gamma = 0.99$). (a) Initial GS. (b) Removal sequence of members.

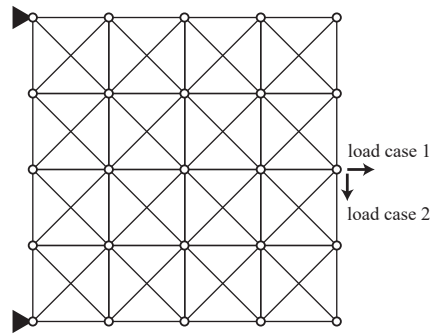


(a)

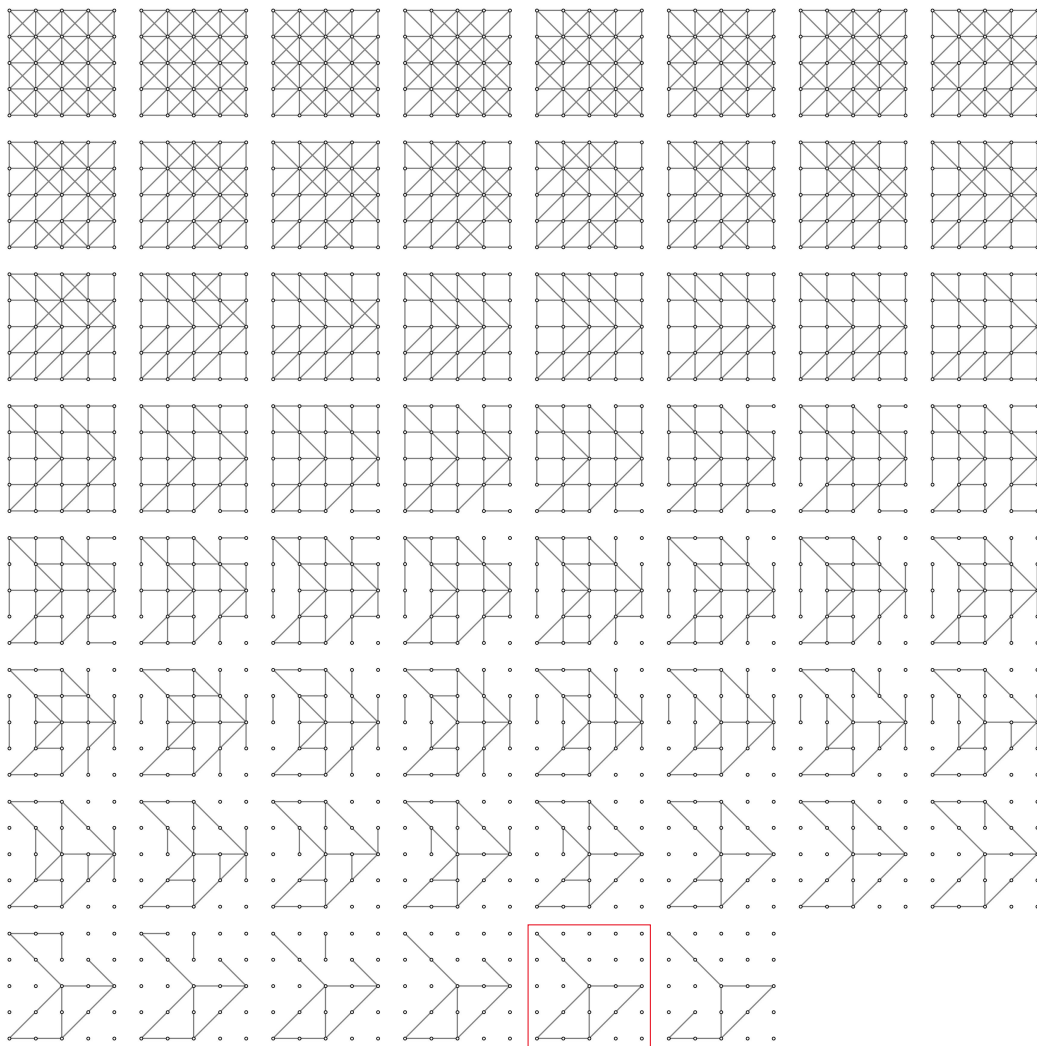


(b)

Figure 4.5: Loading condition L1 of 4×4 -grid truss (reversed the order of load cases, $\gamma = 0.99$). (a) Initial GS. (b) Removal sequence of members.

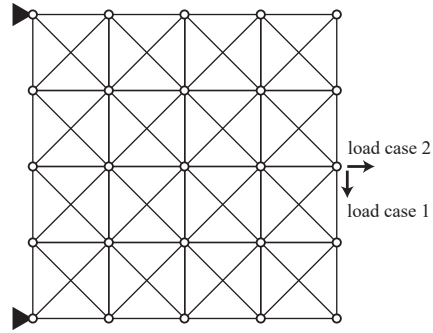


(a)

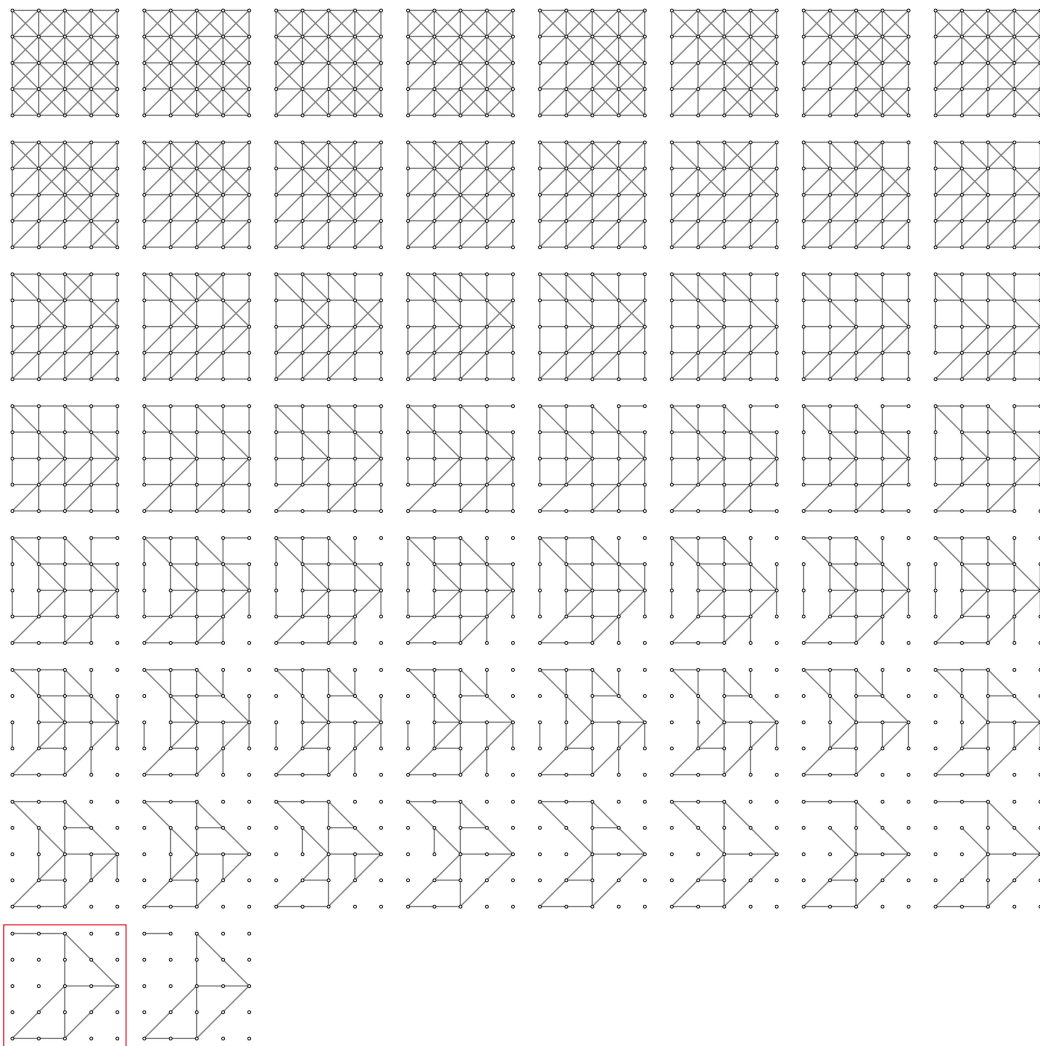


(b)

Figure 4.6: Loading condition L1 of 4×4 -grid truss ($\gamma = 0.9$). (a) Initial GS. (b) Removal sequence of members.



(a)

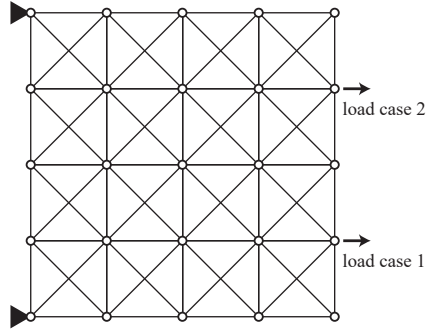


(b)

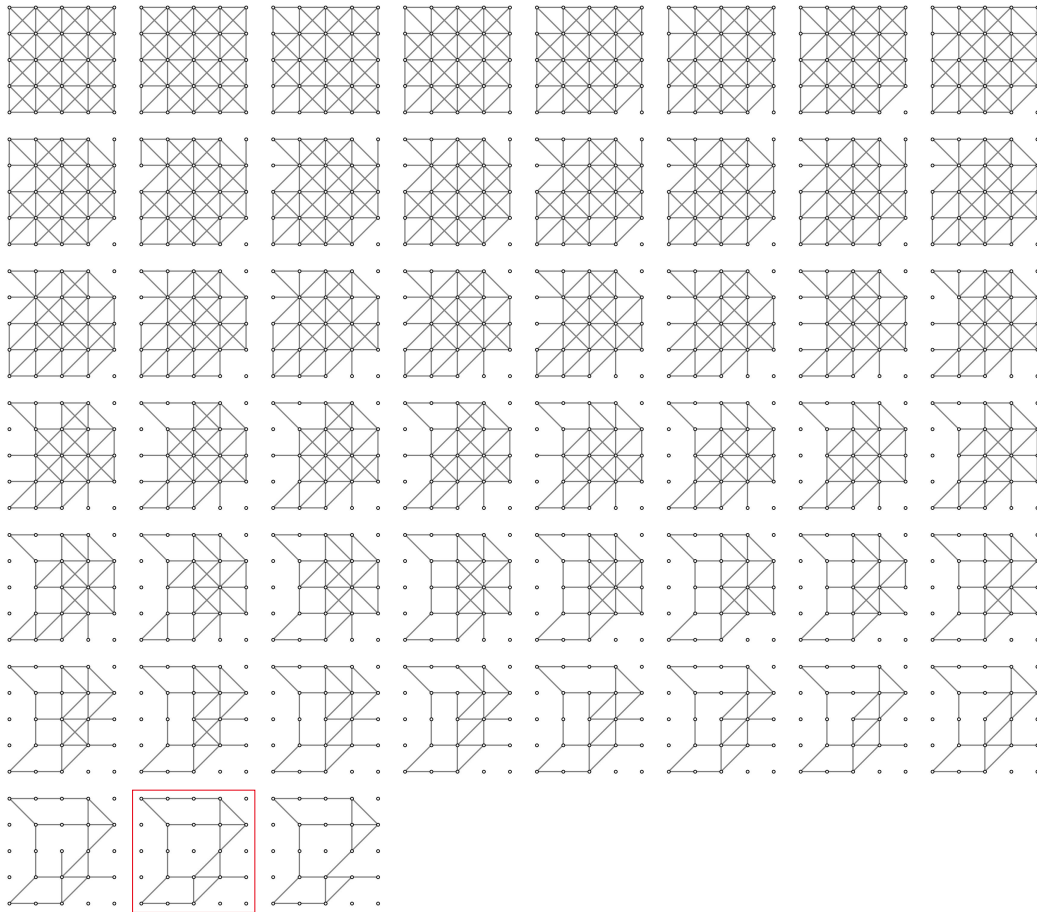
Figure 4.7: Loading condition L1 of 4×4 -grid truss (reversed the order of load cases, $\gamma = 0.9$). (a) Initial GS. (b) Removal sequence of members.

To investigate the performance of the RL agents in another loading condition, the structure with the loads as shown in Fig. 4.8(a), denoted as loading condition L2, is also optimized using the same RL agents trained with $\gamma = 0.99$ and $\gamma = 0.9$. Nodes 1 and 5 are pin-supported and nodes 22 and 24 are subjected to an 1 kN load in the positive x direction separately as two load cases. The removal sequences of members are illustrated in Figs. 4.8 - 4.11. Similarly to loading condition L1 in Fig. 4.4, the sub-optimal topology is a well-converged solution that does not contain unnecessary members. From these results, the agent is confirmed to behave well for a different loading condition.

Furthermore, the robustness of the proposed method is also investigated by implementing 2000-episode training with $\gamma = 0.99$ using different random seeds 20 times. The statistical data with respect to the maximum test scores for each training are as follows; the maximum is 43.41, the median is 43.19, the minimum is 42.77, the average is 43.19, the standard deviation is 0.15, and the coefficient of variation is only 3.55×10^{-3} . Moreover, all the trained RL agents with the best parameters led to 12-member sub-optimal solutions similar to Figs. 4.4 - 4.7 for loading condition L1. These results imply that the proposed method is robust against the randomness of boundary conditions and actions during the training.

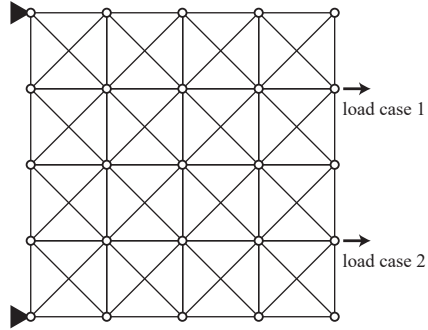


(a)

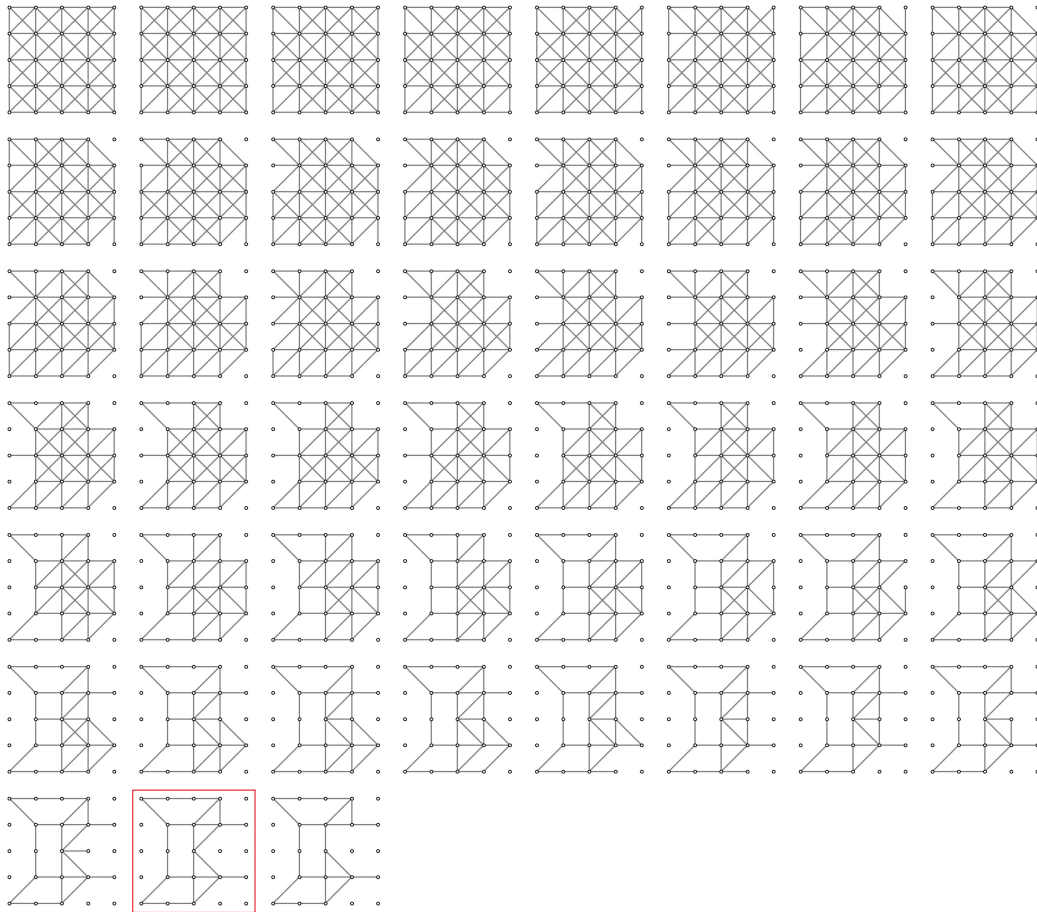


(b)

Figure 4.8: Loading condition L2 of 4×4 -grid truss ($\gamma = 0.99$). (a) Initial GS.
 (b) Removal sequence of members.

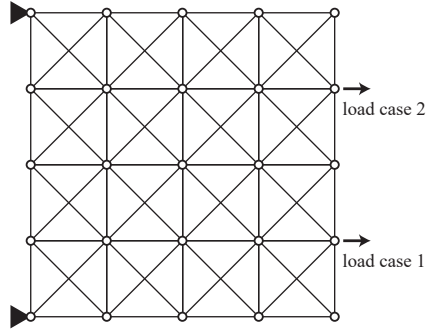


(a)

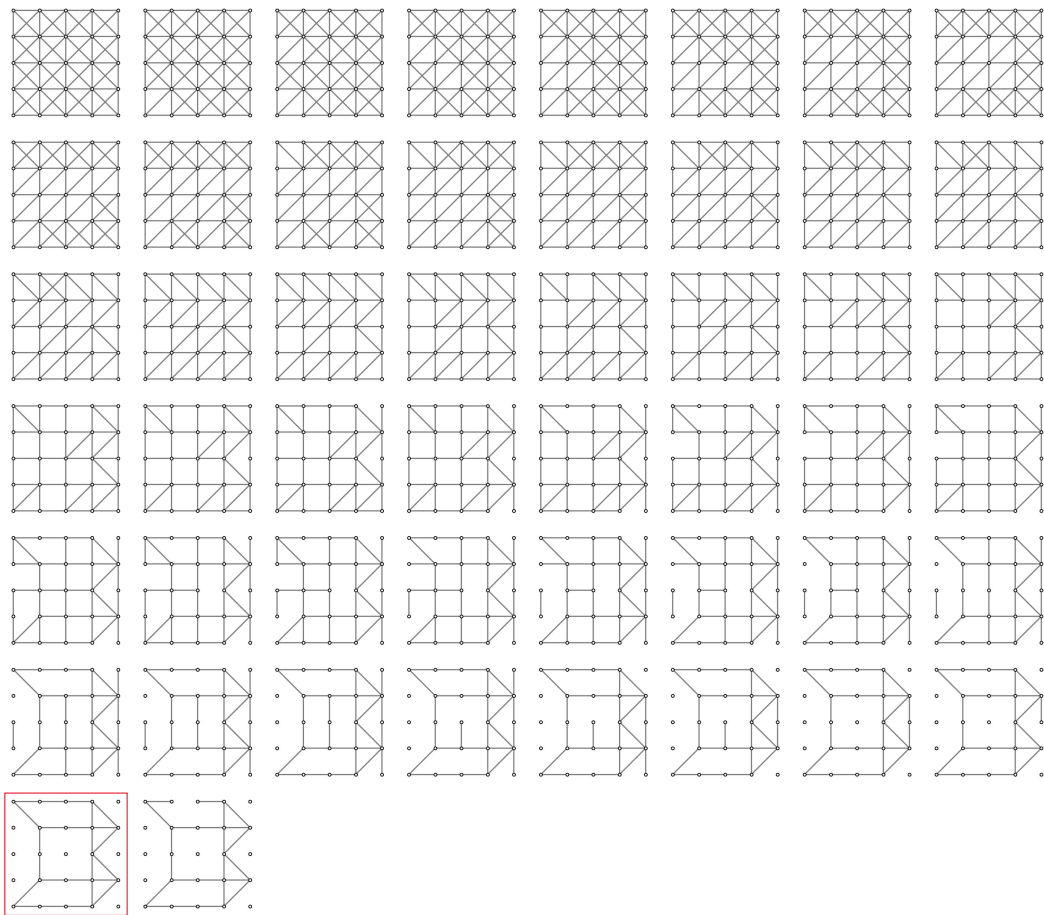


(b)

Figure 4.9: Loading condition L2 of 4×4 -grid truss (reversed the order of load cases, $\gamma = 0.99$). (a) Initial GS. (b) Removal sequence of members.

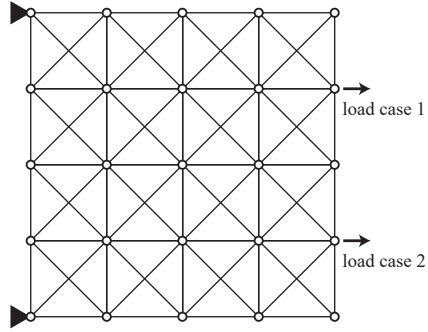


(a)

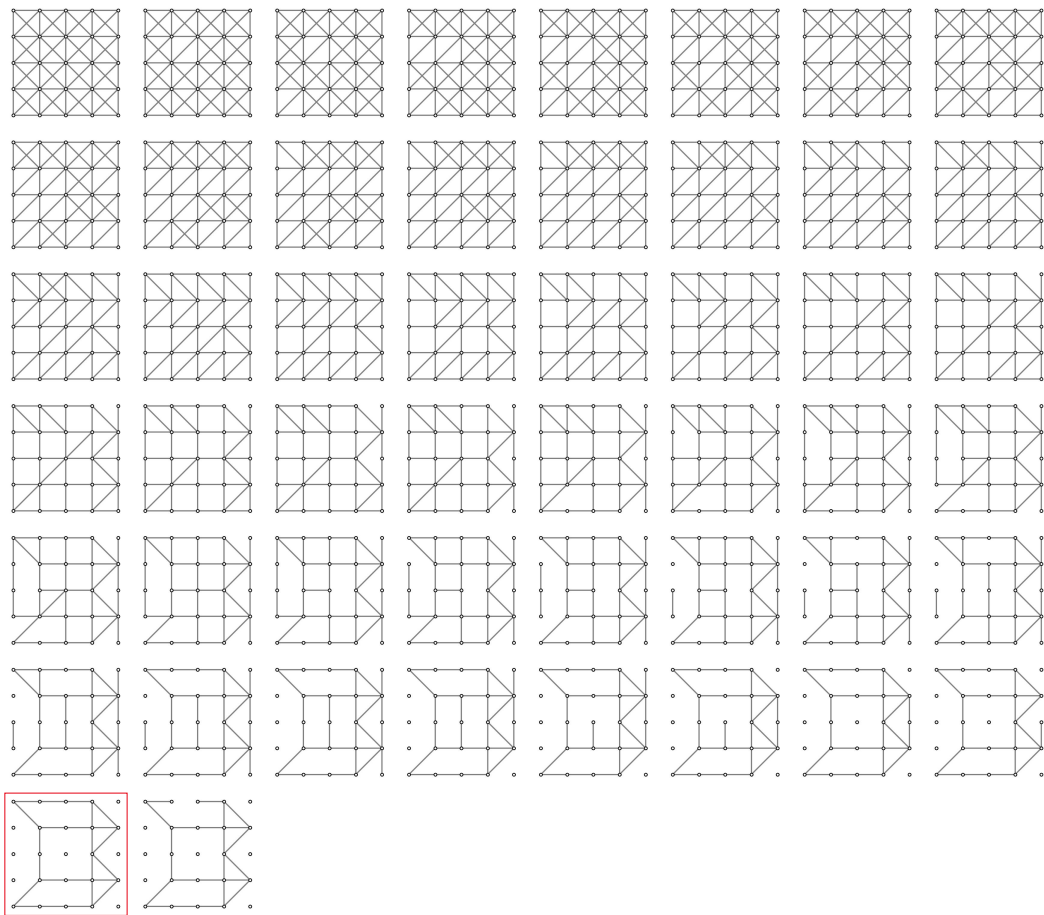


(b)

Figure 4.10: Loading condition L2 of 4×4 -grid truss ($\gamma = 0.9$). (a) Initial GS.
 (b) Removal sequence of members.



(a)



(b)

Figure 4.11: Loading condition L2 of 4×4 -grid truss (reversed the order of load cases, $\gamma = 0.9$). (a) Initial GS. (b) Removal sequence of members.

4.4.2 Investigation of generalization performance 1: 3×2-grid truss

The trained agent with the proposed method is reusable to other trusses with different numbers of nodes and members. In this subsection, the agents trained in Sec. 4.4.1 with $\gamma = 0.99$ and $\gamma = 0.9$ are reused for a smaller 3×2-grid truss without re-training in order to verify the generalization performance of the agents towards a smaller truss. Fig. 4.12 shows the initial GS. As in Example 1, each grid is a square with a side length of 1 m. The topology is optimized for two boundary conditions.

In the first boundary condition B1, as shown in Fig. 4.13(a), left tip nodes 1 and 3 are pin-supported and bottom-right nodes 7 and 10 are subjected to a downward unit load of 1 kN separately as different loading cases. As shown in Fig. 4.13(b), the RL agent trained with $\gamma = 0.99$ utilizes a reasonable policy to eliminate apparently non-load-bearing members in order. From this result, it is confirmed that the agent is capable of eliminating unnecessary members properly for a smaller-scale truss. The topology just before the terminal state is a sub-optimal truss with $V_s = 0.0121[\text{m}^3]$ that efficiently transmits the downward loads to the pin-supports. In Fig. 4.14, another removal sequence of members is observed by exchanging the order of load cases. The sub-optimal truss with $V_s = 0.0175[\text{m}^3]$ contains successive V-shaped braces but node 10 is unstable against a horizontal load.

Similarly, the removal sequences of members by the RL agent trained with $\gamma = 0.9$ are shown in Figs. 4.15 and 4.16. The sub-optimal topology in Fig. 4.15 is the same as Fig. 4.13; on the other hand, the sub-optimal topology in Fig. 4.16 is inferior to the other solutions in the total structural volume. In particular, the 14th member connecting nodes 2 and 3 hangs from the supported node 2 without being connected to other nodes during steps 9-14. According to Table 4.7, the action value to remove the redundant 14th member at the 10th step ($Q = 1.1$) is almost equal to the action value to remove the critical 21st member connecting nodes 5 and 7. This implies that the agent mistakenly understands the importance of the members in this case. In order to quickly remove locally unstable members, one can penalize them in the reward setting to estimate the action value more correctly.

In the second boundary condition B2, the bottom center nodes 4 and 7 are pin-supported and the upper tip nodes 3 and 12 are subjected to outward unit loads along x axis as shown in Fig. 4.17(a). Similarly to the boundary condition B1, the agent eliminates members that do not bear forces as shown in Figs. 4.17 - 4.20. In each result, a tower-like symmetric topology is created with extending members from upper tips to loaded nodes around the 18th step.

Table 4.3: Connectivity of member i ($i = 1, \dots, 29$) of 3 × 2-grid truss

		i														
node		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
start		1	4	7	2	5	8	3	6	9	1	4	7	10	2	5
end		4	7	10	5	8	11	6	9	12	2	5	8	11	3	6
		i														
node		16	17	18	19	20	21	22	23	24	25	26	27	28	29	
start		8	11	1	2	4	5	7	8	2	3	5	6	8	9	
end		9	12	5	4	8	7	11	10	6	5	9	8	12	11	

After the 18th step, the topology became asymmetric in Figs. 4.17 and 4.20 despite the symmetry of the problem definition; however, asymmetric solutions should be accepted because Guo et al. [135] explained that solving a quasi-convex symmetric optimization problem may yield a highly asymmetric solution.

The action values estimated by the RL agents are described in Tables 4.4 - 4.11. It is possible to quantitatively understand which member the agent considers important through the action values; If the action value is small, the agent considers the member corresponding to the action value to be a more necessary member. In particular, if the action value is negative, the member is considered indispensable. As a whole, the action values of removing the remaining members decrease as the number of steps increases. In addition, the members that have not been removed in the sub-optimal topology have smaller action values compared with the members that have been removed. From these results, it was confirmed that the agents were able to estimate the action value based on not only the local information about the member to be removed but also the connection relationship of the entire structure. The agent trained with a learning rate of 0.99 tends to estimate action values larger than the agent trained with a learning rate of 0.9. This is because the former includes the rewards expected to be obtained in further steps in the action value.

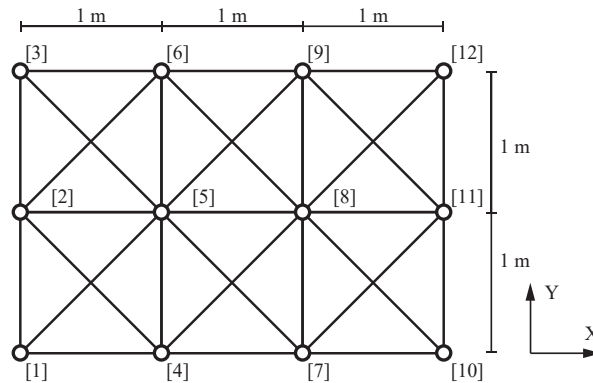


Figure 4.12: 3×2 -grid truss ($V_s = 0.0340 \text{ [m}^3\text{]}$)

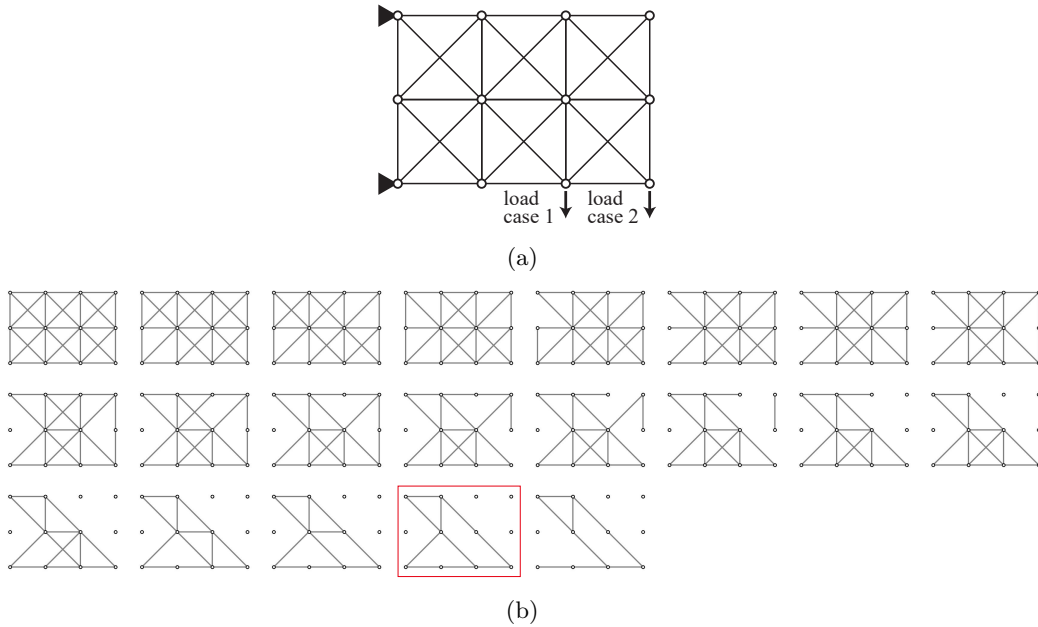


Figure 4.13: Boundary condition B1 of 3×2 -grid truss ($\gamma = 0.99$). (a) Initial GS. (b) Removal sequence of members.

Table 4.4: Action values $Q(s, i)$ ($i = 1, \dots, 29$) at each step of Fig. 4.13(b)

$t \backslash i$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	-2.5	3.8	4.6	5.0	4.8	5.0	0.0	3.9	5.1	4.9	4.7	4.5	4.4	5.1	4.7	4.7	5.0	3.9	5.3	4.8	4.9	5.0	4.9	5.1	4.0	5.0	4.8	5.0	5.2
1	-2.5	3.2	4.3	4.7	4.5	4.7	-0.1	3.7	4.8	4.7	4.3	4.2	4.1	4.8	4.3	4.5	4.7	2.9	-	4.4	4.5	4.7	4.6	4.8	3.6	4.6	4.6	4.8	5.0
2	-2.8	2.5	3.9	4.3	4.1	4.1	-0.3	3.6	4.0	4.3	4.0	3.8	3.7	4.4	4.0	4.0	4.0	2.8	-	4.0	4.1	4.3	4.1	4.4	3.2	4.2	4.1	4.2	-
3	-2.7	2.4	3.7	4.0	3.8	3.9	-0.2	3.0	3.8	4.1	3.7	3.5	3.4	4.1	3.5	3.7	3.8	2.3	-	3.8	3.9	4.0	3.9	-	2.2	3.9	3.8	3.9	-
4	-2.8	2.3	3.5	3.8	3.7	3.8	-0.4	2.9	3.7	4.0	3.6	3.4	3.3	-	3.4	3.6	3.7	2.2	-	3.6	3.8	3.9	3.7	-	2.0	3.7	3.7	3.8	-
5	-2.9	2.1	3.4	3.6	3.5	3.6	-0.5	2.7	3.5	-	3.5	3.2	3.1	-	3.2	3.4	3.5	2.1	-	3.4	3.7	3.7	3.6	-	1.8	3.6	3.5	3.6	-
6	-3.0	1.9	2.5	3.3	3.2	3.3	-0.7	2.3	3.0	-	3.1	2.5	3.2	-	2.8	3.2	2.8	1.8	-	3.1	3.2	-	2.9	-	1.5	3.2	3.2	3.1	-
7	-3.2	1.6	2.1	2.9	2.8	-	-1.0	2.0	2.6	-	2.8	2.1	2.8	-	2.5	2.9	2.3	1.6	-	2.6	2.8	-	2.5	-	1.2	2.9	2.8	2.6	-
8	-3.3	1.4	1.9	-	2.4	-	-1.2	1.7	2.4	-	2.4	1.8	2.5	-	2.1	2.6	2.0	1.2	-	2.3	2.4	-	2.2	-	0.7	2.4	2.5	2.3	-
9	-3.6	1.1	1.6	-	2.0	-	-0.9	1.9	1.9	-	2.1	1.4	2.1	-	1.5	-	1.4	1.1	-	1.6	2.1	-	1.8	-	0.1	2.5	1.4	1.6	-
10	-3.6	0.9	1.3	-	1.7	-	-1.0	1.7	1.5	-	1.9	1.2	1.9	-	1.2	-	1.2	0.8	-	1.4	1.9	-	1.6	-	-0.2	-	1.2	1.4	-
11	-3.7	-0.3	-0.5	-	1.3	-	-0.5	1.7	1.9	-	1.4	1.2	-	-	0.5	-	1.9	0.7	-	1.4	1.3	-	-0.2	-	-0.8	-	-0.0	1.9	-
12	-3.8	-0.4	-0.6	-	1.2	-	-0.6	1.6	-	-	1.4	1.2	-	-	0.4	-	1.7	0.6	-	1.3	1.2	-	-0.3	-	-0.9	-	-0.1	1.8	-
13	-3.8	-0.5	-0.7	-	1.1	-	-0.8	1.4	-	-	1.2	1.1	-	-	0.3	-	1.6	0.4	-	1.1	1.1	-	-0.5	-	-1.1	-	-0.3	-	-
14	-3.8	-0.6	-0.8	-	1.0	-	-0.9	1.3	-	-	1.1	1.0	-	-	0.2	-	-	0.4	-	1.1	1.0	-	-0.6	-	-1.1	-	-0.4	-	-
15	-3.9	-0.7	-0.9	-	0.9	-	-0.9	-	-	-	1.0	0.9	-	-	0.1	-	-	0.3	-	1.0	0.9	-	-0.7	-	-1.3	-	-0.5	-	-
16	-3.4	-1.1	-0.9	-	0.6	-	-1.2	-	-	-	1.0	-	-	-	-0.1	-	-	-0.0	-	1.2	0.3	-	-0.9	-	-1.0	-	-0.9	-	-
17	-3.5	-1.2	-1.1	-	0.5	-	-1.3	-	-	-	0.9	-	-	-	-0.3	-	-	-0.1	-	-	0.1	-	-1.1	-	-1.2	-	-1.1	-	-
18	-2.7	-1.5	-1.1	-	0.4	-	-1.3	-	-	-	-	-	-	-	-0.6	-	-	-0.3	-	-	-1.7	-	-1.3	-	-2.5	-	-1.2	-	-
19	-2.8	-1.6	-1.2	-	-	-	-1.4	-	-	-	-	-	-	-	-0.7	-	-	-0.4	-	-	-1.9	-	-1.4	-	-2.5	-	-1.3	-	-
20	-4.4	-2.1	-1.1	-	-	-	-1.0	-	-	-	-	-	-	-	-0.3	-	-	-	-	-	-1.9	-	-1.4	-	-2.8	-	-0.9	-	-

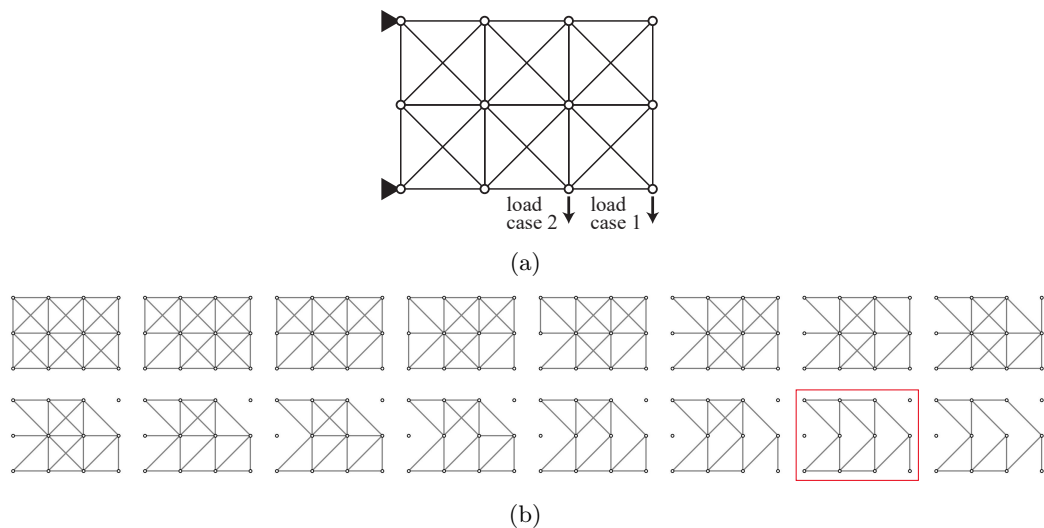


Figure 4.14: Boundary condition B1 of 3×2 -grid truss (reversed the order of load cases, $\gamma = 0.99$). (a) Initial GS. (b) Removal sequence of members.

Table 4.5: Action values $Q(s, i)$ ($i = 1, \dots, 29$) at each step of Fig. 4.14(b)

$t \backslash i$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	-0.4	4.0	4.7	5.0	5.1	5.1	0.5	4.1	5.1	5.0	4.7	4.3	4.0	5.0	4.6	4.7	5.0	4.0	5.3	4.8	5.0	4.9	5.2	5.1	4.1	5.0	4.9	5.1	5.2
1	-0.4	3.5	4.4	4.6	4.7	4.7	0.3	3.8	4.7	4.6	4.2	4.0	3.7	4.7	4.2	4.4	4.6	3.2	—	4.3	4.6	4.6	4.8	4.8	3.5	4.6	4.6	4.8	4.8
2	-0.6	3.1	4.0	4.1	4.0	4.1	-0.1	2.9	4.1	4.1	3.7	3.5	1.0	4.2	3.8	3.7	4.1	2.8	—	3.6	4.0	3.7	—	4.3	3.1	4.1	4.0	4.1	4.1
3	-0.5	3.0	3.8	3.9	3.8	3.9	0.0	2.4	3.9	4.0	3.5	3.3	0.8	4.0	3.5	3.5	3.9	2.3	—	3.5	3.9	3.5	—	—	2.3	3.8	3.8	3.9	3.9
4	-0.6	2.9	3.7	3.7	3.6	3.7	-0.2	2.2	3.8	—	3.3	3.2	0.6	3.8	3.3	3.4	3.7	2.3	—	3.3	3.7	3.4	—	—	2.1	3.6	3.6	3.8	3.8
5	-0.7	2.7	3.5	3.4	3.4	3.6	-0.4	2.0	3.6	—	3.2	3.0	0.5	—	3.2	3.2	3.6	2.2	—	3.1	3.5	3.2	—	—	1.8	3.4	3.4	3.6	3.6
6	-1.0	2.3	3.2	3.1	3.1	3.1	-0.6	1.7	3.4	—	2.9	2.8	0.0	—	2.8	2.8	3.3	1.9	—	2.7	3.2	2.7	—	—	1.5	3.0	3.1	—	2.3
7	-1.1	2.2	3.0	2.9	2.9	2.9	-0.8	1.4	—	—	2.7	2.6	-0.2	—	2.6	2.6	3.1	1.7	—	2.6	3.0	2.5	—	—	1.3	2.8	2.9	—	1.9
8	-1.2	2.0	2.9	2.8	2.8	2.7	-0.9	1.2	—	—	2.6	2.5	-0.7	—	2.5	2.5	—	1.6	—	2.4	2.9	2.3	—	—	1.2	2.6	2.8	—	1.5
9	-0.6	1.4	2.0	2.2	2.1	2.1	-1.7	0.7	—	—	1.4	0.7	-1.4	—	1.6	1.9	—	0.6	—	1.3	—	1.4	—	—	0.8	1.8	1.8	—	1.1
10	-0.7	1.2	1.7	—	1.9	1.9	-1.9	0.4	—	—	1.2	0.4	-1.6	—	1.3	1.7	—	0.2	—	1.0	—	1.2	—	—	0.5	1.5	1.5	—	0.9
11	-0.6	0.9	1.4	—	—	1.5	-2.3	0.1	—	—	0.9	0.1	-2.0	—	1.0	1.3	—	-0.3	—	0.7	—	0.9	—	—	0.3	1.3	0.8	—	0.3
12	-0.8	0.7	1.1	—	—	—	-2.5	-0.1	—	—	0.7	-0.3	-2.3	—	0.7	1.0	—	-0.5	—	0.5	—	0.7	—	—	0.1	1.1	0.4	—	-0.2
13	-0.9	0.4	—	—	—	—	-2.7	-0.3	—	—	0.5	-0.6	-2.6	—	0.6	0.9	—	-0.6	—	0.4	—	0.4	—	—	-0.1	0.9	0.2	—	-0.4
14	-1.5	0.1	—	—	—	—	-2.2	0.4	—	—	0.0	-0.8	-2.7	—	0.2	0.5	—	-0.4	—	-0.4	—	-0.1	—	—	-0.9	—	-0.6	—	-0.6
15	-1.5	-0.0	—	—	—	—	-2.7	0.2	—	—	0.1	-1.0	-2.8	—	0.0	—	—	-0.6	—	-0.1	—	0.0	—	—	-0.8	—	-0.9	—	-0.1

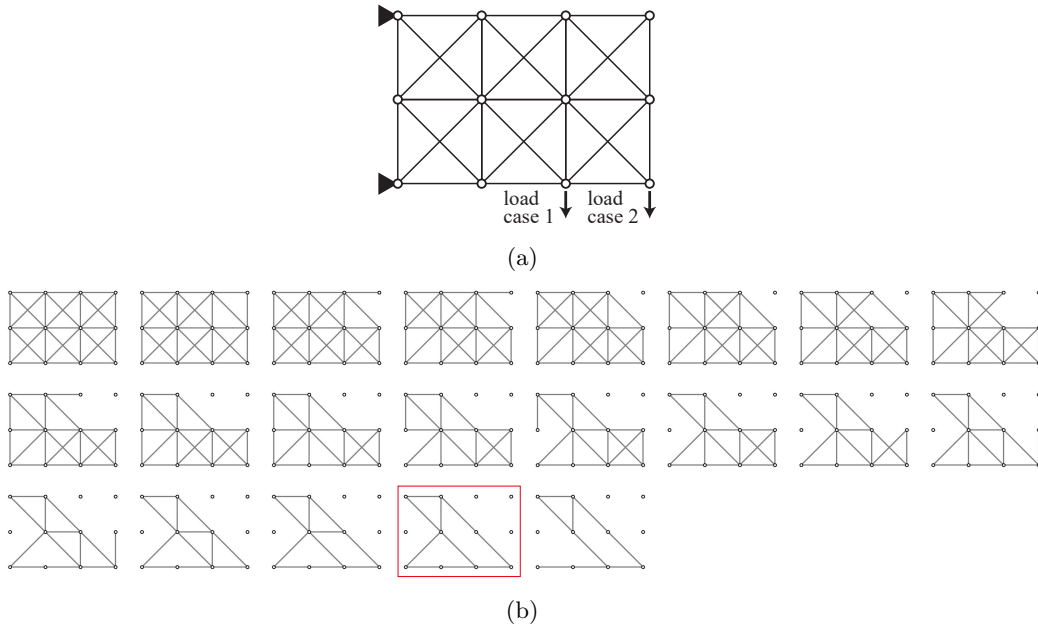


Figure 4.15: Boundary condition B1 of 3×2 -grid truss ($\gamma = 0.9$). (a) Initial GS. (b) Removal sequence of members.

Table 4.6: Action values $Q(s, i)$ ($i = 1, \dots, 29$) at each step of Fig. 4.15(b)

$t \backslash i$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	-0.6	1.2	1.9	1.9	1.9	2.1	-3.8	1.6	2.0	2.0	1.9	1.8	0.6	1.8	1.8	2.1	2.1	1.7	2.1	2.0	1.9	2.0	2.0	2.0	1.5	2.0	2.0	2.3	2.2
1	-0.5	0.8	1.8	1.8	1.8	1.9	-3.9	1.5	2.0	1.9	1.8	1.8	0.6	1.7	1.7	2.0	2.1	1.6	2.0	1.9	1.8	1.9	1.9	2.0	1.4	1.9	1.9	—	1.9
2	-0.6	0.7	1.8	1.8	1.7	1.9	-4.0	1.4	2.0	1.9	1.8	1.7	-0.2	1.7	1.6	2.0	—	1.6	2.0	1.9	1.8	1.9	1.8	1.9	1.4	1.9	1.8	—	1.8
3	-0.3	0.6	1.9	1.9	1.8	2.0	-3.7	1.5	2.0	1.9	1.8	1.8	-0.1	1.8	1.7	2.0	—	1.5	—	1.9	1.8	2.0	1.9	2.0	1.5	2.0	1.9	—	1.9
4	-0.3	0.5	1.8	1.8	1.8	2.0	-3.8	1.4	—	1.9	1.8	1.8	-0.3	1.8	1.7	2.0	—	1.5	—	1.9	1.8	1.9	1.8	2.0	1.4	1.9	1.9	—	1.9
5	-0.2	0.6	1.8	1.8	1.7	1.9	-3.5	1.4	—	1.9	1.7	1.7	-0.5	1.8	1.5	1.9	—	1.4	—	1.9	1.8	1.9	1.8	—	1.4	1.8	1.9	—	1.8
6	-0.2	0.3	1.6	1.6	1.6	1.7	-3.8	0.6	—	1.7	1.6	1.6	-1.0	1.6	1.4	—	—	1.3	—	1.7	1.6	1.7	1.6	—	1.2	1.7	1.7	—	1.8
7	-0.0	-0.5	0.8	1.2	1.2	1.2	-2.0	1.3	—	1.2	1.1	0.9	0.1	1.2	0.7	—	—	1.0	—	1.3	1.2	1.0	0.5	—	0.4	1.3	0.6	—	—
8	0.0	-0.6	0.6	1.1	1.2	1.1	-1.8	1.2	—	1.2	1.0	0.9	-0.0	1.1	0.7	—	—	0.9	—	1.2	1.1	0.9	0.4	—	0.4	—	0.5	—	—
9	-0.0	-0.7	0.5	1.1	1.1	1.1	-2.0	—	—	1.1	1.0	0.8	-0.1	1.1	0.7	—	—	0.8	—	1.1	1.1	0.8	0.3	—	0.3	—	0.2	—	—
10	0.0	-0.8	0.3	1.1	0.9	1.0	-1.9	—	—	1.1	1.0	0.8	-0.5	1.0	0.6	—	—	0.7	—	—	0.8	0.5	-0.7	—	0.4	—	-0.3	—	—
11	-0.1	-0.9	0.2	1.0	0.9	1.0	-1.9	—	—	1.0	0.8	-0.6	1.0	0.6	—	—	—	0.7	—	—	0.8	0.5	-0.8	—	0.4	—	-0.4	—	—
12	-0.2	-1.0	0.1	—	0.8	0.9	-1.9	—	—	0.8	0.7	-0.7	0.9	0.4	—	—	—	0.6	—	—	0.6	0.3	-1.2	—	0.1	—	-0.6	—	—
13	-0.1	-1.1	-0.1	—	0.7	0.7	-1.7	—	—	0.7	0.6	-0.8	—	0.4	—	—	—	0.5	—	—	0.5	0.2	-1.4	—	0.0	—	-0.6	—	—
14	-0.3	-0.8	-2.1	—	0.4	—	-1.6	—	—	0.5	0.4	0.5	—	0.1	—	—	—	0.3	—	—	0.3	0.5	-2.9	—	-0.1	—	-0.2	—	—
15	-0.4	-0.6	-2.1	—	0.3	—	-1.6	—	—	0.4	0.3	0.3	—	0.0	—	—	—	0.2	—	—	0.2	—	-2.2	—	-0.2	—	-0.3	—	—
16	-0.5	-0.7	-2.1	—	0.1	—	-1.6	—	—	0.1	0.2	—	-0.1	—	—	—	—	0.0	—	—	0.0	—	-1.9	—	-0.4	—	-0.4	—	—
17	-0.5	-0.7	-1.9	—	0.0	—	-1.6	—	—	0.1	—	—	-0.2	—	—	—	—	-0.0	—	—	-0.0	—	-1.4	—	-0.4	—	-0.4	—	—
18	-0.7	-0.8	-1.7	—	-0.3	—	-1.9	—	—	—	—	—	—	-0.5	—	—	—	-0.3	—	—	-0.9	—	-0.9	—	-0.9	—	-0.9	—	—
19	-0.7	-0.8	-1.5	—	—	—	-1.6	—	—	—	—	—	—	-0.6	—	—	—	-0.4	—	—	-0.9	—	-0.9	—	-0.9	—	-0.7	—	—
20	-0.9	-0.8	-1.4	—	—	—	-1.7	—	—	—	—	—	—	-0.5	—	—	—	—	—	—	-0.9	—	-1.0	—	-1.0	—	-0.7	—	—

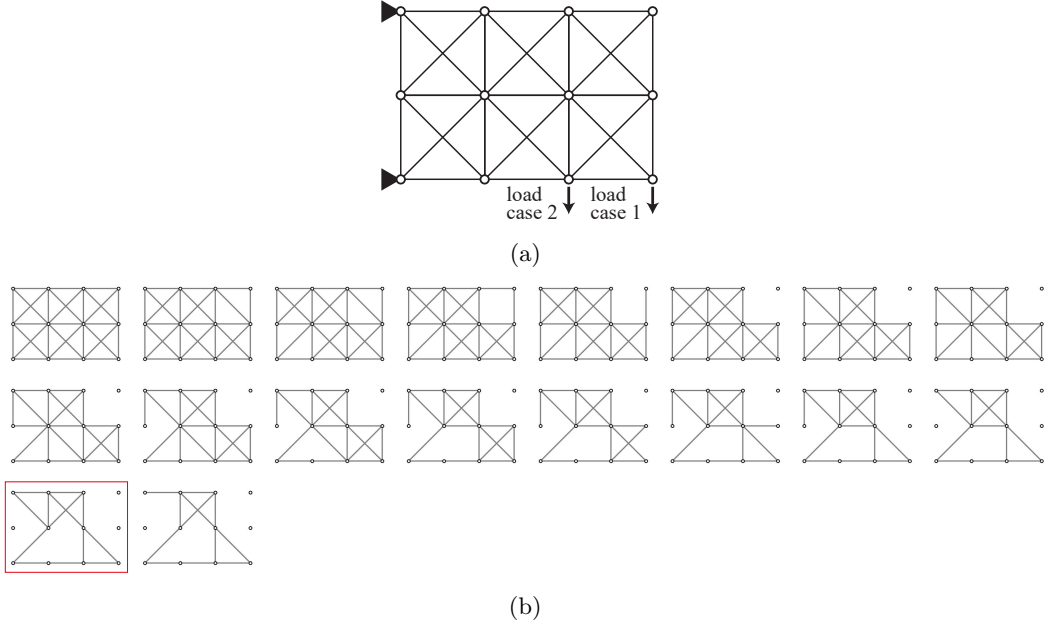
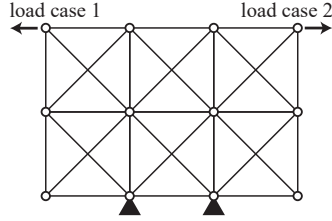


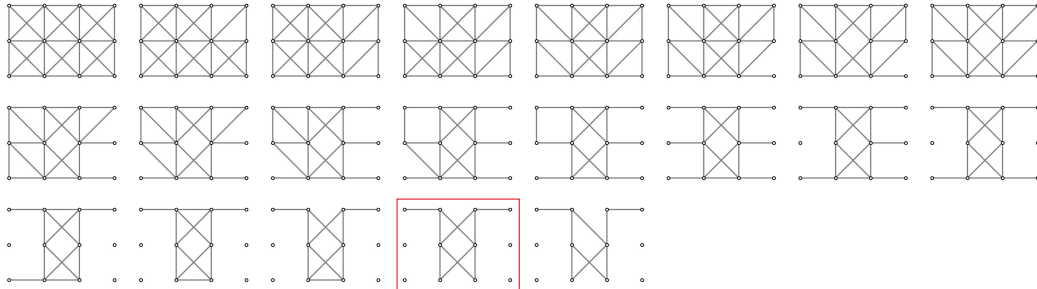
Figure 4.16: Boundary condition B1 of 3×2 -grid truss (reversed the order of load cases, $\gamma = 0.9$). (a) Initial GS. (b) Removal sequence of members.

Table 4.7: Action values $Q(s, i)$ ($i = 1, \dots, 29$) at each step of Fig. 4.16(b)

$t \backslash i$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	-0.9	1.8	1.9	2.1	2.0	2.2	-3.2	1.7	2.2	2.1	2.1	1.9	1.8	1.9	1.8	2.2	2.2	1.8	2.3	2.1	2.0	2.1	2.2	2.1	1.7	2.1	2.1	2.4	2.3
1	-0.9	1.8	1.9	2.1	2.0	2.1	-3.2	1.7	2.2	2.1	2.1	1.8	1.9	1.9	1.8	2.2	2.2	1.8	2.3	2.1	2.0	2.0	2.1	2.1	1.7	2.1	2.1	—	2.2
2	-0.5	1.7	2.0	2.1	2.1	2.2	-2.9	1.8	2.3	2.2	2.1	1.9	1.9	2.0	1.8	2.2	2.3	1.7	—	2.2	2.1	2.1	2.2	2.2	1.7	2.2	2.1	—	2.3
3	-0.5	1.0	1.4	1.7	1.6	1.6	-3.1	1.6	1.9	1.8	1.7	1.3	1.6	1.6	1.4	1.6	1.8	1.4	—	1.7	1.7	1.6	1.4	1.8	1.2	1.7	1.4	—	—
4	-0.6	1.0	1.4	1.7	1.5	1.6	-3.2	1.5	—	1.7	1.7	1.3	1.6	1.6	1.3	1.5	1.7	1.3	—	1.7	1.6	1.6	1.4	1.7	1.2	1.7	1.4	—	—
5	-0.7	0.8	1.3	1.6	1.4	1.5	-3.3	1.4	—	1.6	1.6	1.2	1.5	1.5	1.3	1.4	—	1.3	—	1.7	1.6	1.5	1.3	1.7	1.1	1.6	1.3	—	—
6	-0.6	0.7	1.2	1.4	1.3	1.4	-2.3	1.2	—	1.5	1.4	1.0	1.3	1.3	1.0	1.2	—	1.0	—	1.5	1.4	1.3	1.1	—	0.8	1.4	1.1	—	—
7	-0.5	-0.1	1.1	1.4	1.3	1.3	-2.5	1.2	—	1.4	1.4	1.0	1.3	1.3	1.0	1.2	—	1.0	—	1.3	1.3	1.0	—	0.8	1.3	1.1	—	—	—
8	-0.5	-0.2	1.1	1.3	1.2	1.3	-2.5	1.2	—	—	1.3	1.0	1.2	1.3	1.0	1.1	—	0.9	—	—	1.2	1.3	0.9	—	0.8	1.3	1.0	—	—
9	-0.4	-0.5	1.0	—	1.1	1.2	-2.7	1.0	—	—	1.2	0.9	1.1	1.2	0.9	1.0	—	0.9	—	—	1.2	1.2	0.8	—	0.6	1.2	0.9	—	—
10	-0.5	-1.1	0.9	—	1.0	1.1	-2.8	0.9	—	—	0.8	1.1	1.1	0.8	0.9	—	0.6	—	—	1.1	1.1	0.6	—	0.5	1.1	0.8	—	—	—
11	-0.9	-1.1	0.0	—	0.3	0.4	-2.0	-0.1	—	—	—	-0.1	0.4	0.3	0.0	0.1	—	-0.6	—	—	—	0.3	0.2	—	0.1	0.1	0.0	—	—
12	-0.9	-0.6	-1.8	—	0.1	0.3	-2.1	-0.2	—	—	—	-0.3	—	0.2	-0.1	-0.0	—	-0.8	—	—	—	0.3	-1.1	—	-0.1	-0.0	-0.1	—	—
13	-0.9	-0.7	-1.6	—	-0.0	0.1	-2.2	-0.3	—	—	—	-0.5	—	0.1	-0.2	-0.2	—	-0.9	—	—	—	—	-0.9	—	-0.2	-0.2	-0.2	—	—
14	-1.0	-0.7	-1.5	—	-0.1	—	-2.1	-0.4	—	—	—	-0.6	—	0.0	-0.3	-0.2	—	-1.0	—	—	—	—	-1.0	—	-0.3	-0.2	-0.3	—	—
15	-1.0	-0.8	-1.5	—	-0.2	—	-1.9	-0.4	—	—	—	-0.6	—	—	-0.4	-0.3	—	-1.1	—	—	—	—	-1.1	—	-0.4	-0.3	-0.3	—	—
16	-1.2	-1.1	-1.3	—	—	—	-1.5	-1.3	—	—	—	-1.0	—	—	-1.3	-1.3	—	-1.6	—	—	—	—	-1.4	—	-0.9	-1.2	-1.2	—	—
17	-1.6	-0.8	-1.4	—	—	—	-1.3	-0.9	—	—	—	-0.8	—	—	-0.7	-1.0	—	-1.1	—	—	—	—	-1.3	—	—	-1.0	-0.6	—	—



(a)

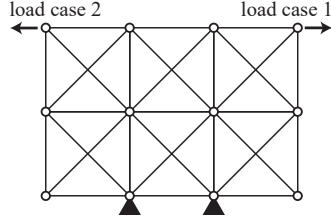


(b)

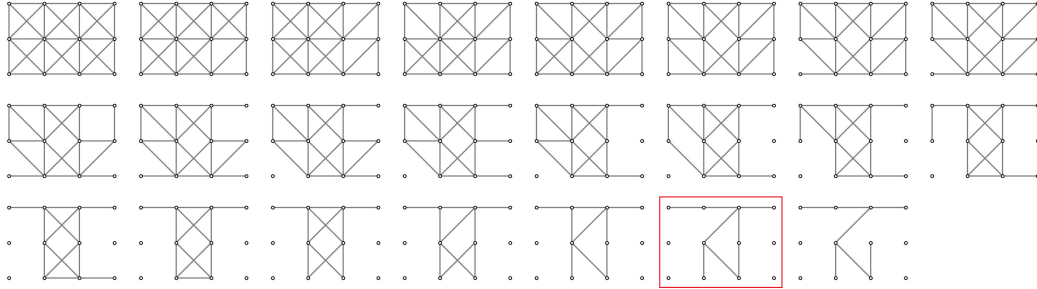
Figure 4.17: Boundary condition B2 of 3×2 -grid truss ($\gamma = 0.99$). (a) Initial GS. (b) Removal sequence of members.

Table 4.8: Action values $Q(s, i)$ ($i = 1, \dots, 29$) at each step of Fig. 4.17(b)

$t \backslash i$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	2.6	2.4	2.6	2.8	3.0	2.7	0.2	2.7	1.3	2.8	0.7	-0.2	2.9	2.7	2.5	2.6	2.8	2.9	2.5	1.0	0.6	2.5	3.2	3.0	3.0	2.5	2.8	2.7	3.2
1	2.5	2.3	2.5	2.6	2.9	2.5	0.1	2.5	1.2	2.6	0.6	-0.3	2.7	2.6	2.4	2.4	2.6	2.8	2.4	0.9	0.6	2.3	3.1	2.9	2.9	2.4	2.7	2.5	—
2	2.4	2.2	2.4	2.5	2.7	2.4	-0.0	2.4	1.0	2.6	0.5	-0.7	2.6	2.5	2.3	2.2	2.5	2.7	2.3	0.8	0.5	2.3	—	2.8	2.8	2.3	2.5	2.4	—
3	2.2	2.0	2.2	2.4	2.5	2.2	0.1	2.1	1.0	2.3	0.4	-0.8	2.5	2.3	2.1	2.1	2.3	2.5	2.1	0.7	0.6	2.1	—	—	2.5	2.2	2.3	2.2	—
4	2.0	1.8	2.0	2.1	2.3	2.1	-0.0	1.9	0.9	2.2	0.3	-0.9	2.3	2.1	1.8	1.9	2.1	—	1.9	0.5	0.3	1.9	—	—	2.2	2.0	2.1	2.1	—
5	1.9	1.8	2.0	2.1	2.2	2.0	-0.1	1.9	0.8	2.1	0.2	-1.0	—	2.0	1.8	1.9	2.1	—	1.9	0.4	0.2	1.8	—	—	2.2	2.0	2.1	2.0	—
6	1.6	1.4	1.7	1.7	—	1.6	-0.3	1.6	0.6	1.8	0.0	-1.3	—	1.8	1.4	1.6	1.8	—	1.6	0.2	-0.1	1.6	—	—	1.8	1.6	1.7	1.7	—
7	1.3	1.2	1.4	1.4	—	1.4	0.1	1.0	-0.6	1.5	-0.4	-2.6	—	1.4	1.0	1.1	—	—	1.4	0.4	-0.2	1.7	—	—	1.5	1.3	0.7	1.5	—
8	1.2	1.0	1.3	1.3	—	1.3	0.1	0.9	-0.6	1.4	-0.5	-2.7	—	1.4	0.9	1.0	—	—	1.3	0.3	-0.4	—	—	—	1.4	1.2	0.6	1.4	—
9	1.1	0.9	1.2	1.2	—	1.2	0.0	0.8	-0.6	—	-0.5	-2.7	—	1.3	0.8	0.9	—	—	1.1	0.3	-0.5	—	—	—	1.3	1.1	0.5	1.3	—
10	1.0	0.8	1.1	1.1	—	1.0	-0.1	0.7	-0.7	—	-0.6	-2.7	—	1.2	0.7	0.8	—	—	1.0	0.1	-0.6	—	—	—	1.2	1.0	0.4	—	—
11	0.8	0.5	0.8	1.0	—	0.8	-1.9	0.6	-0.5	—	-1.1	-3.6	—	1.1	0.5	0.3	—	—	1.1	-0.1	-0.1	—	—	—	—	0.4	0.1	—	—
12	0.6	0.4	0.7	0.9	—	0.7	-1.9	0.5	-0.6	—	-1.1	-3.7	—	1.0	0.4	0.2	—	—	—	-0.2	-0.2	—	—	—	—	0.3	0.0	—	—
13	0.5	0.3	0.7	0.7	—	0.7	-1.9	0.3	-0.6	—	-1.2	-3.7	—	—	0.3	0.2	—	—	—	-0.3	-0.3	—	—	—	—	0.2	-0.1	—	—
14	0.4	0.1	0.5	—	—	0.6	-2.0	0.2	-0.7	—	-1.2	-3.9	—	—	0.2	0.1	—	—	—	-0.4	-0.4	—	—	—	—	0.1	-0.2	—	—
15	0.2	-0.0	0.4	—	—	—	-2.1	0.0	-0.8	—	-1.3	-3.9	—	—	0.1	-0.1	—	—	—	-0.5	-0.5	—	—	—	—	0.0	-0.3	—	—
16	0.2	-0.1	—	—	—	—	-2.2	-0.0	-0.9	—	-1.3	-3.9	—	—	-0.0	-0.1	—	—	—	-0.6	-0.6	—	—	—	—	-0.1	-0.3	—	—
17	—	-0.1	—	—	—	—	-2.2	-0.1	-0.9	—	-1.3	-4.0	—	—	-0.1	-0.2	—	—	—	-0.6	-0.7	—	—	—	—	-0.1	-0.4	—	—
18	—	-0.4	—	—	—	—	-1.6	—	-1.1	—	-1.2	-2.9	—	—	-0.9	-0.9	—	—	—	-1.4	-1.6	—	—	—	—	-0.8	-1.4	—	—
19	—	—	—	—	—	—	-1.6	—	-1.1	—	-1.2	-2.8	—	—	-0.9	-1.0	—	—	—	-1.4	-1.6	—	—	—	—	-0.9	-1.4	—	—
20	—	—	—	—	—	—	-2.5	—	-1.1	—	-0.9	-5.3	—	—	-0.9	-0.1	—	—	—	-1.4	-0.3	—	—	—	—	—	-1.7	—	—



(a)

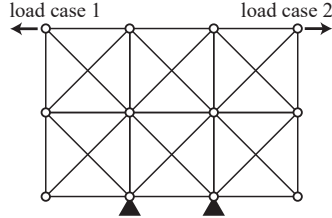


(b)

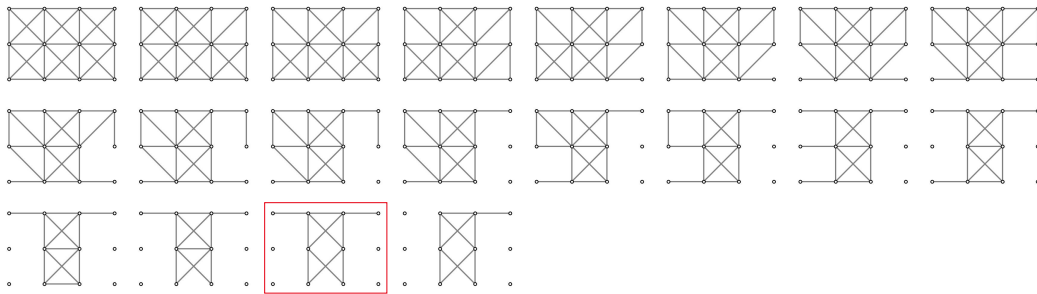
Figure 4.18: Boundary condition B2 of 3×2 -grid truss (reversed the order of load cases, $\gamma = 0.99$). (a) Initial GS. (b) Removal sequence of members.

Table 4.9: Action values $Q(s, i)$ ($i = 1, \dots, 29$) at each step of Fig. 4.18(b)

$t \backslash i$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	2.7	2.6	2.8	2.7	3.1	2.8	0.8	2.7	0.4	2.9	-0.1	0.7	2.9	2.6	2.6	2.6	2.8	3.0	2.6	0.5	1.2	2.5	3.3	3.1	2.9	2.6	2.9	3.0	3.2
1	2.5	2.4	2.6	2.6	2.9	2.6	0.6	2.5	0.2	2.7	-0.2	0.5	2.8	2.5	2.5	2.4	2.6	2.8	2.5	0.2	1.1	2.3	—	3.0	2.7	2.4	2.7	2.8	3.1
2	2.4	2.3	2.4	2.4	2.7	2.6	0.5	2.2	0.2	2.6	-0.4	0.4	2.6	2.3	2.3	2.2	2.4	2.7	2.3	0.2	0.9	2.2	—	2.8	2.6	2.3	2.6	2.6	—
3	2.2	2.1	2.2	2.2	2.5	2.3	0.4	2.0	0.1	2.3	-0.6	0.3	2.4	2.0	2.0	2.0	2.2	2.5	2.0	0.2	0.8	2.0	—	—	2.3	2.0	2.3	2.4	—
4	1.8	1.7	1.8	1.8	—	1.8	0.1	1.5	-0.3	2.0	-1.0	-0.1	1.9	1.7	1.5	1.4	1.8	2.1	1.7	-0.2	0.4	1.6	—	—	1.8	1.5	1.8	1.9	—
5	1.6	1.5	1.6	1.6	—	1.6	-0.1	1.2	-0.4	1.8	-1.3	-0.2	1.8	1.5	1.2	1.2	1.6	—	1.5	-0.5	0.1	1.4	—	—	1.5	1.3	1.6	1.7	—
6	1.6	1.4	1.6	1.5	—	1.5	-0.2	1.2	-0.5	—	-1.3	-0.3	1.7	1.5	1.2	1.2	1.5	—	1.4	-0.5	0.1	1.4	—	—	1.4	1.2	1.6	1.6	—
7	1.5	1.3	1.5	1.4	—	1.4	-0.2	1.1	-0.5	—	-1.4	-0.3	—	1.4	1.1	1.1	1.5	—	1.4	-0.6	0.0	1.3	—	—	1.4	1.2	1.5	1.5	—
8	1.2	1.0	1.2	1.1	—	1.2	-0.1	0.4	-2.1	—	-2.3	-0.7	—	1.1	0.5	0.8	1.3	—	1.0	0.3	-0.1	1.2	—	—	1.0	0.8	0.8	—	—
9	1.2	0.9	1.1	1.0	—	1.1	-0.1	0.3	-2.0	—	-2.4	-0.8	—	1.0	0.4	0.7	—	—	0.9	0.2	-0.2	1.1	—	—	0.9	0.7	0.8	—	—
10	—	0.8	1.0	0.9	—	1.0	-0.2	0.2	-2.0	—	-2.4	-0.9	—	0.9	0.4	0.7	—	—	0.7	0.1	-0.3	1.0	—	—	0.8	0.6	0.7	—	—
11	—	0.6	0.9	0.9	—	0.9	-0.3	0.1	-2.1	—	-2.4	-0.9	—	0.8	0.3	0.6	—	—	0.6	0.0	-0.4	—	—	—	0.8	0.6	0.6	—	—
12	—	0.5	0.7	0.8	—	—	-0.4	-0.0	-2.2	—	-2.6	-1.0	—	0.7	0.2	0.5	—	—	0.5	-0.1	-0.5	—	—	—	0.7	0.4	0.5	—	—
13	—	0.2	0.4	—	—	—	-1.1	0.0	-2.0	—	-3.8	-1.3	—	0.7	0.0	0.2	—	—	0.8	-0.3	-0.2	—	—	—	0.8	-0.3	0.3	—	—
14	—	0.1	0.4	—	—	—	-1.2	-0.1	-2.0	—	-3.9	-1.3	—	0.6	-0.0	0.1	—	—	—	-0.4	-0.2	—	—	—	0.7	-0.3	0.2	—	—
15	—	0.0	0.3	—	—	—	-1.3	-0.2	-2.1	—	-3.9	-1.4	—	0.5	-0.1	-0.0	—	—	—	-0.5	-0.3	—	—	—	—	-0.4	0.1	—	—
16	—	-0.0	0.2	—	—	—	-1.3	-0.3	-2.2	—	-3.9	-1.5	—	—	-0.2	-0.1	—	—	—	-0.6	-0.4	—	—	—	—	-0.4	0.0	—	—
17	—	-0.1	—	—	—	—	-1.4	-0.4	-2.2	—	-3.9	-1.4	—	—	-0.2	-0.1	—	—	—	-0.7	-0.4	—	—	—	—	-0.4	-0.1	—	—
18	—	—	—	—	—	—	-1.4	-0.4	-2.3	—	-3.8	-1.4	—	—	-0.3	-0.2	—	—	—	-0.7	-0.4	—	—	—	—	-0.5	-0.1	—	—
19	—	—	—	—	—	—	-1.5	-1.1	-1.8	—	-5.2	-1.0	—	—	-0.1	-1.0	—	—	—	-0.1	-1.4	—	—	—	—	-1.8	—	—	—
20	—	—	—	—	—	—	-1.5	-1.2	-1.8	—	-5.2	-1.1	—	—	-0.2	-1.0	—	—	—	—	-1.4	—	—	—	—	-1.9	—	—	—
21	—	—	—	—	—	—	-1.6	-1.3	-1.9	—	-5.2	-1.1	—	—	—	-1.1	—	—	—	—	—	—	—	—	—	-1.9	—	—	—
22	—	—	—	—	—	—	-1.8	-1.2	-3.3	—	-5.7	-0.9	—	—	—	—	—	—	—	—	-1.8	—	—	—	—	-1.1	—	—	—



(a)



(b)

Figure 4.19: Boundary condition B2 of 3×2 -grid truss ($\gamma = 0.9$). (a) Initial GS. (b) Removal sequence of members.

Table 4.10: Action values $Q(s, i)$ ($i = 1, \dots, 29$) at each step of Fig. 4.19(b)

$t \backslash i$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	2.5	2.3	2.7	2.5	2.5	2.5	0.5	2.5	0.3	2.8	0.3	-0.5	2.9	2.7	2.6	2.6	2.3	2.8	2.6	-0.3	1.3	2.7	3.0	3.0	2.6	2.7	2.7	2.8	3.2
1	2.5	2.4	2.7	2.5	2.5	2.7	0.6	2.5	0.9	2.9	0.5	-0.4	2.9	2.7	2.6	2.6	2.1	2.9	2.6	0.0	1.4	2.8	3.0	3.1	2.6	2.7	2.7	2.7	—
2	2.5	2.4	2.7	2.6	2.5	2.6	1.0	2.5	1.0	2.8	0.4	-0.4	2.9	2.7	2.5	2.6	1.9	2.8	2.6	-0.0	1.5	2.8	3.0	—	2.6	2.6	2.7	2.7	—
3	2.4	2.2	2.6	2.4	2.4	2.5	0.8	2.4	0.7	2.7	0.2	-0.5	2.8	2.5	2.4	2.4	1.6	2.7	2.5	-0.5	1.3	2.6	—	—	2.4	2.5	2.6	2.5	—
4	2.3	2.2	2.5	2.4	2.3	2.5	0.7	2.3	0.7	2.6	0.2	-0.6	—	2.5	2.3	2.4	1.4	2.7	2.4	-0.6	1.2	2.6	—	—	2.4	2.4	2.5	2.5	—
5	2.2	2.0	2.4	2.2	2.1	2.3	0.6	2.1	0.6	2.4	0.1	-0.5	—	2.3	2.1	2.2	1.2	—	2.3	-0.9	1.2	2.4	—	—	2.2	2.2	2.3	2.3	—
6	2.1	1.9	2.3	2.2	2.1	2.2	0.5	2.0	0.5	—	0.0	-0.5	—	2.2	2.0	2.1	1.2	—	2.2	-1.0	1.1	2.3	—	—	2.1	2.1	2.3	2.2	—
7	1.7	1.6	1.9	1.7	1.7	2.0	0.4	0.8	-0.4	—	0.2	0.2	—	1.7	1.5	1.7	1.9	—	1.8	0.9	1.1	—	—	—	1.7	1.5	1.8	2.0	—
8	1.5	1.4	1.7	1.5	1.5	—	0.3	0.6	-0.5	—	0.1	0.1	—	1.5	1.3	1.5	1.7	—	1.6	0.5	0.9	—	—	—	1.6	1.3	1.6	1.8	—
9	1.3	1.1	1.4	1.3	1.2	—	0.3	0.4	-0.4	—	0.0	0.2	—	1.2	1.0	1.2	1.4	—	1.3	0.3	0.7	—	—	—	1.3	1.0	1.3	—	—
10	1.2	1.1	—	1.2	1.1	—	0.3	0.4	-0.3	—	0.1	0.3	—	1.2	0.9	1.2	1.3	—	1.2	0.3	0.6	—	—	—	1.2	0.9	1.2	—	—
11	1.1	1.0	—	1.1	1.0	—	0.3	0.3	0.2	—	0.0	0.3	—	1.1	0.8	1.0	—	—	1.1	0.2	0.5	—	—	—	1.1	0.8	1.1	—	—
12	0.4	0.3	—	0.4	0.2	—	-0.0	0.1	-0.1	—	-0.5	-0.4	—	0.5	-0.0	0.2	—	—	—	-0.1	0.2	—	—	—	0.5	-0.0	0.1	—	—
13	0.2	0.1	—	0.3	0.0	—	-0.1	-0.1	-0.2	—	-0.6	-0.5	—	0.3	-0.2	0.0	—	—	—	-0.3	-0.0	—	—	—	—	-0.3	-0.1	—	—
14	0.2	0.1	—	0.2	-0.0	—	-0.2	-0.1	-0.3	—	-0.6	-0.6	—	—	-0.3	-0.0	—	—	—	-0.4	-0.1	—	—	—	—	-0.3	-0.2	—	—
15	0.1	0.0	—	—	-0.0	—	-0.1	-0.2	-0.3	—	-0.6	-0.6	—	—	-0.3	-0.1	—	—	—	-0.4	-0.1	—	—	—	—	-0.3	-0.2	—	—
16	—	-0.1	—	—	-0.1	—	-0.2	-0.3	-0.4	—	-0.7	-0.7	—	—	-0.4	-0.2	—	—	—	-0.5	-0.2	—	—	—	—	-0.5	-0.4	—	—
17	—	—	—	—	-0.2	—	-0.3	-0.4	-0.4	—	-0.9	-0.8	—	—	-0.5	-0.3	—	—	—	-0.7	-0.3	—	—	—	—	-0.6	-0.5	—	—
18	—	—	—	—	—	—	-0.2	-0.4	-0.4	—	-0.8	-0.8	—	—	-0.6	-0.3	—	—	—	-0.7	-0.3	—	—	—	—	-0.6	-0.4	—	—
19	—	—	—	—	—	—	—	-0.3	-0.3	—	-0.7	-0.7	—	—	-0.6	-0.2	—	—	—	-0.6	-0.4	—	—	—	—	-0.5	-0.3	—	—

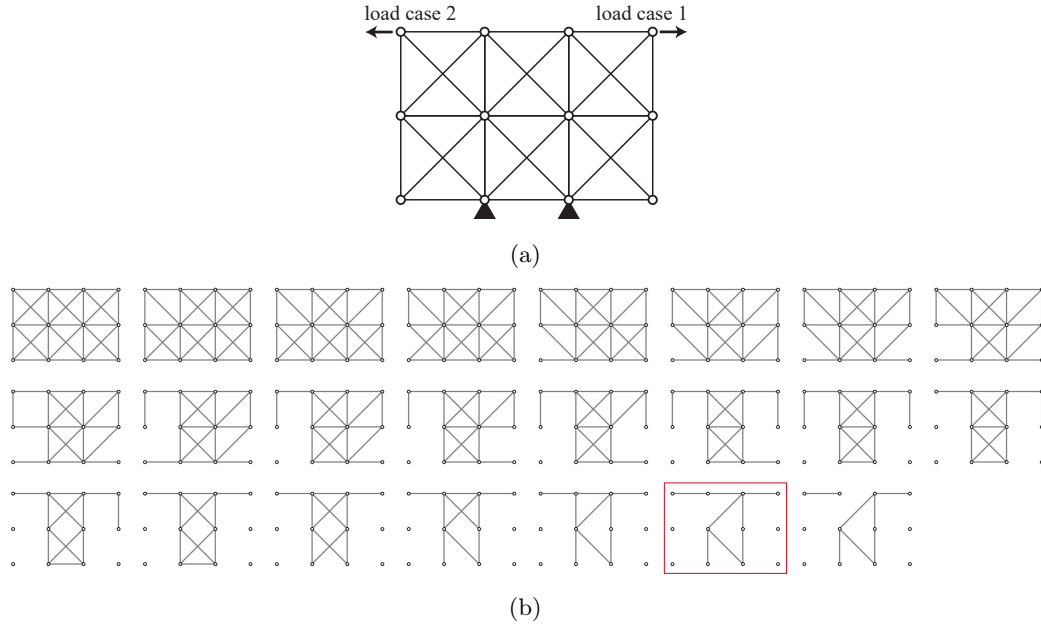


Figure 4.20: Boundary condition B2 of 3×2 -grid truss (reversed the order of load cases, $\gamma = 0.9$). (a) Initial GS. (b) Removal sequence of members.

Table 4.11: Action values $Q(s, i)$ ($i = 1, \dots, 29$) at each step of Fig. 4.20(b)

$t \backslash i$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	
0	2.7	2.3	2.5	2.6	2.5	2.5	-0.8	2.3	-0.7	3.0	0.3	0.4	2.9	2.2	2.6	2.7	2.4	3.0	2.8	1.1	0.3	2.7	2.9	3.2	2.8	2.6	2.7	2.5	3.2	
1	2.7	2.3	2.5	2.6	2.5	2.5	-0.8	2.3	-0.6	3.0	0.2	0.5	2.9	1.5	2.6	2.7	2.4	3.0	2.7	1.1	0.5	2.7	2.9	—	2.7	2.6	2.7	2.5	3.2	
2	2.7	2.3	2.5	2.6	2.5	2.5	-0.9	2.3	-0.5	2.9	0.2	0.6	2.9	1.4	2.5	2.6	1.8	2.9	2.7	1.4	0.6	2.6	2.9	—	2.7	2.6	2.6	2.4	—	
3	2.6	2.2	2.4	2.5	2.4	2.5	-1.1	2.2	-0.6	—	0.1	0.6	2.8	1.4	2.4	2.5	1.8	2.9	2.6	1.2	0.4	2.6	2.8	—	2.6	2.5	2.5	2.4	—	
4	2.5	2.1	2.3	2.4	2.3	2.3	-1.3	1.9	-0.7	—	-0.2	0.4	2.7	1.1	2.3	2.4	1.6	—	2.5	1.0	-0.1	2.4	2.7	—	2.4	2.4	2.4	2.2	—	
5	2.3	1.9	2.1	2.2	2.1	2.2	-1.4	1.6	-1.0	—	-0.4	0.0	2.5	1.0	2.1	2.2	0.8	—	2.3	0.8	-0.4	2.3	—	—	2.2	2.2	2.2	2.1	—	
6	2.3	1.9	2.1	2.1	2.1	2.1	-1.5	1.5	-1.0	—	-0.4	-0.0	—	0.9	2.1	2.1	0.3	—	2.3	0.7	-0.4	2.2	—	—	2.2	2.1	2.2	2.0	—	
7	1.7	1.5	1.5	1.8	1.5	1.5	-1.7	0.4	-0.6	—	0.1	-0.0	—	1.6	1.6	1.4	-0.7	—	—	1.0	0.9	1.6	—	—	1.8	1.6	1.4	1.4	—	
8	1.5	1.2	1.3	1.5	1.3	1.2	-1.8	0.3	-0.8	—	0.2	-0.3	—	1.3	1.3	1.1	-1.0	—	—	0.7	0.6	1.4	—	—	1.3	1.1	1.1	—	—	
9	1.3	1.0	1.1	—	1.0	1.1	-1.8	0.1	-0.7	—	0.1	-0.4	—	1.1	1.1	0.9	-1.1	—	—	0.5	0.4	1.2	—	—	—	1.1	0.9	0.9	—	—
10	—	0.9	1.0	—	0.9	1.0	-1.7	0.1	-0.7	—	0.2	-0.4	—	1.0	1.0	0.8	-1.2	—	—	0.5	0.2	1.1	—	—	—	1.0	0.8	0.9	—	—
11	—	-0.0	0.1	—	-0.0	0.2	-1.5	-0.2	-0.8	—	-0.7	-1.1	—	0.0	-0.0	-0.2	0.0	—	—	-0.0	-0.5	—	—	—	—	-0.1	-0.3	0.1	—	—
12	—	-0.2	-0.1	—	-0.1	—	-1.5	-0.3	-0.8	—	-0.8	-0.9	—	-0.1	-0.2	-0.3	-0.2	—	—	-0.2	-0.6	—	—	—	—	-0.3	-0.4	-0.0	—	—
13	—	-0.3	-0.2	—	-0.3	—	-1.4	-0.4	-0.8	—	-0.9	-1.0	—	-0.2	-0.3	-0.5	-0.3	—	—	-0.3	-0.7	—	—	—	—	-0.4	-0.7	—	—	—
14	—	-0.4	—	—	-0.4	—	-1.4	-0.5	-0.9	—	-1.0	-1.1	—	-0.3	-0.4	-0.6	-0.4	—	—	-0.4	-0.9	—	—	—	—	-0.5	-0.8	—	—	—
15	—	-0.5	—	—	-0.5	—	-1.3	-0.6	-0.9	—	-1.1	-1.1	—	—	-0.5	-0.6	-0.5	—	—	-0.5	-1.0	—	—	—	—	-0.6	-0.9	—	—	—
16	—	-0.3	—	—	—	—	-1.1	-0.5	-0.7	—	-0.9	-0.9	—	—	-0.4	-0.5	-0.3	—	—	-0.4	-0.9	—	—	—	—	-0.5	-0.7	—	—	—
17	—	-0.4	—	—	—	—	-1.1	-0.5	-0.7	—	-1.0	-1.0	—	—	-0.4	-0.6	—	—	—	-0.4	-0.9	—	—	—	—	-0.5	-0.7	—	—	—
18	—	—	—	—	—	—	-1.2	-0.6	-0.7	—	-1.1	-1.2	—	—	-0.5	-0.7	—	—	—	-0.5	-1.0	—	—	—	—	-0.6	-0.9	—	—	—
19	—	—	—	—	—	—	-1.1	-1.0	-1.4	—	-1.7	-1.5	—	—	-0.7	-1.3	—	—	—	-1.5	—	—	—	—	—	-1.5	-0.6	—	—	—
20	—	—	—	—	—	—	-1.0	-1.0	-1.5	—	-1.6	-1.4	—	—	-0.7	-1.2	—	—	—	-1.4	—	—	—	—	—	-1.5	—	—	—	—
21	—	—	—	—	—	—	-1.0	-0.9	-1.4	—	-1.6	-1.3	—	—	-1.1	—	—	—	—	-1.3	—	—	—	—	—	-1.4	—	—	—	—
22	—	—	—	—	—	—	-0.7	—	-0.9	—	-1.3	-1.1	—	—	-0.8	—	—	—	—	-1.0	—	—	—	—	—	-1.0	—	—	—	—

4.4.3 Investigation of generalization performance 2: 6×6-grid truss

To verify the generalization performance of the trained agents to various trusses, it is necessary to demonstrate their performance through not only smaller trusses but also larger trusses. Next, the agents trained with $\gamma = 0.99$ and $\gamma = 0.9$ are applied without re-training. to a larger-scale truss, as shown in Fig. 4.21. The initial GS consists of 6×6 grids and the number of members is more than double the 4×4-grid truss. The truss is optimized for two loading conditions.

In the loading condition L1, the left two corners 1 and 7 are pin-supported, and rightward and downward unit loads are separately applied at the bottom-right corner 43, as shown in Fig. 4.22(a). Topology at each step of the removal sequence is illustrated in Figs. 4.22 - 4.25. Although the agent is applied to a larger-scale truss, a sparse optimal solution is successfully obtained.

In the loading condition L2, the left two corners are again pin-supported, and the outward unit load is separately applied at nodes 4 and 46, as shown in Fig. 4.26(a). One of the loads applied at node 4 is an irregular case where pin-supports and the loaded node aligns on the same straight line. Even in this irregular case, the agent successfully obtained the sparse optimal solution, as shown in Figs. 4.26 - 4.28. During their removal process, there are few isolated members apart from load-bearing ones and existing members efficiently transmit forces to the supports. On the other hand, the topology one step before the terminal state in Fig. 4.29 obviously contains several unnecessary members, and there is still room for reducing the total volume of the members. In order to avoid such a result, it is necessary to improve the reward function Eq. (4.4) so as not to leave unnecessary members as much as possible.

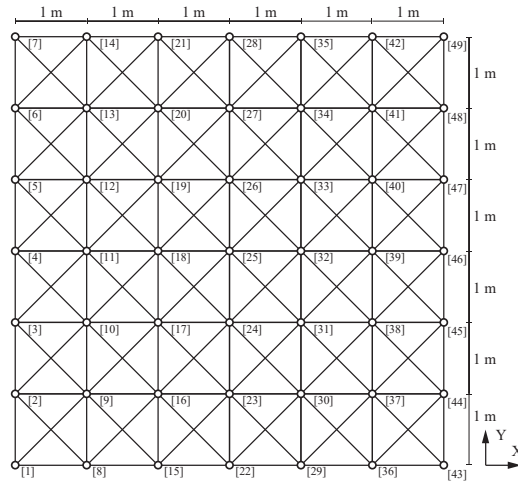
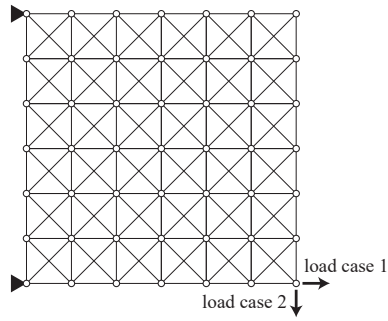
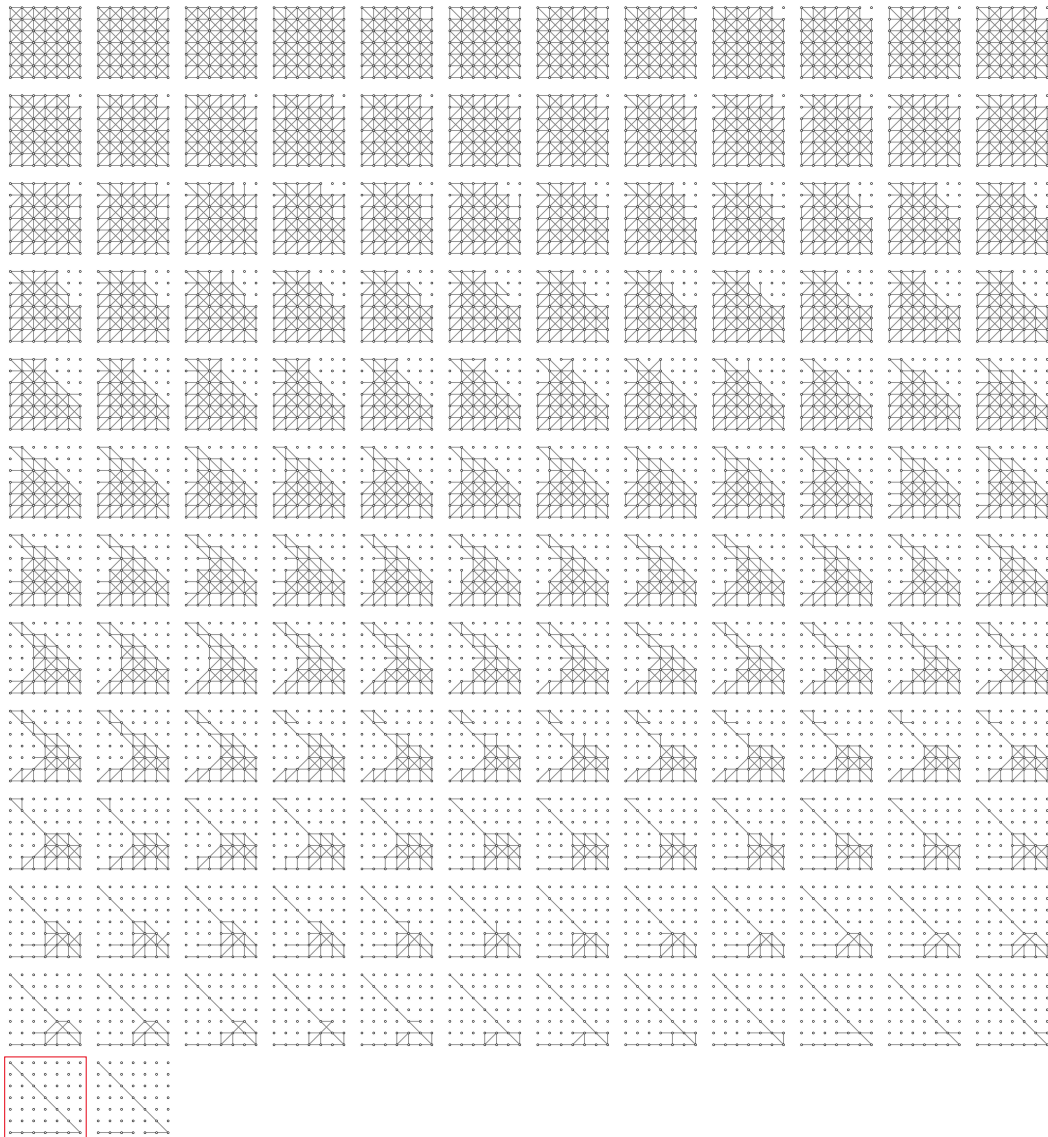


Figure 4.21: 6×6-grid truss ($V_s = 0.1858 \text{ [m}^3\text{]}$)

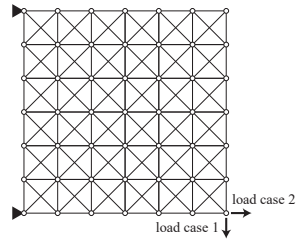


(a)

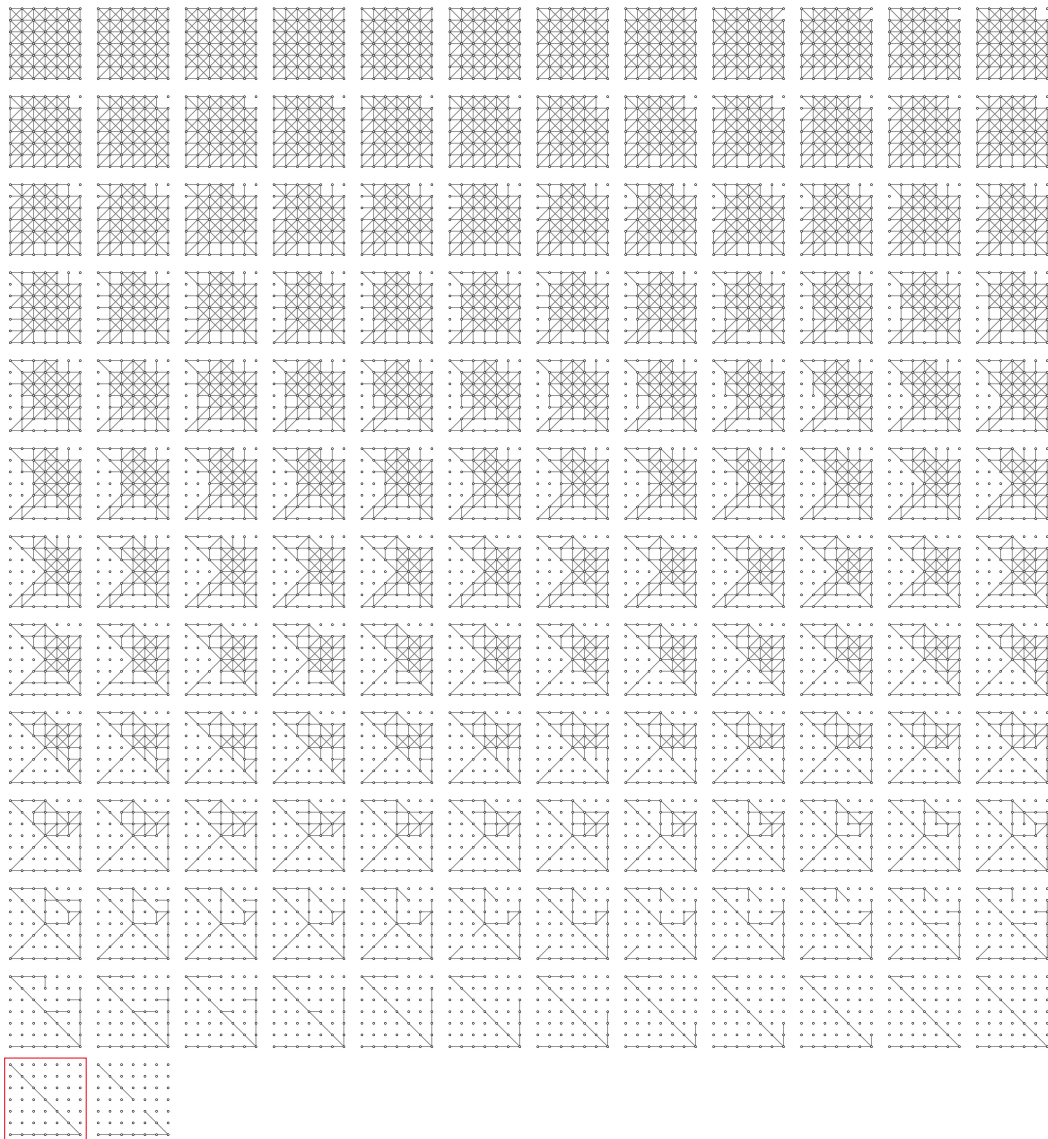


(b)

Figure 4.22: Loading condition L1 of 6 × 6-grid truss ($\gamma = 0.99$). (a) Initial GS.
 (b) Removal sequence of members.

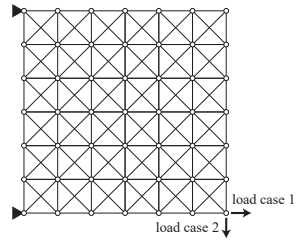


(a)

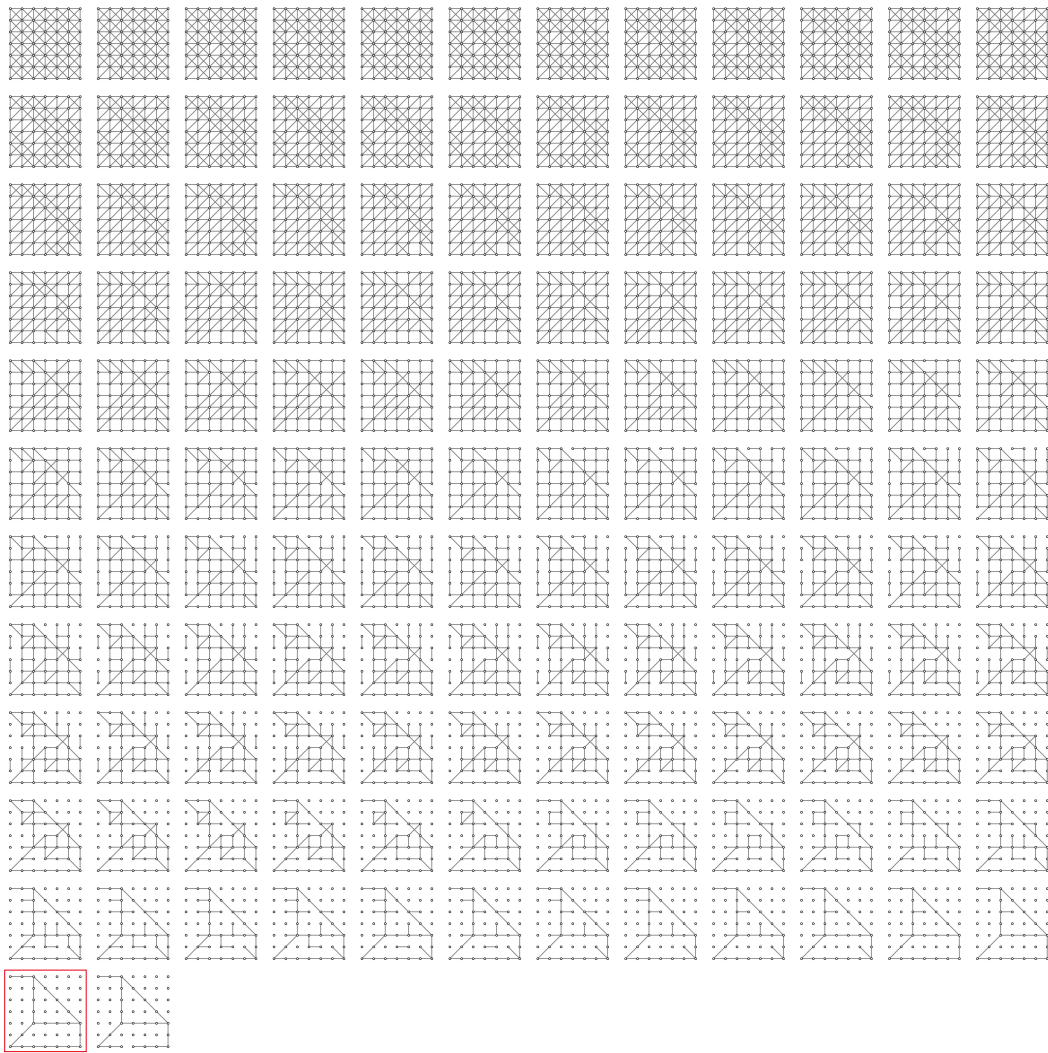


(b)

Figure 4.23: Loading condition L1 of 6×6 -grid truss (reversed the order of load cases, $\gamma = 0.99$). (a) Initial GS. (b) Removal sequence of members.

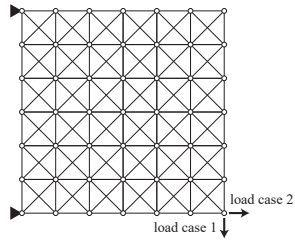


(a)

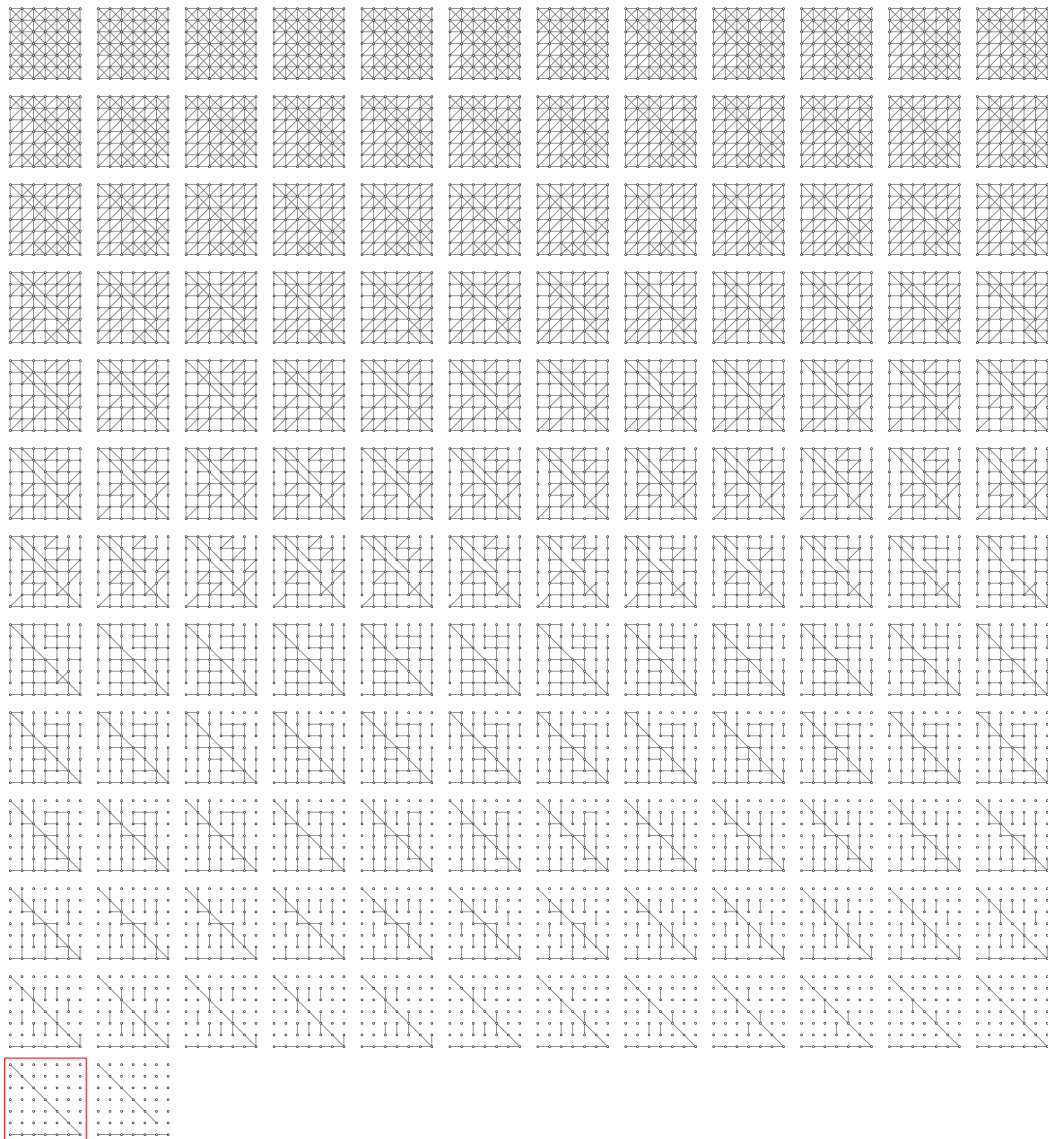


(b)

Figure 4.24: Loading condition L1 of 6×6 -grid truss ($\gamma = 0.9$). (a) Initial GS.
 (b) Removal sequence of members.

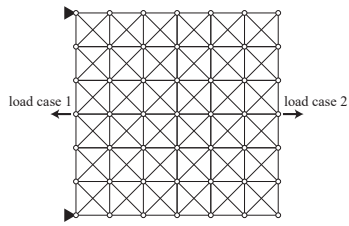


(a)

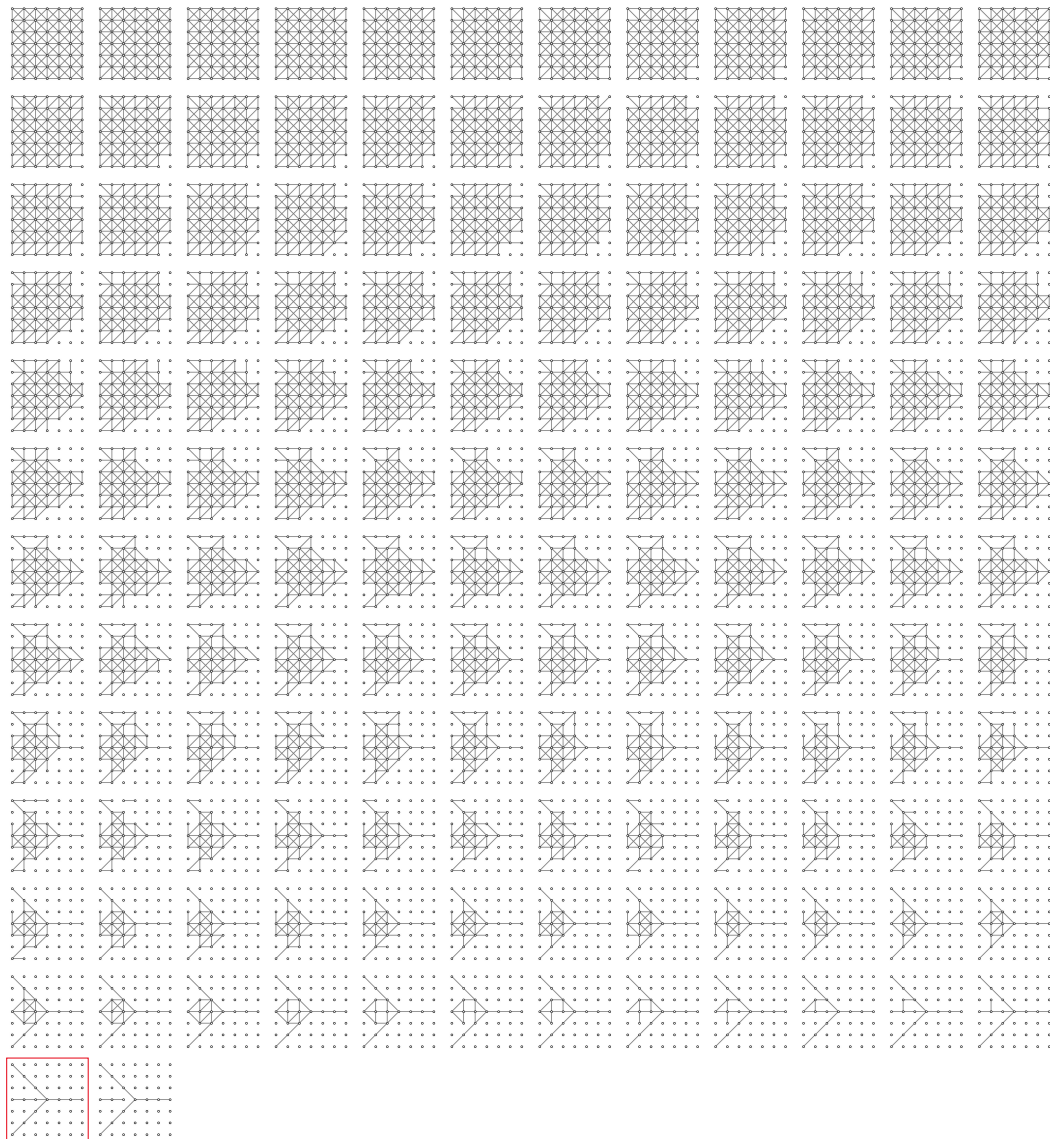


(b)

Figure 4.25: Loading condition L1 of 6×6 -grid truss (reversed the order of load cases, $\gamma = 0.9$). (a) Initial GS. (b) Removal sequence of members.

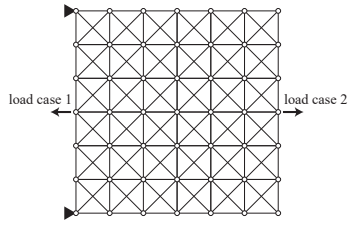


(a)

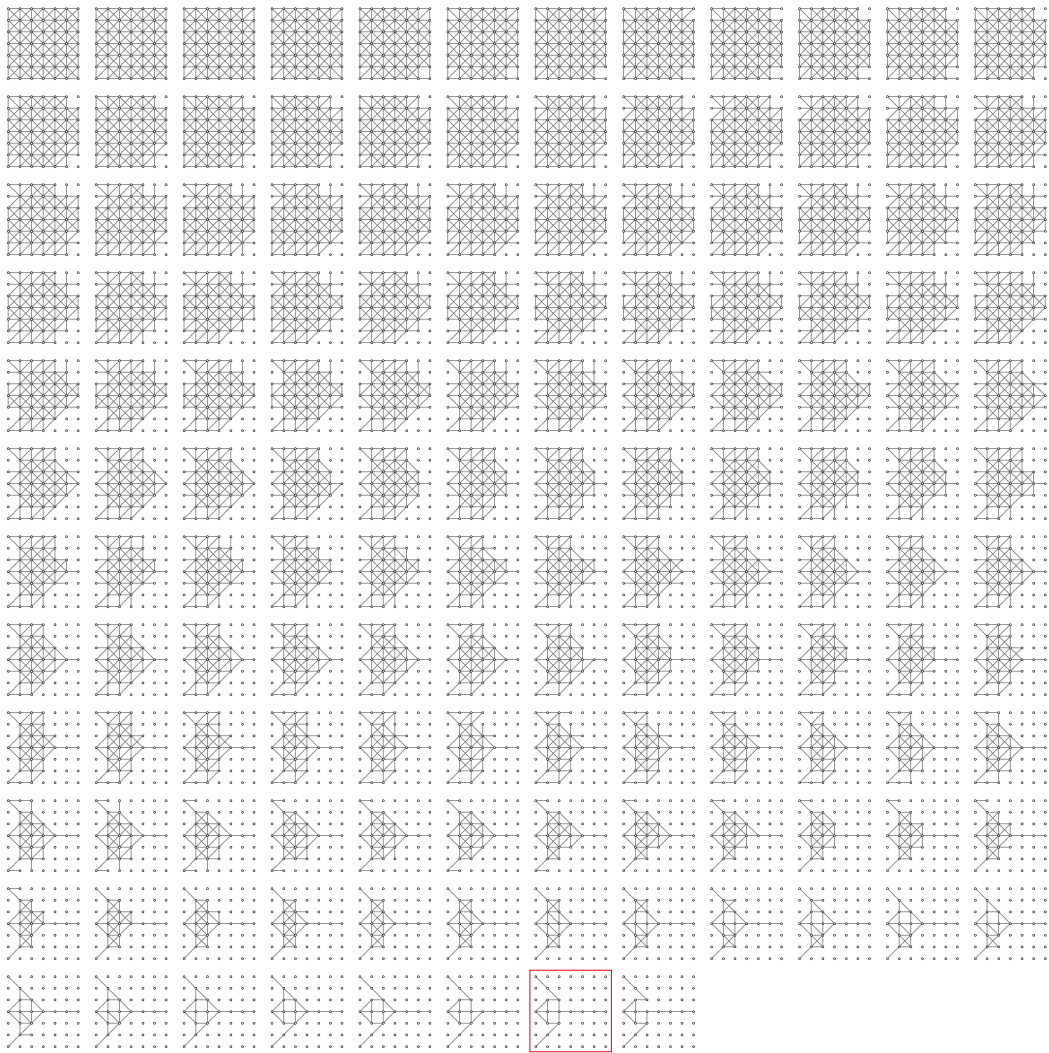


(b)

Figure 4.26: Loading condition L2 of 6×6 -grid truss ($\gamma = 0.99$). (a) Initial GS. (b) Removal sequence of members.

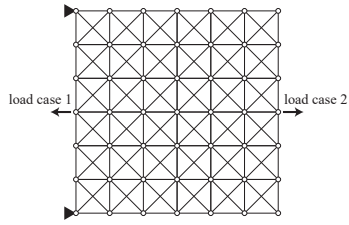


(a)

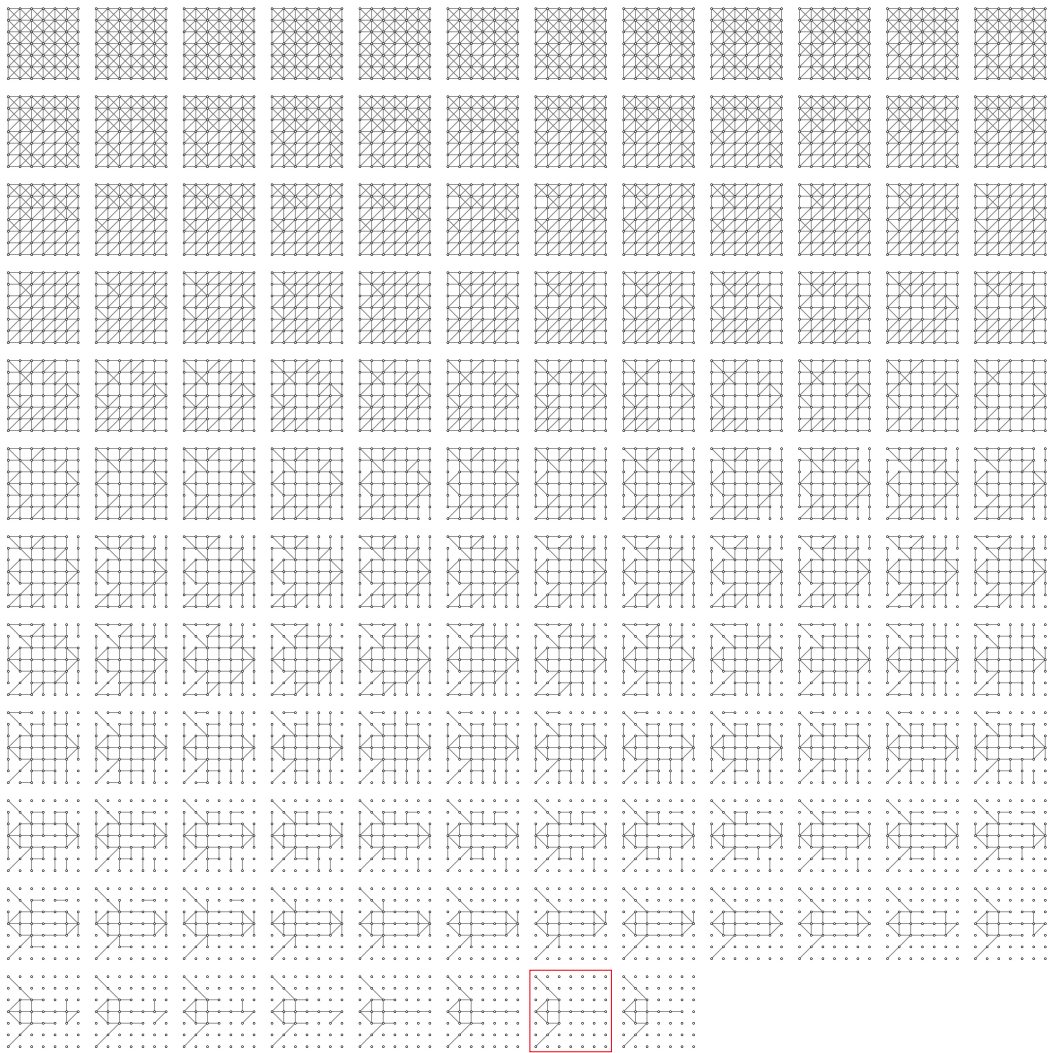


(b)

Figure 4.27: Loading condition L2 of 6×6 -grid truss (reversed the order of load cases, $\gamma = 0.99$). (a) Initial GS. (b) Removal sequence of members.

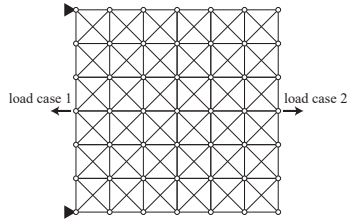


(a)

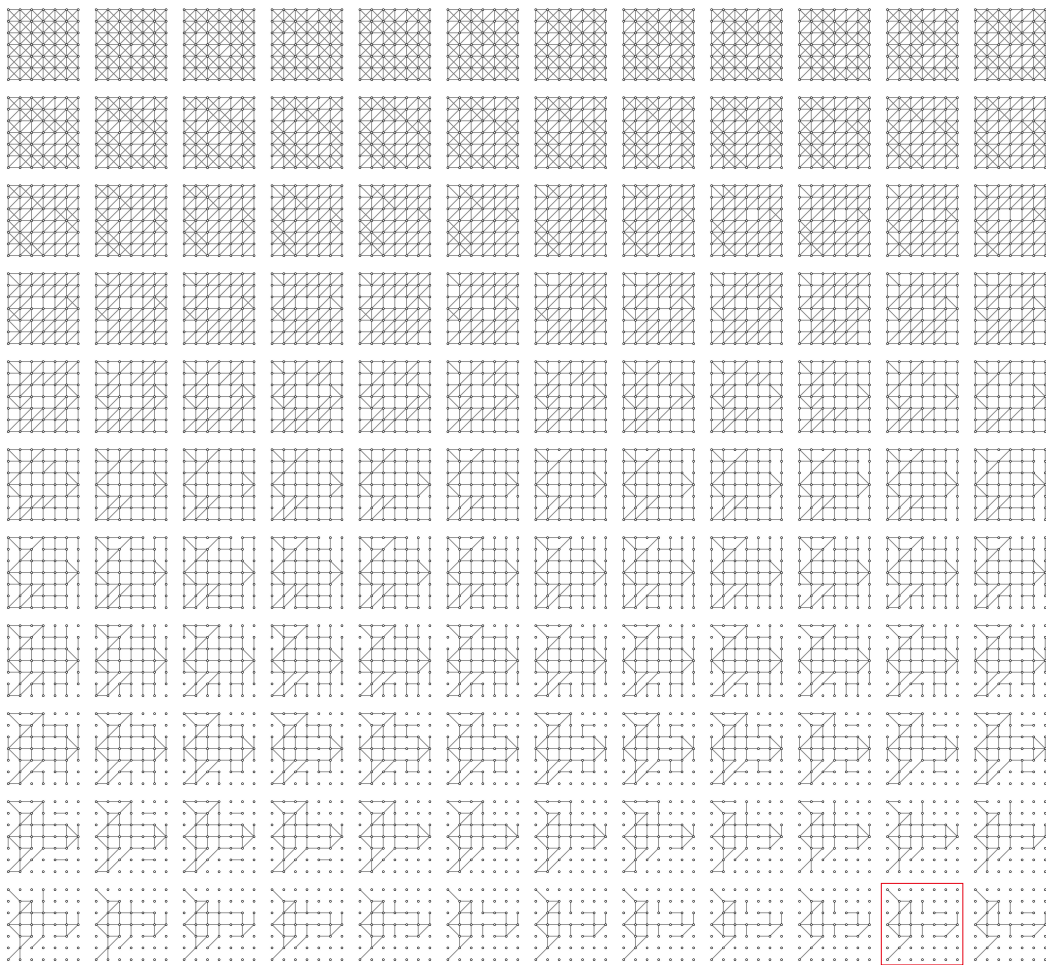


(b)

Figure 4.28: Loading condition L2 of 6×6 -grid truss ($\gamma = 0.9$). (a) Initial GS.
 (b) Removal sequence of members.



(a)



(b)

Figure 4.29: Loading condition L2 of 6×6 -grid truss (reversed the order of load cases, $\gamma = 0.9$). (a) Initial GS. (b) Removal sequence of members.

4.4.4 Comparison of efficiency and accuracy with genetic algorithm

Among the trained RL agents in the previous examples, the agent trained with $\alpha = 5.0 \times 10^{-5}$ and $\gamma = 0.99$ is further investigated in terms of efficiency and accuracy through comparison with a GA. GAs are one of the most prevalent metaheuristic approaches for binary optimization problems, which are inspired by the process of natural selection [13]. In GAs, a set of solutions are repeatedly modified using the operations such as *selection*, where superior solutions at the current generation are selected for the next generation, *crossover*, where the selected solutions are combined to breed child solutions sharing the same characteristics as the parents, and *mutation*, where the selected solutions randomly change their values with a low probability. A simple GA method used in this section is explained in Table 4.12. This algorithm is terminated if the best cost function value f_b is not updated for $n_t = 10$ consecutive generations.

Because the optimization problem Eq. (4.3) contains constraint functions, the cost function F used in the GA is defined using the penalty term as

$$F(\mathbf{A}) = V_s(\mathbf{A}) + b_1 C_1(\mathbf{A}) + b_2 C_2(\mathbf{A}) \quad (4.5a)$$

$$C_1(\mathbf{A}) = \max \left\{ \left(\max_{i \in \Omega_m, j \in \{1, \dots, n_L\}} \left(\frac{|\sigma_{i,j}(\mathbf{A})|}{\bar{\sigma}} \right) - 1 \right), 0 \right\} \quad (4.5b)$$

$$C_2(\mathbf{A}) = \max \left\{ \left(\max_{i \in \Omega_d, j \in \{1, \dots, n_L\}} \left(\frac{|u_{i,j}(\mathbf{A})|}{\bar{u}} \right) - 1 \right), 0 \right\} \quad (4.5c)$$

where b_1 and b_2 are penalty coefficients for stress and displacement constraints; both are set to be 1000 in this study. If the stress and displacement constraints are satisfied, the penalty terms become zero and the cost function becomes equivalent to the total structural volume $V_s(\mathbf{A})$. The same material property and constraints as the examples of RL are applied to the following problems. The GA algorithm is run 10 times with different initial solutions that are generated randomly, and only the best result that yields a solution with the least total structural volume is provided in the GA column in Table 4.13.

The benchmark solutions for the 3×2 -grid truss provided in Table 4.13 are global optimal solutions; this global optimality is verified through enumeration which took 44.1 hours for each boundary condition. The other solutions are assumed to be global optima which have not been verified through enumeration due to extremely high computational costs.

Although $n_{\text{load}}! = 2$ different removal sequences can be obtained by re-ordering the load cases, only the better result with less total structural volume is provided in the RL+GE column in Table 4.13. Note that the total CPU time $t_c[\text{s}]$ for obtaining this removal sequence of members includes initialization of the truss structure, importing the trained RL agent, and computing the removal sequence.

According to Table 4.13, the proposed RL+GE method is much more efficient than the GA; the CPU time exponentially increases as the number of variable increases in the GA; on the other hand, the CPU time increases almost linearly in RL+GE. The proposed method is also comparable to the GA with $n_p = 200$ in terms of proximity to the global optimum; RL+GE generally

Table 4.12: GA method used in this study

input: F : cost function, $n_p (= 200)$: number of solutions, n_m : number of variables, $n_t (= 10)$: stopping criterion for change of f_b ,
 $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_{n_p}\}$: initial solutions, $F(\mathbf{x})$: cost function,
 $r_e (= 0.2)$: elite rate, $r_m (= 0.1)$: mutation probability

output: \mathbf{x}_b : best feasible solution

$I \leftarrow 0$
 $n_e \leftarrow \text{round}(r_e n_p)$
 $f_b \leftarrow \infty$

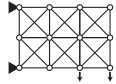
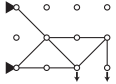
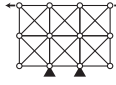
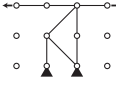

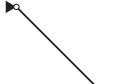



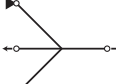
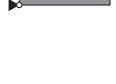
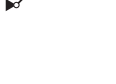
while $I \leq n_t$ **do**
 $I \leftarrow I + 1$
 for $i \leftarrow 1$ **to** n_p **do**
 $f_i \leftarrow F(\mathbf{x}_i)$
 for $i \leftarrow 1$ **to** n_p **do**
 if $f_i < f_b$ **and** \mathbf{x}_i satisfy constraints **then**
 $\mathbf{x}_b \leftarrow \mathbf{x}_i$
 $f_b \leftarrow f_i$
 $I \leftarrow 0$

$\mathbf{X}_e \leftarrow n_e$ elite solutions from \mathbf{X}
 $\mathbf{X} \leftarrow \mathbf{X}_e$
 for $i \leftarrow n_e + 1$ **to** n_p **do**
 if a sample from uniform distribution $[0, 1] > r_m$ **then**
 $\mathbf{x}_{j_1}, \mathbf{x}_{j_2} \leftarrow$ two randomly chosen solutions from \mathbf{X}_e
 $k \leftarrow$ randomly chosen integer from $\{1, \dots, n_m - 1\}$
 $\mathbf{x}_i \leftarrow \{x_{j_1,1}, \dots, x_{j_1,k}, x_{j_2,k+1}, \dots, x_{j_2,n_m}\}$
 else
 $\mathbf{x}_i \leftarrow$ randomly chosen solution from \mathbf{X}_e
 $k \leftarrow$ randomly chosen integer from $\{1, \dots, n_m\}$
 if $x_{i,k} = \bar{A}$ **then**
 $x_{i,k} \leftarrow \bar{A} \times 10^{-6}$
 else
 $x_{i,k} \leftarrow \bar{A}$
 append \mathbf{x}_i into \mathbf{X}

return \mathbf{x}_b

reached the feasible solutions with less total structural volume compared with the solutions obtained by the GA. In addition, the accuracy of the trained agent is less dependent on the problem size; the trained agent reached the presumable global optimum for the 10×10 -grid truss with L2 loading condition, although the agent was caught at the bad solution $V_s = 0.0608$ for the 8×8 -grid truss with L1 loading condition.

Table 4.13: Comparison between the proposed method (RL+GE) and the GA in view of the total structural volume $V_s[\text{m}^3]$ and CPU time for one optimization $t_c[\text{s}]$ using benchmark solutions

problem	grids	n_m	RL+GE	GA	benchmark
	3×2 (B1)	29	$\begin{cases} t_c = 0.2 \\ V_s = 0.0121 \end{cases}$	$\begin{cases} t_c = 1.6 \\ V_s = 0.0107 \end{cases}$	 $V_s = 0.0107$
	3×2 (B2)	29	$\begin{cases} t_c = 0.2 \\ V_s = 0.0088 \end{cases}$	$\begin{cases} t_c = 1.5 \\ V_s = 0.0088 \end{cases}$	 $V_s = 0.0088$
	4×4	72	$\begin{cases} t_c = 0.5 \\ V_s = 0.0097 \end{cases}$	$\begin{cases} t_c = 3.7 \\ V_s = 0.0097 \end{cases}$	$V_s = 0.0097$
	6×6 (L1)	156	$\begin{cases} t_c = 1.2 \\ V_s = 0.0145 \end{cases}$	$\begin{cases} t_c = 12.2 \\ V_s = 0.0145 \end{cases}$	 $V_s = 0.0145$
	8×8	272	$\begin{cases} t_c = 2.3 \\ V_s = 0.0608 \end{cases}$	$\begin{cases} t_c = 54.8 \\ V_s = 0.0284 \end{cases}$	 $V_s = 0.0193$
	10×10	420	$\begin{cases} t_c = 6.2 \\ V_s = 0.0281 \end{cases}$	$\begin{cases} t_c = 140.3 \\ V_s = 0.0658 \end{cases}$	$V_s = 0.0241$
	4×4	72	$\begin{cases} t_c = 0.5 \\ V_s = 0.0097 \end{cases}$	$\begin{cases} t_c = 3.7 \\ V_s = 0.0097 \end{cases}$	$V_s = 0.0097$
	6×6 (L2)	156	$\begin{cases} t_c = 1.2 \\ V_s = 0.0145 \end{cases}$	$\begin{cases} t_c = 11.9 \\ V_s = 0.0145 \end{cases}$	 $V_s = 0.0145$
	8×8	272	$\begin{cases} t_c = 2.6 \\ V_s = 0.0233 \end{cases}$	$\begin{cases} t_c = 41.8 \\ V_s = 0.0473 \end{cases}$	 $V_s = 0.0193$
	10×10	420	$\begin{cases} t_c = 5.6 \\ V_s = 0.0241 \end{cases}$	$\begin{cases} t_c = 137.7 \\ V_s = 0.0380 \end{cases}$	$V_s = 0.0241$

4.5 Generation of initial solutions by a trained agent for simultaneous optimization of geometry and topology of trusses using force density method

In this section, the sparse topology obtained through the proposed method is utilized as an initial solution for solving the simultaneous optimization problem of geometry and topology of trusses using a force density method (FDM). FDMs have originally been used for finding the self-equilibrium shape of tension structures such as cable nets and tensegrity structures [136, 137]. The force density of a bar member is defined as the ratio of the axial force to the length, and the equilibrium nodal locations are obtained by solving a set of linear equations using a force density matrix, which is defined by the force densities of all members.

In Refs. [72, 73], the author used a force density method for simultaneous optimization of geometry and topology of trusses. The optimization problem to minimize the compliance under a constraint on the total structural volume can be formulated as functions of force densities only. The computational cost can be saved compared with the standard methods where the nodal coordinates and the cross-sectional areas are assigned as design variables. The numerical difficulty caused by coalescent (melting) nodes can be easily avoided by setting upper bounds for the design variables. Therefore, it is no more necessary to set constraints on nodal locations and thus various optimal solutions of topology and geometry can be generated from a relatively sparse initial GS.

The computational cost of the above optimization is highly dependent on the number of members because the number of variables is equal to that of members in the optimization problem. In the following, a sparse refined initial solution is generated from a densely connected regular GS using the trained agent in order to reduce the computational cost for the optimization with FDM.

4.5.1 Force density method for obtaining nodal locations

The force density of member i is defined with respect to the axial force N_i and the length L_i as

$$q_i = \frac{N_i}{L_i} \quad (4.6)$$

Using a force density vector $\mathbf{q} = [q_1, \dots, q_{n_m}]^T$ and a connectivity matrix of the truss \mathbf{C} , the force density matrix $\mathbf{E} \in \mathbb{R}^{n_n \times n_n}$ is defined as

$$\mathbf{E} = \mathbf{C}^T \text{diag}(\mathbf{q}) \mathbf{C} \quad (4.7)$$

The diagonal element E_{jj} of the symmetric matrix \mathbf{E} is equal to the total amount of force densities of members connecting to node j , and off-diagonal element E_{jk} ($j \neq k$) is equal to $-q_i$, if node k is connected to node j by member i . Let $[\Delta_{i,x}, \Delta_{i,y}, \Delta_{i,z}]$ the unitary directional vector of the member i . Then $q_{i,x}$, $q_{i,y}$ and $q_{i,z}$, x , y - and z -directional components of the force density q_i , respectively, are expressed as

$$q_{i,x} = \frac{N_i |\Delta_{i,x}|}{L_i |\Delta_{i,x}|} = \frac{N_i}{L_i} = q_i \quad (4.8a)$$

$$q_{i,y} = \frac{N_i |\Delta_{i,y}|}{L_i |\Delta_{i,y}|} = \frac{N_i}{L_i} = q_i \quad (4.8b)$$

$$q_{i,z} = \frac{N_i |\Delta_{i,z}|}{L_i |\Delta_{i,z}|} = \frac{N_i}{L_i} = q_i \quad (4.8c)$$

Equation (4.8) implies that the same matrix \mathbf{E} can be used for obtaining the nodal coordinates \mathbf{x} , \mathbf{y} and $\mathbf{z} \in \mathbb{R}^{n_n}$ as

$$\mathbf{E}\mathbf{x} = \mathbf{P}_x \quad (4.9a)$$

$$\mathbf{E}\mathbf{y} = \mathbf{P}_y \quad (4.9b)$$

$$\mathbf{E}\mathbf{z} = \mathbf{P}_z \quad (4.9c)$$

where \mathbf{P}_x , \mathbf{P}_y , and $\mathbf{P}_z \in \mathbb{R}^{n_n}$ are the nodal load vectors including the reactions in x -, y -, and z -directions, respectively. Equations (4.9a) - (4.9c) are sets of simultaneous linear equations with respect to the nodal coordinates. Therefore, the nodal locations are directly obtained by solving these equations, which is more efficient compared with solving stiffness equations in which nodal displacements are obtained, and then nodal coordinates are obtained from the displacements. The details can be found in [136, 138, 139].

In the conventional FDMs, the definition of free and fixed coordinates depends on the support condition only; the nodal coordinates whose displacement is constrained by a pin support or roller support are regarded as fixed. Instead, *fixed* coordinates are those that do not change locations during the optimization in the following formulation. Therefore, fixed nodes include all pin supports, roller supports, and loaded nodes hereafter. This implies that the numbers of free and fixed coordinates, respectively, in x - y - and z -directions are all the same.

The nodes are re-ordered such that nodal coordinates free to move during the optimization precede those fixed as

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_{\text{free}} \\ \mathbf{x}_{\text{fix}} \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} \mathbf{y}_{\text{free}} \\ \mathbf{y}_{\text{fix}} \end{pmatrix}, \quad \mathbf{z} = \begin{pmatrix} \mathbf{z}_{\text{free}} \\ \mathbf{z}_{\text{fix}} \end{pmatrix} \quad (4.10)$$

The force density matrices in three directions are combined to $\tilde{\mathbf{E}} \in \mathbb{R}^{3n_n \times 3n_n}$ as

$$\tilde{\mathbf{E}} = \left(\begin{array}{c|c} \overbrace{\begin{array}{ccc} \mathbf{E}_{\text{free}}^x & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{E}_{\text{free}}^y & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{E}_{\text{free}}^z \end{array}}^{n_{\text{free}}} & \overbrace{\begin{array}{ccc} \mathbf{E}_{\text{link}}^x & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{E}_{\text{link}}^y & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{E}_{\text{link}}^z \end{array}}^{n_{\text{fix}}} \\ \hline \begin{array}{ccc} \mathbf{E}_{\text{link}}^{x\top} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{E}_{\text{link}}^{y\top} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{E}_{\text{link}}^{z\top} \end{array} & \begin{array}{ccc} \mathbf{E}_{\text{fix}}^x & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{E}_{\text{fix}}^y & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{E}_{\text{fix}}^z \end{array} \end{array} \right) \left. \begin{array}{l} \\ \\ \\ \\ \end{array} \right\} \begin{array}{l} n_{\text{free}} \\ \\ \\ n_{\text{fix}} \end{array} \quad (4.11)$$

where n_{free} and n_{fix} are the numbers of free and fixed coordinates satisfying

$$n_{\text{free}} + n_{\text{fix}} = 3n_n \quad (4.12)$$

Matrices $(\mathbf{E}_{\text{free}}^x, \mathbf{E}_{\text{free}}^y, \mathbf{E}_{\text{free}}^z)$, $(\mathbf{E}_{\text{link}}^x, \mathbf{E}_{\text{link}}^y, \mathbf{E}_{\text{link}}^z)$, and $(\mathbf{E}_{\text{fix}}^x, \mathbf{E}_{\text{fix}}^y, \mathbf{E}_{\text{fix}}^z)$ are assembled into $\tilde{\mathbf{E}}_{\text{free}} \in \mathbb{R}^{n_{\text{free}} \times n_{\text{free}}}$, $\tilde{\mathbf{E}}_{\text{link}} \in \mathbb{R}^{n_{\text{free}} \times n_{\text{free}}}$, and $\tilde{\mathbf{E}}_{\text{link}} \in \mathbb{R}^{n_{\text{free}} \times n_{\text{fix}}}$, and $\tilde{\mathbf{E}}_{\text{fix}} \in \mathbb{R}^{n_{\text{fix}} \times n_{\text{fix}}}$, respectively. Then, $\tilde{\mathbf{E}}$ in Eq. (4.11) is rewritten as

$$\tilde{\mathbf{E}} = \begin{pmatrix} \tilde{\mathbf{E}}_{\text{free}} & \tilde{\mathbf{E}}_{\text{link}} \\ \tilde{\mathbf{E}}_{\text{link}}^\top & \tilde{\mathbf{E}}_{\text{fix}} \end{pmatrix} \quad (4.13)$$

Let $\mathbf{P}_{\text{free}} \in \mathbb{R}^{n_{\text{free}}}$ and $\mathbf{P}_{\text{fix}} \in \mathbb{R}^{n_{\text{fix}}}$ denote the nodal load vector corresponding to $\mathbf{X}_{\text{free}} = (\mathbf{x}_{\text{free}}^\top, \mathbf{y}_{\text{free}}^\top, \mathbf{z}_{\text{free}}^\top)^\top$ and $\mathbf{X}_{\text{fix}} = (\mathbf{x}_{\text{fix}}^\top, \mathbf{y}_{\text{fix}}^\top, \mathbf{z}_{\text{fix}}^\top)^\top$. The set of equilibrium equations in Eqs. (4.9a) - (4.9c) is rewritten as

$$\begin{pmatrix} \tilde{\mathbf{E}}_{\text{free}} & \tilde{\mathbf{E}}_{\text{link}} \\ \tilde{\mathbf{E}}_{\text{link}}^\top & \tilde{\mathbf{E}}_{\text{fix}} \end{pmatrix} \begin{pmatrix} \mathbf{X}_{\text{free}} \\ \mathbf{X}_{\text{fix}} \end{pmatrix} = \begin{pmatrix} \mathbf{P}_{\text{free}} \\ \mathbf{P}_{\text{fix}} \end{pmatrix} \quad (4.14)$$

From Eq. (4.14), the locations of free nodes \mathbf{X}_{free} are obtained as

$$\tilde{\mathbf{E}}_{\text{free}} \mathbf{X}_{\text{free}} = \mathbf{P}_{\text{free}} - \tilde{\mathbf{E}}_{\text{link}} \mathbf{X}_{\text{fix}} \quad (4.15)$$

Since the loaded nodes are regarded as fixed, \mathbf{P}_{free} is a zero vector for any arbitrary truss. Thus, the term \mathbf{P}_{free} in Eq. (4.15) can be omitted as

$$\tilde{\mathbf{E}}_{\text{free}} \mathbf{X}_{\text{free}} = -\tilde{\mathbf{E}}_{\text{link}} \mathbf{X}_{\text{fix}} \quad (4.16)$$

If the locations of fixed nodes are known and the force densities of all members are assigned, \mathbf{X}_{fix} , $\tilde{\mathbf{E}}_{\text{free}}$ and $\tilde{\mathbf{E}}_{\text{link}}$ can be computed from them. Therefore, \mathbf{X}_{free} is considered as a function of force densities.

Similarly, \mathbf{P}_{fix} is computed from Eq. (4.14) as

$$\mathbf{P}_{\text{fix}} = \tilde{\mathbf{E}}_{\text{link}}^\top \mathbf{X}_{\text{free}} + \tilde{\mathbf{E}}_{\text{fix}} \mathbf{X}_{\text{fix}} \quad (4.17)$$

\mathbf{P}_{fix} can be regarded as the vector of reaction forces $\mathbf{R} = \{R_1, \dots, R_{n_{\text{fix}}}\} \in \mathbb{R}^{n_{\text{fix}}}$ corresponding to \mathbf{X}_{fix} . Since the loaded nodes are regarded as fixed, the reaction forces at loaded nodes must be specified by adding the following constraint:

$$\sum_{i \in \Omega_R} (R_i - \bar{P}_i)^2 = 0 \quad (4.18)$$

where Ω_R is the set of indices corresponding to the loaded nodes and \bar{P}_i is the specified load value.

The square of the length of member i connecting nodes j and k is computed as

$$L_i^2 = (\mathbf{X}_k - \mathbf{X}_j)^\top (\mathbf{X}_k - \mathbf{X}_j) \quad (4.19)$$

where $\mathbf{X}_j \in \mathbb{R}^3$ and $\mathbf{X}_k \in \mathbb{R}^3$ are the nodal coordinate vectors containing the x -, y - and z - coordinates of nodes j and k , respectively. Since each nodal location is a function of force densities, the member length is also a function of force densities \mathbf{q} .

4.5.2 Optimization problem

Consider a problem for minimizing the compliance under a constraint on the total structural volume. The solution to this problem is a statically determinate truss with the same absolute value of axial stress for all members [140, 141]. Let $\bar{\sigma}$ denote the absolute value of stress for existing members of the optimal solution. From member length L_i , the cross-section of i -th member A_i is expressed as

$$\begin{aligned} A_i &= \frac{|N_i|}{\bar{\sigma}} \\ &= \frac{|q_i|L_i}{\bar{\sigma}} \end{aligned} \quad (4.20)$$

The total structural volume V_s is computed as

$$\begin{aligned} V &= \sum_{i=1}^{n_m} A_i L_i \\ &= \sum_{i=1}^{n_m} \frac{|q_i|L_i^2}{\bar{\sigma}} \end{aligned} \quad (4.21)$$

The compliance is obtained as twice the sum of strain energies of each member as

$$\begin{aligned} F &= 2 \sum_{i=1}^{n_m} \frac{A_i L_i}{2E} \bar{\sigma}^2 \\ &= 2 \sum_{i=1}^{n_m} \frac{\bar{\sigma} |q_i| L_i^2}{2E} \\ &= \sum_{i=1}^{n_m} \frac{\bar{\sigma} |q_i| L_i^2}{E} \end{aligned} \quad (4.22)$$

From Eq. (4.20), $\bar{\sigma}$ is regarded as a scaling parameter for A_i of a statically determinate truss, for which N_i is independent of A_i . The product of the total structural volume and the compliance is computed as

$$\begin{aligned}
VF &= \sum_{i=1}^{n_m} \frac{q_i^2 L_i^4}{E} \\
&= 2 \sum_{i=1}^{n_m} \frac{N_i^2 L_i^2}{E}
\end{aligned} \tag{4.23}$$

As seen in Eq. (4.23), VF is independent of $\bar{\sigma}$, which implies that the total structural volume can be calculated after the optimization is implemented by minimizing the compliance with an arbitrary positive value of $\bar{\sigma}$. The tradeoff between the total structural volume and the compliance is illustrated in Fig. 4.30; V_s and F are inversely proportional under the condition that $\bar{\sigma}$ is a constant.

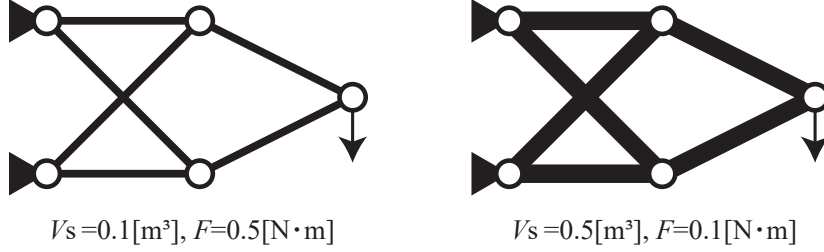


Figure 4.30: The relationship of the total structural volume V_s and the compliance F under the condition of a constant absolute value of stresses $\bar{\sigma}$

Let c denote a very small positive value. We further introduce smoothing in computing the compliance as

$$\tilde{F} = \sum_{i=1}^{n_m} \frac{\bar{\sigma} L_i^2 \sqrt{q_i^2 + c}}{E} \tag{4.24}$$

where the absolute value of each force density $|q_i|$ is replaced by $\sqrt{q_i^2 + c}$ for the purpose of smoothing the compliance. Figure 4.31 illustrates the difference of two functions: the dotted line does not apply the smoothing and the single line does. The discontinuity of the gradient at $q_i = 0$, which may cause difficulty of convergence, can be avoided by introducing the smoothing. The effect on the compliance can be neglected if the parameter c is sufficiently small. c is set to be 1.0×10^{-6} in this study.

Finally, the optimization problem is formulated as

$$\text{Minimize}_{\mathbf{q}} \quad \tilde{F} = \sum_{i=1}^{n_m} \frac{\bar{\sigma} L_i^2 \sqrt{q_i^2 + c}}{E} \tag{4.25a}$$

$$\text{Subject to} \quad \sum_{i \in \Omega_R} (R_i - \bar{P}_i)^2 = 0 \tag{4.25b}$$

$$q_i^L < q_i < q_i^U \tag{4.25c}$$

where q_i^L and q_i^U are the lower and upper bounds of the force density of member i , respectively.

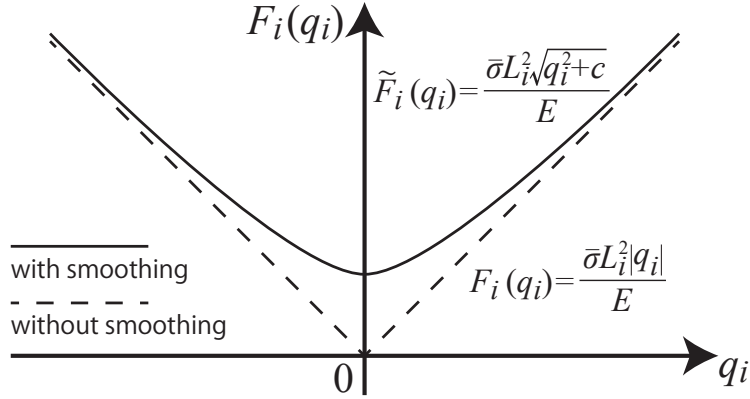


Figure 4.31: Compliance of member i with and without smoothing

4.5.3 Sensitivity analysis

The gradient of the objective and constraint functions are analytically obtained in order to reduce the computational cost of the gradient-based optimization method. For simplicity, only the sensitivity with respect to the force density of member i is to be focused. The objective function Eq. (4.24) is differentiated with respect to q_i as

$$\frac{\partial \tilde{F}}{\partial q_i} = \frac{\bar{\sigma} q_i L_i^2}{E \sqrt{q_i^2 + c}} + \sum_{i=1}^{n_m} \left(\frac{\bar{\sigma} \sqrt{q_i^2 + c}}{E} \cdot \frac{\partial L_i^2}{\partial q_i} \right) \quad (4.26)$$

From Eq. (4.19), the sensitivity coefficient of L_i^2 with respect to q_i is obtained as

$$\frac{\partial L_i^2}{\partial q_i} = 2 (\mathbf{X}_k - \mathbf{X}_j)^\top \frac{\partial (\mathbf{X}_k - \mathbf{X}_j)}{\partial q_i} \quad (4.27)$$

The constraint function in Eq. (4.18) is differentiated with respect to q_i as

$$\sum_{i \in \Omega_R} 2 (R_i - \bar{P}_i) \frac{\partial R_i}{\partial q_i} = 0 \quad (4.28)$$

The sensitivity coefficients of the reaction forces are obtained from Eq. (4.17) as

$$\frac{\partial \mathbf{R}}{\partial q_i} = \frac{\partial \tilde{\mathbf{E}}_{\text{link}}^\top}{\partial q_i} \mathbf{X}_{\text{free}} + \tilde{\mathbf{E}}_{\text{link}}^\top \frac{\partial \mathbf{X}_{\text{free}}}{\partial q_i} + \frac{\partial \tilde{\mathbf{E}}_{\text{fix}}}{\partial q_i} \mathbf{X}_{\text{fix}} \quad (4.29)$$

Differentiation of Eq. (4.16) with respect to q_i leads to

$$\tilde{\mathbf{E}}_{\text{free}} \frac{\partial \mathbf{X}_{\text{free}}}{\partial q_i} + \frac{\partial \tilde{\mathbf{E}}_{\text{free}}}{\partial q_i} \mathbf{X}_{\text{free}} = - \frac{\partial \tilde{\mathbf{E}}_{\text{link}}}{\partial q_i} \mathbf{X}_{\text{fix}} \quad (4.30)$$

Equations (4.26) - (4.29) implies that the sensitivity coefficients of objective and constraint functions can be obtained after obtaining $\partial \mathbf{X}_{\text{free}} / \partial q_i$ by solving a set of linear equations (4.30).

4.5.4 Workflow

The workflow of the overall optimization process is described in Fig. 4.32. The topology of an initial GS consisting of regular grids is optimized using the agent trained with the proposed method RL+GE with $\gamma = 0.99$. As in the previous sections, the same elastic modulus $E = 200$ [kN/mm²] and the same initial cross-sectional area $A_i = 1000$ [mm²] ($i = 1, \dots, n_m$) are used. Then, isolated nodes with no member connected and unstable nodes with two members linearly connected are removed.

The simplified topology after this process becomes an initial solution for the optimization with the FDM. SNOPT Ver. 7.2 [142], a package for solving optimization problems based on the sequential quadratic programming (SQP) [143, 144], is used as an optimization tool. The initial variables are assigned by solving the stiffness equation (4.1) and computing the force densities from the obtained displacements. The lower and upper bounds for the variables in the optimization problem (4.25) are $q_i^L = -1000$ and $q_i^U = 1000$ ($i = 1, \dots, n_m$), respectively. Since only a single loading condition can be handled with the optimization with the FDM, the following examples assume that loads are applied to the structure simultaneously as one loading condition when optimizing with the FDM. The total structural volume after the optimization with the FDM is scaled to \bar{V} that is equal to the total structural volume of the structure after the topology optimization by RL+GE, in order to compare the results between the FDM only and combination of a RL+GE and the FDM.

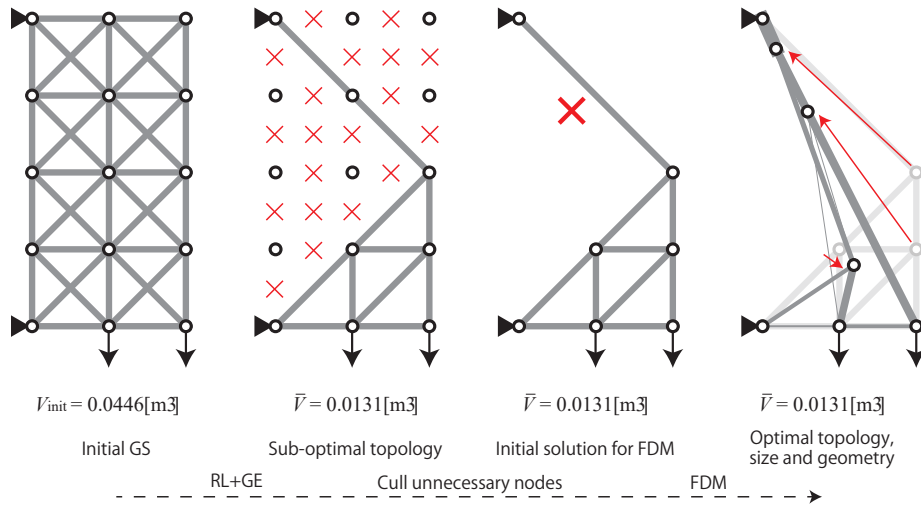


Figure 4.32: The optimization workflow combining RL+GE and the FDM

4.5.5 Numerical examples

3 × 2-grid truss

The trained agent is first applied to a 3 × 2-grid truss as shown in Fig. 4.33. The left corner nodes are pin-supported, and a downward load of 1 kN is applied at the lowest end and the node left to the lowest end. The upper sequence

is the optimization result with the FDM only and the bottom sequence is the optimization result with RL+GE and the FDM. The total structural volume $V_{\text{init}} = 0.0340 \text{ [m}^3\text{]}$ of the initial GS is reduced to $\bar{V} = 0.0121 \text{ [m}^3\text{]}$ by the RL agent. The size of the members of the optimal solution after the FDM is rescaled to \bar{V} . The compliance at each phase is also shown in Fig. 4.33; the compliance of the optimal solution $F = 0.0190 \text{ [kN} \cdot \text{m]}$ obtained by the FDM only is slightly smaller than $F = 0.0192 \text{ [kN} \cdot \text{m]}$ that obtained by RL+GE and the FDM. However, the configuration of the latter solution is much simpler than the former. Considering that a good solution comparable to the optimal solution by the FDM only is obtained with a much smaller number of nodes and members, it can be said that the solution obtained by the proposed optimization workflow RL+GE and the FDM is more preferable from the viewpoint of constructability.

The elapsed CPU time t_c is the average of 10 simulations. Since no random process such as the ϵ -greedy policy and the randomization of the initial variables are included, all the 10 simulations converge to the same solution. Compared with the optimization with the FDM only, the combination of RL+GE and the FDM is much more computationally efficient; the total elapsed CPU time is reduced by approximately 40% by introducing the RL agent. The computation time to remove unnecessary nodes is small enough to be neglected in this example.

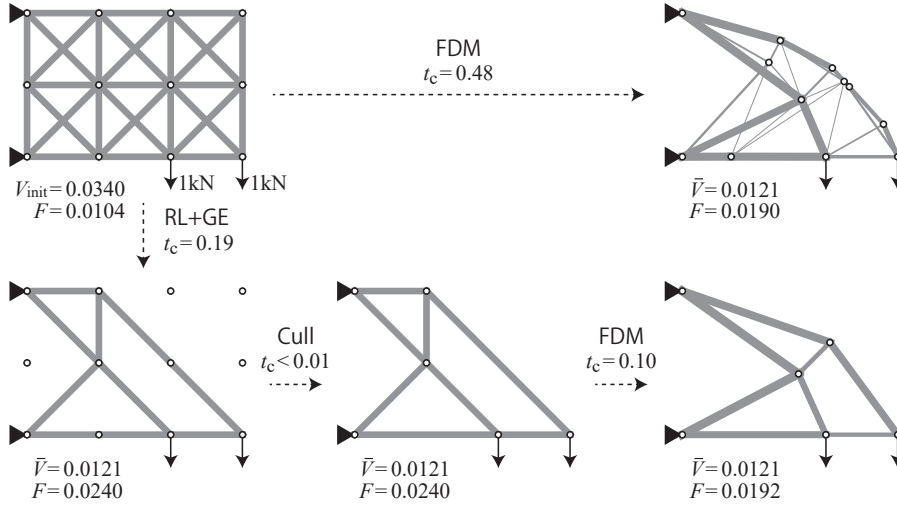


Figure 4.33: The transition of the total structural volume $V_s \text{ [m}^3\text{]}$ and the compliance $F \text{ [kN} \cdot \text{m]}$ of the 3×2 -grid truss with the elapsed CPU time $t_c \text{ [s]}$. The upper row is the optimization result with the FDM only. The bottom row is the optimization result with RL+GE and the FDM.

4×4 -grid truss

The performance of the proposed optimization workflow is also demonstrated through a 4×4 -grid truss as shown in Fig. 4.34. The left corner nodes are pin-supported, and the lowest end and the node left to the lowest end are subject to a downward load of 1 kN. Similarly to Fig. 4.33, the upper row is the optimization result with the FDM only and the bottom row is the optimization result with

RL+GE and the FDM. The total structural volume $V_{\text{init}} = 0.0853 \text{ [m}^3\text{]}$ of the initial GS is reduced to $\bar{V} = 0.0281 \text{ [m}^3\text{]}$ by the RL agent. The number of nodes is reduced from 25 to 12 after removing the unnecessary nodes. Two nodes are further merged to other nodes after the optimization with the FDM to create a simple configuration with 10 nodes and $F = 0.0132 \text{ [kN} \cdot \text{m]}$, while the solution obtained by the FDM only is a very complex configuration with $F = 0.0125 \text{ [kN} \cdot \text{m]}$. Given that the number of nodes is also an important factor to reduce the construction cost, the solution obtained by the proposed workflow RL+GE and the FDM may be a more realistic option for construction.

As seen from t_c that is sampled from 10 repetitive simulations, the effect of reducing the computational cost by the proposed method is more remarkable for a truss with a more complicated initial GS. The total elapsed CPU time is reduced to 1/3 by introducing the RL agent. The computational cost to cull unnecessary nodes is small enough to be neglected in this example, too.

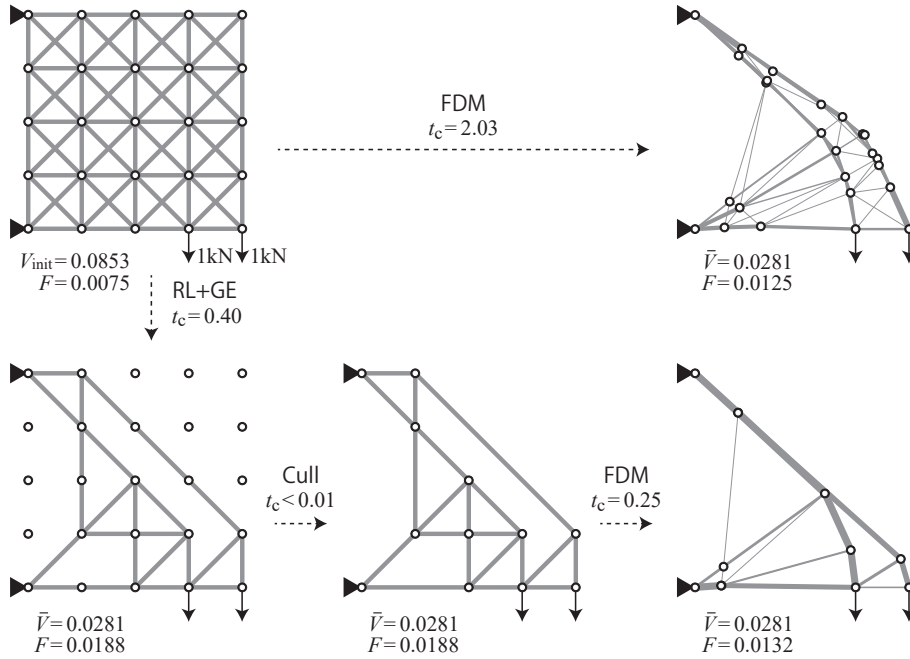


Figure 4.34: The transition of the total structural volume $V_s \text{ [m}^3\text{]}$ and the compliance $F \text{ [kN} \cdot \text{m]}$ of the 4×4 -grid truss with the elapsed CPU time $t_c \text{ [s]}$. The upper row is the optimization result with the FDM only. The bottom row is the optimization result with RL+GE and the FDM.

4.6 Conclusion

In this chapter, the RL method proposed in Chapter 3 has been demonstrated through the binary topology optimization of trusses to minimize the total structural volume under stress constraints. By varying the loading and support conditions during the training using the 4×4 -grid truss, the agent can learn the process of member removal for various force flows.

The training is implemented with different learning rates α , sizes of the edge feature n_f and discount rates γ to investigate the difference of the learning results. The RL agents that recorded good learning results are capable of removing unnecessary members from a densely connected initial GS while keeping the maximum stress that the existing members bear as small as possible. The trained RL agents are applied to 3×2 - and 6×6 -grid trusses without re-training and are found to perform well. The accuracy and computational efficiency of using the trained RL agent is compared with a GA and it is confirmed that a solution comparable to the GA can be obtained using the RL agent at a much lower computational cost than the GA.

As an application example, initial solutions for simultaneous optimization of geometry and topology of trusses using the FDM are generated using the trained RL agent. The optimization problem aims at minimizing the compliance under a constraint on the total structural volume, and has been formulated as functions of the force density only. The numerical difficulty caused by the melting nodes can be easily avoided by expressing the nodal location as a function of force density. Compared with previous approaches in which nodal locations are directly handled as design variables, the FDM-based approach does not require regions where nodes can move during optimization and can yield various optimization results in geometry and topology.

Since the number of design variables of the optimization problem by the FDM is equal to the number of the members, the computational cost can be greatly reduced by starting the optimization from an initial solution in which unnecessary members are removed in advance. In addition, more feasible optimal trusses with sparser connectivity can be obtained than simply optimizing a densely connected initial GS using the FDM only.

Chapter 5

Case study 2: Cross-section optimization of steel frames

5.1 Discrete cross-section optimization problem of planar steel frames under constraints on stress, displacement and column over-strength factor

In this section, a discrete cross-section optimization problem of planar steel frames is formulated to minimize the total structural volume under constraints on stress, displacement, and column over-strength factor (COF), which becomes the RL task to be handled in Secs. 5.2 - 5.4.

5.1.1 Allowable stress and displacement design

The design strength F_d is a reference value for determining allowable stresses of the material against tension, compression, bending, and shear forces. For steel materials, the yield stress is generally equal to its design strength under short-term loads. In this example, the values of F_d for columns and beams are 235 N/mm² and 325 N/mm², respectively.

Based on the design strength F_d , the allowable stresses of the steel against tension, compression and bending forces, denoted as f_t , f_c and f_b , respectively, are given as Table 5.1, where λ is the effective slenderness ratio and Λ is the critical slenderness ratio. Table 5.1 is a simplified form of the Japanese building standards [145]. In general, the long-term allowable stress design considers the self-weight of the structure and the live load, and the short-term allowable stress design considers the seismic and wind loads in addition to the above loads. The frame structure is designed against both long-term and short-term static loads in this chapter. More specifically, there are three load cases to be considered, as shown in Fig. 5.1; one is a long-term load case, and the others are short term load cases in which the horizontal seismic loads are applied in the right and left directions, respectively.

To calculate the stresses and the displacements of the members, the static nodal loads are computed in the following way. The long-term loads include

Table 5.1: Allowable stresses of steel material

	long-term allowable stress	short-term allowable stress
tension f_t	$F_d/1.5$	F_d
compression f_c	$\lambda \leq \Lambda$	$\frac{1-0.4(\lambda/\Lambda)^2}{3/2+(2/3)(\lambda/\Lambda)^2} F_d$
	$\lambda > \Lambda$	$\frac{18}{65(\lambda/\Lambda)^2} F_d$
bending f_b	$F_d/1.5$	F_d

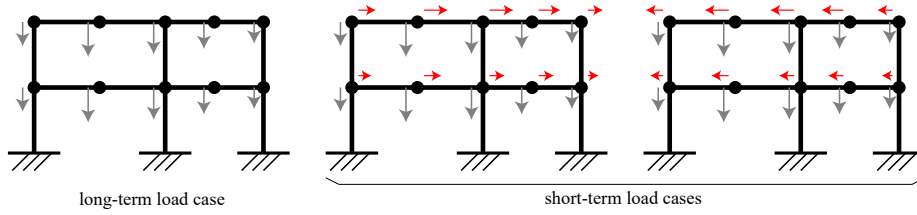


Figure 5.1: Three load cases to be considered for the frame design in this chapter

the self-weight of the structure that can be calculated by the product of the steel density 77 kN/m^3 and the total volume of the members, the finishes of the frame calculated as 1.0 kN/m^2 per unit floor area, and live loads calculated as 2.4 kN/m^2 per unit floor area. In addition to the long-term loads, the short-term loads shall receive seismic loads estimated from the A_i^s distribution for the self-weight, the fixed loads of the finishes, and live loads of 1.3 kN/m^2 per unit floor area. The load condition is given by converting them into the nodal loads weighted for each span length, as shown in Fig. 5.2.

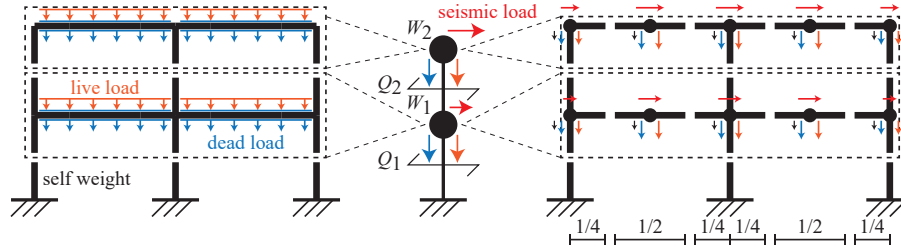


Figure 5.2: Process of converting distributed loads to nodal loads

The calculation procedure of the seismic loads of a frame with n_{sf} stories is explained in detail below. Let W_i and ψ_i denote the weight of the i -th story and the ratio of W_i to $W_T = \sum_{i=1}^{n_{sf}} W_i$, respectively. In the current Japanese seismic design code [145], the design shear force of the i -th story Q_i is expressed as

$$Q_i = C_i^s W_T \psi_i \quad (5.1)$$

where C_i^s is the shear force coefficient that determines the intensity and the vertical variation of the seismic loads and is calculated by the following equation:

$$C_i^s = Z R^s A_i^s C_B \quad (5.2)$$

Z is set within the range of 1.0 to 0.7 depending on the degree of seismic damage based on the records of past earthquakes in the region, the status of the current seismic activities, and the nature of the earthquakes. R^s represents the vibration characteristics of a building and is calculated with a value of 1 or less according to the natural period in the elastic region of the building and the ground type. For simplicity, Z and R^s are set to 1.0. C_B is the base shear coefficient, which is set to 0.2 in this study. A_i^s represents the distribution of the seismic shear force coefficient in the height direction of the building according to the vibration characteristics of the building, and can be roughly calculated by the following equation:

$$A_i^s = 1 + \left(\frac{1}{\sqrt{\psi_i}} - \psi_i \right) \frac{2T_f}{1 + 3T_f} \quad (5.3)$$

where T_f is the primary natural period for the building design, which is roughly obtained for a steel structure with a height of H as

$$T_f = 0.03H \quad (5.4)$$

The seismic load in the top story is equal to the shear force in the top story, and the seismic load in the i -th story is computed from $Q_i - Q_{i+1}$ as a difference between the shear forces of the i -th and $(i + 1)$ -th stories.

The frame is assumed to be linear elastic with Young's modulus E and the shear modulus G . The standard 6-degree-of-freedom 2D beam element is used. Let $\bar{\mathbf{p}}_i$ denote the member force vector. The local stiffness equation about member i is described with the local stiffness matrix \mathbf{K}_i , the local displacement vector $\bar{\mathbf{u}}_i$ and the local load vector applied to each DOF in the member $\bar{\mathbf{p}}_i$ as

$$\mathbf{K}_i \bar{\mathbf{u}}_i = \bar{\mathbf{p}}_i \quad (5.5)$$

Using a transformation matrix, elements of \mathbf{K}_i are converted to those in the global coordinate system to be assembled into the global stiffness matrix \mathbf{K} . Let \mathbf{u} and \mathbf{p} denote the displacement and load vectors about all DOFs in the global coordinate system. Then the stiffness equation after assigning boundary conditions is constructed as

$$\mathbf{K}\mathbf{u} = \mathbf{p} \quad (5.6)$$

Since \mathbf{p} is already specified, Eq. (5.6) is a set of simultaneous linear equations with respect to \mathbf{u} . When solving the stiff equation to calculate the displacement of the frame nodes, the axial stiffness of the beams is set to a value 10 times larger than the original value in order to simulate the rigid floor assumption. From the displacements \mathbf{u} , the inter-story drift with respect to the member length of the column \tilde{d}_i^c and the deflection at the intermediate node with respect to the member length of the beam \tilde{d}_i^b can be calculated. According to [146], in order to ensure safety and serviceability of the building, the upper-bounds for \tilde{d}_i^c and \tilde{d}_i^b are 1/200 and 1/300, respectively.

After solving Eq. (5.6), the axial force N_i and the bending moments $M_{i,1}$ and $M_{i,2}$ at the two ends of each member are also computed from $\bar{\mathbf{p}}_i$ ($i = 1, \dots, n_m$). Using the cross-sectional area A_i and the section modulus Z_i of the i -th member, the axial stress σ_i^a and the bending stresses $\sigma_{i,1}^b$ and $\sigma_{i,2}^b$ at the two ends are computed as

$$\sigma_i^a = \frac{N_i}{A_i} \quad (5.7a)$$

$$\sigma_{i,1}^b = \frac{M_{i,1}}{Z_i} \quad (5.7b)$$

$$\sigma_{i,2}^b = \frac{M_{i,2}}{Z_i} \quad (5.7c)$$

Note that σ_i^a is negative for compression and is positive for tension. Then, the stress ratio $\tilde{\sigma}_i$ is computed as the total of compression, tension, and bending stress ratios to the allowable stresses as

$$\tilde{\sigma}_i = \begin{cases} -\frac{\sigma_i^a}{f_c} + \frac{\max\{|\sigma_{i,1}^b|, |\sigma_{i,2}^b|\}}{f_b} & (\text{if } \sigma_i^a < 0) \\ \frac{\sigma_i^a}{f_t} + \frac{\max\{|\sigma_{i,1}^b|, |\sigma_{i,2}^b|\}}{f_b} & (\text{if } \sigma_i^a \geq 0) \end{cases} \quad (5.8)$$

$\tilde{\sigma}_i$ must be less than or equal to 1.0 to ensure the safety.

5.1.2 “Strong column-weak beam” design

To avoid the local story collapse without enough energy dissipation and ensure the ductility of the frame, it is important to design the frame in accordance with a “strong column-weak beam” principle. Figure 5.3 shows two typical collapse modes. If a frame is designed so that plastic hinges are formed at the beam ends rather than at the column ends, the structure is expected to form the whole collapse mode where hinges at the beams are dominant as shown in the left figure of Fig. 5.3. On the other hand, if the plastic moment capacity of the columns is smaller than that of the beams, there is a possibility of the story collapse as shown in the right figure of Fig. 5.3.

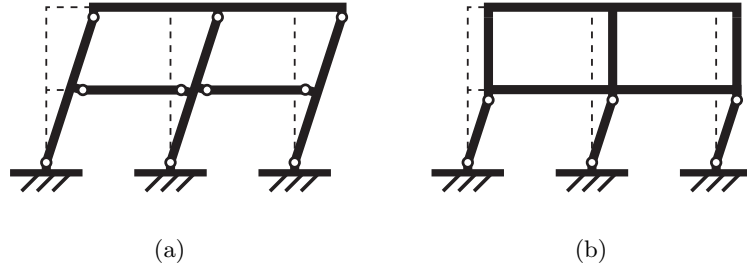


Figure 5.3: (a) Desirable collapse mechanism. (b) Undesirable collapse mechanism.

The column-to-beam strength ratio, or column overstrength factor (COF), is a strong column-weak beam criterion about a joint and calculated as

$$\beta = \frac{\sum M_{pc}}{\sum M_{pb}} \quad (5.9)$$

where $\sum M_{pc}$ and $\sum M_{pb}$ are the sums of the full plastic moments of the columns and beams connecting to the joint, respectively.

Nakashima and Sawaizumi [147] investigated the magnitude of the COF required for ensuring the occurrence of plastic hinges only at the ends of beams and the first-story column bases. It was found that the required COFs increase steadily with the increase of the ground motion velocity, and reach about 1.5 for the ground motion amplitude of 0.5 m/s, which is considered as the strong earthquake in the Japanese seismic design code [145]. Therefore, frames are designed to satisfy $\beta \geq 1.5$ in this study.

According to the Japanese building standards law, M_{pb} in Eq. (5.9) is given as

$$M_{pb} = F_{yb}Z_{pb} \quad (5.10)$$

where F_{yb} is the design strength and Z_{pb} is the plastic section modulus of a beam element. Although M_{pc} for a column is given in a similar way using the design strength F_{yc} and the plastic section modulus Z_{pc} , the value is discounted using the axial force ratio η_c , the ratio of the column's compressive axial stress to the yield stress, as

$$M_{pc} = \tau F_{yc}Z_{pc} \quad (5.11a)$$

$$\tau = \begin{cases} 1 - \frac{4\eta_c^2}{3} & (\eta_c \leq 0.5) \\ \frac{4(1-\eta_c)}{3} & (\eta_c > 0.5) \end{cases} \quad (5.11b)$$

Note that hinges are allowed to exist at the upper ends of the top story columns because it is too difficult for only one column to outperform the sum of the full plastic moments of two beams. Similarly, hinges are allowed to exist at the bottom ends of the first-story columns, because the bottom nodes are considered to be fixed in both translation and rotation, and the beams connecting them are not considered.

5.1.3 Optimization problem

Consider a discrete cross-section optimization problem to minimize the total structural volume under constraints on stress, displacement, and COF. A set of the cross-sections $\mathbf{J} = \{J_1 \cdots J_{n_m}\}$ is to be selected from Table 5.2 for the columns and from Table 5.3 for the beams. Then, the optimization problem is formulated as

$$\text{minimize } V_s(\mathbf{J}) \quad (5.12a)$$

$$\text{subject to } \begin{cases} \tilde{\sigma}_i \leq 1 & (i \in \{1, \dots, n_m\}) \\ \tilde{d}_i^b \leq 1/300 & (i \in \Omega_b) \\ \tilde{d}_i^c \leq 1/200 & (i \in \Omega_c) \\ \beta_k \leq 1.5 & (k \in \Omega_\beta) \end{cases} \quad (5.12b)$$

where Ω_b , Ω_c and Ω_β are the sets of indices of beam members, column members, and middle-layer nodes that does not include bottom and top ones, respectively.

Table 5.2: Dimension list of column sections



J_i	Dimension [mm] $H \times B \times t$		A_i [cm ²]	I_i [cm ⁴]	Z_i [cm ³]	$Z_{p,i}$ [cm ³]
200	200 × 200 × 12		85.3	4860	486	588
250	250 × 250 × 12		109.3	10100	805	959
300	300 × 300 × 16		173.0	22600	1510	1810
350	350 × 350 × 19		239.2	42400	2420	2910
400	400 × 400 × 22		307.7	69500	3480	4220
450	450 × 450 × 22		351.7	103000	4560	5490
500	500 × 500 × 25		442.8	159000	6360	7660
550	550 × 550 × 25		492.8	217000	7900	9460
600	600 × 600 × 25		542.8	288000	9620	11400
650	650 × 650 × 28		656.3	407000	12500	14900
700	700 × 700 × 28		712.3	518000	14800	17600
750	750 × 750 × 32		866.3	717000	19100	22800
800	800 × 800 × 32		930.3	884000	22100	26200
850	850 × 850 × 32		994.3	1070000	25300	29900
900	900 × 900 × 36		1177.0	1420000	31500	37300
950	950 × 950 × 36		1249.0	1680000	35500	42000
1000	1000 × 1000 × 36		1321.0	1990000	39700	46900

Table 5.3: Dimension list of beam sections

J_i	Dimension [mm] $H \times B \times t_1 \times t_2$		A_i [cm ²]	I_i^z [cm ⁴]	I_i^y [cm ⁴]	Z_i^z [cm ³]	Z_i^y [cm ³]	$Z_{p,i}$ [cm ³]
200	194 × 150 × 6 × 9		38.11	2630	507	271	67.6	301
250	244 × 175 × 7 × 11		55.49	6040	984	495	112	550
300	294 × 200 × 8 × 12		71.05	11100	1600	756	160	842
350	340 × 250 × 8 × 14		99.53	21200	3650	1250	292	1380
400	400 × 200 × 9 × 19		110.0	31600	2540	1580	254	1770
450	450 × 200 × 9 × 22		126.0	45900	2940	2040	294	2750
500	500 × 250 × 9 × 22		152.5	70700	5730	2830	459	3130
550	550 × 250 × 9 × 22		157.0	87300	5730	3180	459	3520
600	600 × 250 × 12 × 25		192.5	121000	6520	4040	522	4540
650	650 × 250 × 12 × 25		198.5	145000	6520	4460	522	5030
700	700 × 250 × 12 × 25		205.8	173000	6520	4940	522	5580
750	750 × 300 × 14 × 28		267.9	261000	12600	6970	841	7850
800	800 × 300 × 14 × 28		274.9	302000	12600	7560	841	8520
850	850 × 300 × 16 × 28		297.8	355000	12600	8350	842	9540
900	900 × 300 × 16 × 28		305.8	404000	12600	8990	842	10300
950	950 × 300 × 16 × 28		313.8	458000	12600	9640	842	11100
1000	1000 × 300 × 16 × 28		321.8	515000	12600	10300	842	11900

5.2 Conversion to a reinforcement learning task

In this section, the optimization problem (5.12) is transformed into a RL task that enables learning by the proposed method.

The input of each node $\mathbf{v}_k \in \mathbb{R}^4$ is shown in the Table 5.4. In order to consider the support positions of the frame, an input that is 1 at the support and 0 otherwise is defined in the first component. In addition, the second and third components are defined for identifying the nodes at the top where the COF constraints are not necessary, and the nodes at the side ends where the COF values differ greatly due to the difference in the number of beams to be connected, respectively. The COF is considered in the fourth component of the node input, and the value is clipped not to exceed 1.0 by multiplying the reciprocal of the COF value by its lower bound 1.5.

Similarly, the input of each member $\mathbf{w}_i \in \mathbb{R}^6$ is shown in Table 5.5. Inputs that take a binary value of 1 or 0 depending on the member type (column or beam) are defined separately for indices 1 and 2. According to Silver et al. [25], when inputting the board of Go to a CNN, instead of expressing the position of black stones as +1 and the position of white stones as -1 with the same input, the positions of black and white stones are expressed separately; one input expresses the position of black stones as +1 and the others to be 0; the other input expresses the position of white stones as +1 and the others to be 0. If the positions of black and white stones are expressed with the same input, this input is weighted with the same parameters in the NN. On the other hand, if the inputs of black and white stones are independently defined, the NN can separately consider the positions of black and white stones and weight them with different parameters, which is a more rational input setting. This is the reason why separate inputs are defined for columns and beams. The member input is constituted of member type, the member length, and the numbers representing the size of the cross-section, stress factor, and displacement factor. Note that the inputs are scaled and clipped so as to values within the range $[0, 1]$.

Table 5.4: Detail of node input \mathbf{v}_k

index	input
1	1 if fixed, 0 otherwise
2	1 if at the top, 0 otherwise
3	1 if at the side ends, 0 otherwise
4	$\min \{1.5/\beta_k, 1.0\}$

Table 5.5: Detail of member input \mathbf{w}_i

index	input
1	1 if column, 0 otherwise
2	1 if beam, 0 otherwise
3	$L_i/12.0$
4	$J_i/1000$
5	$\min \{\tilde{\sigma}_i, 1.0\}$
6	$\min \{\tilde{d}_i, 1.0\}$

Here, different agents are trained for increasing and reducing the member cross-sections, respectively. That is, the first agent changes the cross-section of a member one step larger by the action, and the second agent changes the cross-section of a member one step smaller by the action. In the same manner as Chapter 4, any arbitrary state-action pair deterministically leads to the unique next state. That is, when the action of increasing or reducing the cross-section of member i is selected, the cross-section of the members other than the member i never changes. If the lower columns on the same axis become thinner than the upper column as a result of the action, the cross-section of the lower columns is corrected so that it becomes equal to the upper column.

The reward is obtained after each design change of the member cross-sections. In this study, two types of reward definition are considered based on the following concept from the viewpoint of structural engineers. Regardless of whether the member cross-sections are reduced or increased, it is necessary to consider the stress and displacement of each member and the COFs at both ends, and keep them within the constraint. Assuming that the information of all the other members is not visible and the constraint function values about only one member are given, it is possible to consider to some extent whether the cross-section of the member can be reduced or increased. This is a judgment based on the local information of each member. On the other hand, it is also necessary to compare all the members and consider how much better it is to reduce or increase the cross-section of a member compared to the others. This is a judgment based on the global information about the entire structure. This way, it is necessary to design rewards that take into account both local and global information.

When reducing the member cross-sections, the process starts with a redundant design that satisfies all the constraints, and a negative reward or penalty is assigned for the termination state in which the structural design exceeds the constraints. If a sufficiently large cross-section is reduced, the reward closer to +1.0 is assigned; if the cross-section that is likely to exceed the constraint is reduced, the reward closer to 0 is assigned.

If a positive reward is given to increase the member cross-sections, the agent may learn a *detour* strategy in an attempt to make the steps as long as possible. Since it is preferable to reach a feasible cross-section design that satisfies the constraint in fewer steps, a negative reward is given to the cross-section that does not satisfy the constraint, and a positive reward is assigned only to the terminal state that satisfies the constraint. A negative reward closer to 0 is given when the bad cross-section that deviates from the constraint most is increased through comparison among all the members, and a negative reward closer to -1 is given when a cross-section large enough to meet the local constraints at the member is increased.

Reward setting 1

In this setting, the common reward function is defined for both tasks of increasing and reducing the member cross-sections.

$$r_1 = \frac{1}{3} \left(G_1 \left(\frac{\max_i \tilde{\sigma}'_i}{\max_i \tilde{\sigma}_i} \right) + G_1 \left(\frac{\min_i \beta_i}{\min_i \beta'_i} \right) + G_1 \left(\frac{\max_i \tilde{d}'_i}{\max_i \tilde{d}_i} \right) \right) \quad (5.13a)$$

$$G_1(x) = \begin{cases} \min \{x, 1.0\} & \text{(if feasible)} \\ 0 & \text{(else if } x \text{ satisfies its constraint)} \\ \frac{n_e}{\sqrt{n_{sf}}} \max \left\{ -\frac{1}{x}, -1.0 \right\} & \text{(else)} \end{cases} \quad (5.13b)$$

where n_e is the number of members whose cross-section has been changed by the action at the step and n_{sf} is the number of stories of the frame. $G_1(x)$ is the clipping and scaling function to adjust the magnitude of reward values. Here, the term *feasible* means that the solution satisfies all the constraints of the optimization problem.

Reward setting 2

In this setting, different reward functions are defined for the tasks of increasing and reducing the member cross-sections. For the task of increasing the cross-sections, the reward function is defined as

$$r_2^+ = \begin{cases} 1.0 & \text{(if feasible)} \\ \frac{n_e}{3\sqrt{n_{sf}}} \left(G_2^+ \left(-\frac{\max_i \tilde{\sigma}'_i}{\max_i \tilde{\sigma}_i} \right) + G_2^+ \left(-\frac{\min_i \beta'_i}{\min_i \beta_i} \right) + G_2^+ \left(-\frac{\max_i \tilde{d}'_i}{\max_i \tilde{d}_i} \right) \right) & \text{(else)} \end{cases} \quad (5.14a)$$

$$G_2^+(x) = \max \{x, -1.0\} \quad (5.14b)$$

The reward function for reducing the cross-section of member a is defined as

$$r_2^- = \begin{cases} (1.0 - \tilde{\sigma}_a) + \left(1.0 - \frac{1.5}{\min_{j \in \{1,2\}} \beta_{a,j}} \right) + (1.0 - \tilde{d}_a) & \text{(if feasible)} \\ -1.0 & \text{(else)} \end{cases} \quad (5.15)$$

where $\beta_{a,j}$ is the COF of the j -th tip node of the selected member a .

5.3 Training setting

The detail of the training implementation is explained in this section. The agents are trained using a 5-layer, 3-span planar frame structure as shown in Fig. 5.4. The elastic modulus of the members is set to be 2.05×10^5 N/mm². The I-beams are arranged so that the strong axis becomes on the same plane of the planar frame. The bottom nodes are rigidly supported against both translation and rotation.

At the beginning of each episode, the frame geometry is randomized while fixing its connectivity. Each span is randomly initialized in the range of 5-12 m. The floor heights of the 2nd to 5th floors should be equal, and they are randomly initialized in the range of 3.5 to 4.5m. The floor height of the first floor is set 0.5 m higher than the other floor heights.

The initial cross-sections depend on the task to be trained. All the initial cross-sections are set to the maximum of $J_i = 1000$ ($i = 1, \dots, n_m$) for the task of reducing the member cross-sections and the minimum of $J_i = 200$ ($i = 1, \dots, n_m$) for the task of increasing the member cross-sections.

Once the shape and cross-sections are determined, the loads are calculated according to the procedure explained in Sec. 5.1.1. Since the live loads and the seismic loads are given in proportion to the floor area, it is necessary to set the out-of-plane span in a pseudo manner. Here, 0.75 times the sum of the in-plane spans is assumed to be the out-of-plane span for which the planar frame bears the load. At this phase, the seismic load with the same magnitude is applied leftward and rightward, acting as two separate load cases.

In the task of increasing the member cross-sections, the terminal state is a cross-section design that satisfies all the constraints, and in the task of reducing the member cross-sections, the terminal state is a cross-section design that exceeds any of the constraints. The cross-section is changed and the long-term and short-term loads are also updated at each step until the terminal state. The training episode is stopped when the terminal state is reached, and a new episode starts with the frame whose geometry is re-randomized.

In the same manner as Chapter 4, the number of episodes n_{ep} is set to be 5000. The size of member feature is set as $n_f = 100$. The training is implemented with different combinations of learning rate α and discount rate γ . In addition to Intel(R) Core(TM) i9-7900X CPU @ 3.30GHz, a graphics processing unit (GPU) GeForce(R) GTX 1080 Ti is further introduced to accelerate the training. See Appendix A for more details about speeding up the training with GPUs.

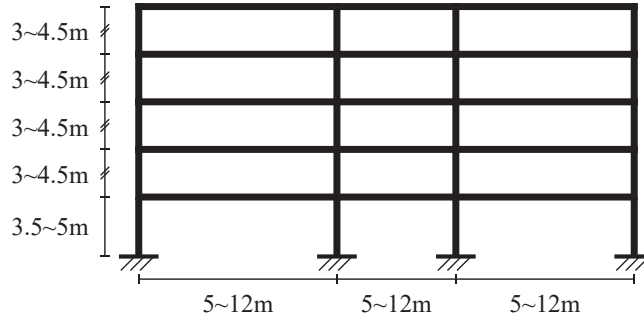


Figure 5.4: Steel frame used for training

5.4 Training result

5.4.1 Training history and performance for 3×5 -grid frame

Reward setting 1

First, the result trained for reducing the size of the members is explained. Figure 5.6 shows the histories of obtained cumulative reward in the test simulations recorded at every 10 episodes for reward setting 1. The training is implemented for different learning rates $\alpha = 5.0 \times 10^{-5}$ and $\alpha = 1.0 \times 10^{-5}$. For the task of

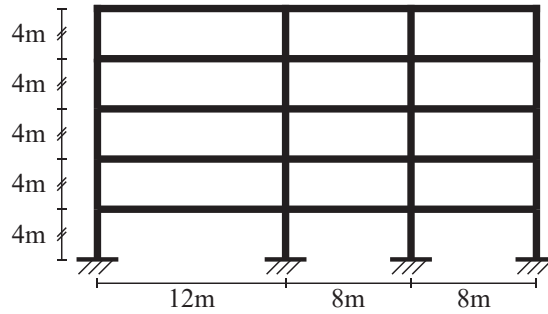


Figure 5.5: Steel frame used for test

reducing the cross-sections with reward setting 1, $\alpha = 5.0 \times 10^{-5}$ is clearly not a good hyperparameter, because the training history greatly fluctuates.

The cumulative reward obtained by the agent for $\alpha = 1.0 \times 10^{-5}$ have increased as the number of training episodes increases. It took about 20.7 hours for the training through approximately 1.40 million linear structural analyses with Intel(R) Core(TM) i9-7900X CPU @ 3.30GHz and updates of trainable parameters with GeForce(R) GTX 1080 Ti. Meanwhile, it took 47.1 hours to implement the training by the CPU only. It turned out that the computational cost required for updating the trainable parameters occupies a large proportion of the total training process, and that the effect of reducing the computation time by the GPU is very large.

The sequence of design changes of the cross-sections in the test simulation when the maximum cumulative reward is obtained for $\alpha = 1.0 \times 10^{-5}$ is illustrated in Fig. 5.7. The cross-sections of all members are changed in a well-balanced manner without concentrating on the cross-section of a specific member. The stress ratio of the beam whose cross-section was reduced in the final step exceeded 1.0, as illustrated in Fig. 5.9. Therefore, the 293rd step which is just before the terminal state is regarded as the sub-optimal solution, as illustrated in Fig. 5.8.

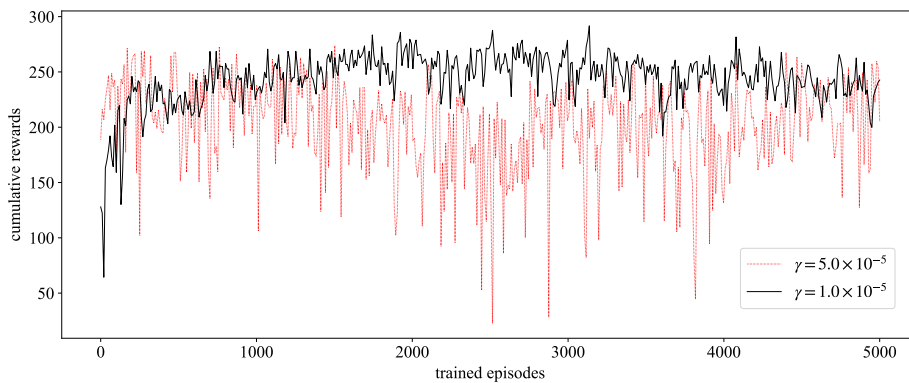


Figure 5.6: Histories of the cumulative reward of each test measured every 10 episodes (reducing cross-sections, reward setting 1)

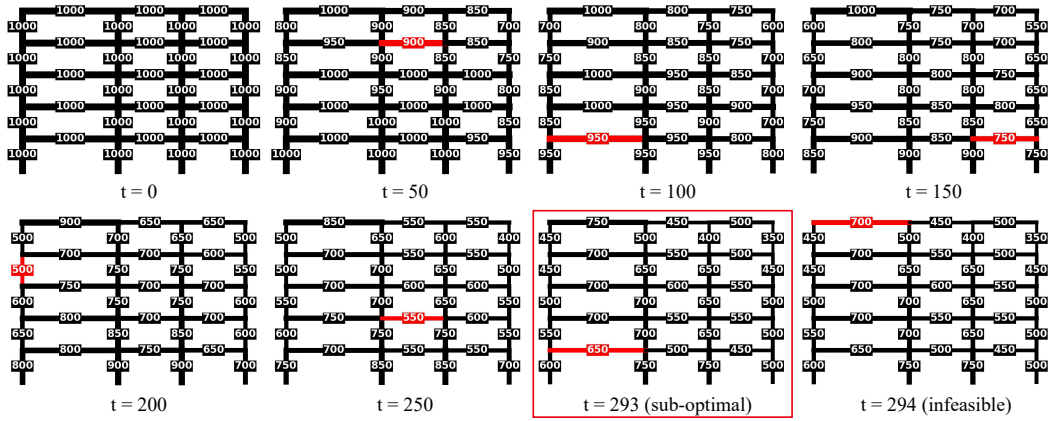


Figure 5.7: Decremental sequence of cross-sections (reward setting 1)

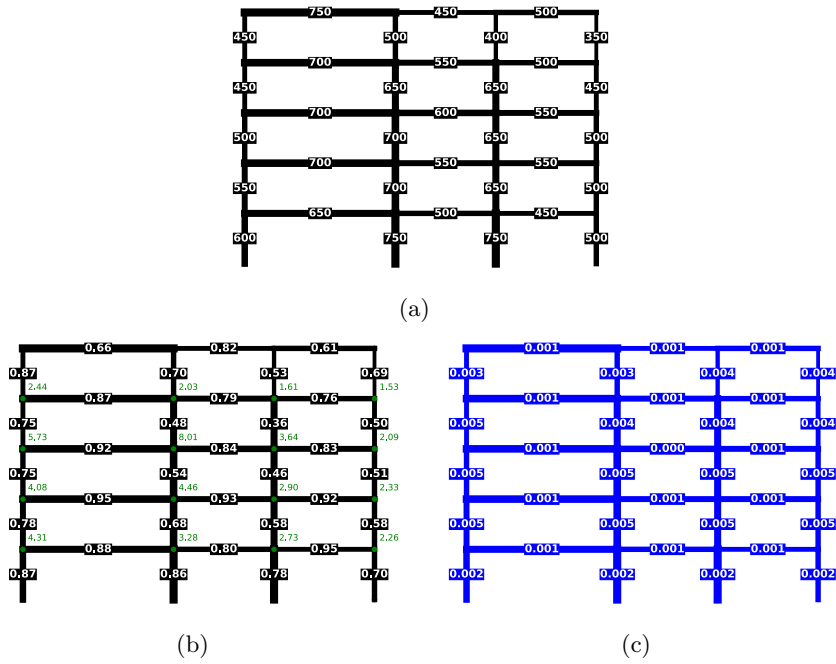
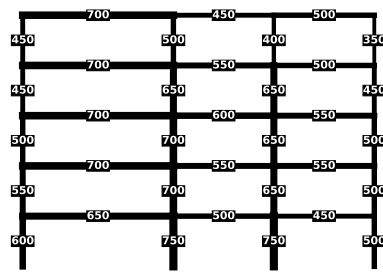
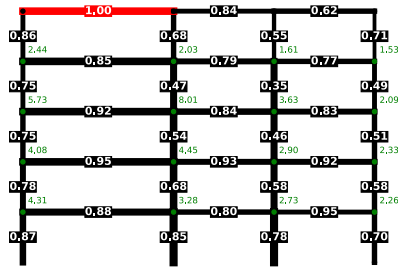


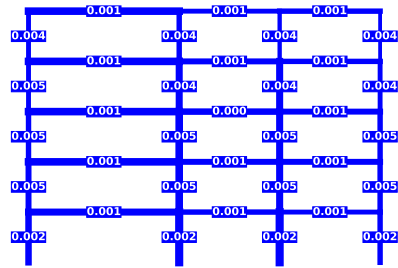
Figure 5.8: The result at step 293 (reducing cross-sections, reward setting 1, feasible solution). (a) Cross-section index J_i ($V_s = 6.778[m^3]$). (b) The maximum stress ratio in the member and the COF at the node. (c) The inter-story drift ratio of the column and the deformation at the mid-point of the beam over its length.



(a)



(b)



(c)

Figure 5.9: The result at step 294 (reducing cross-sections, reward setting 1, infeasible solution). (a) Cross-section index J_i ($V_s = 6.704[\text{m}^3]$). (b) The maximum stress ratio in the member and the COF at the node. The beam highlighted in red exceeds the stress constraint. (c) The inter-story drift ratio of the column and the deformation at the mid-point of the beam over its length.

Next, the result trained for increasing the size of the members is explained. Figure 5.10 shows the histories of the cumulative reward in the test simulations recorded at every 10 episodes for reward setting 1. Similarly to the task of reducing the cross-sections, the training is implemented for different learning rates $\alpha = 5.0 \times 10^{-5}$ and $\alpha = 1.0 \times 10^{-5}$. With both learning rates, the performance of the RL agent can be improved steadily as the number of training episodes increases. The case of $\alpha = 1.0 \times 10^{-5}$ is focused here in order to match the hyperparameters when the member cross-section is reduced.

It took about 19.9 hours for the training with $\alpha = 1.0 \times 10^{-5}$ through approximately 1.36 million linear structural analyses with Intel(R) Core(TM) i9-7900X CPU @ 3.30GHz, while taking 44.4 hours with the CPU only. The RL agent at the time of having learned 4500 episodes with the highest cumulative reward is considered to be the best performing agent. The agent acquires a reasonable strategy to intensively increase the size of columns and beams in the lower layer in the early stage of the test simulation, as seen in Fig. 5.11.

Because the member sizes are sequentially increased from the infeasible solution to the feasible solution in this simulation, the cross-sectional design of the 261st step, which is the terminal state, becomes the sub-optimal solution. The cross-section of the beam on the left side of the upper end is designed to be slightly excessive, and the total structural volume $V_s = 7.000[\text{m}^3]$ is inferior to the case of reducing the cross-sections in Fig. 5.8 $V_s = 6.778[\text{m}^3]$ as a result, but the cross-sections are designed with a good balance of stresses, displacements, and COFs.

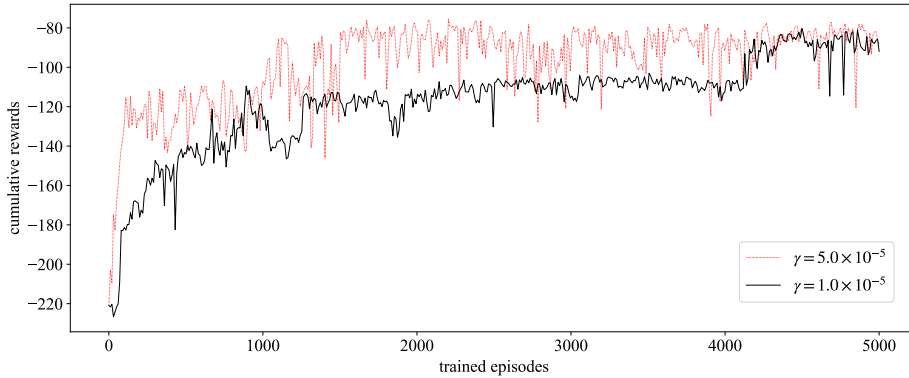


Figure 5.10: Histories of the cumulative reward of each test measured every 10 episodes (increasing cross-sections, reward setting 1)

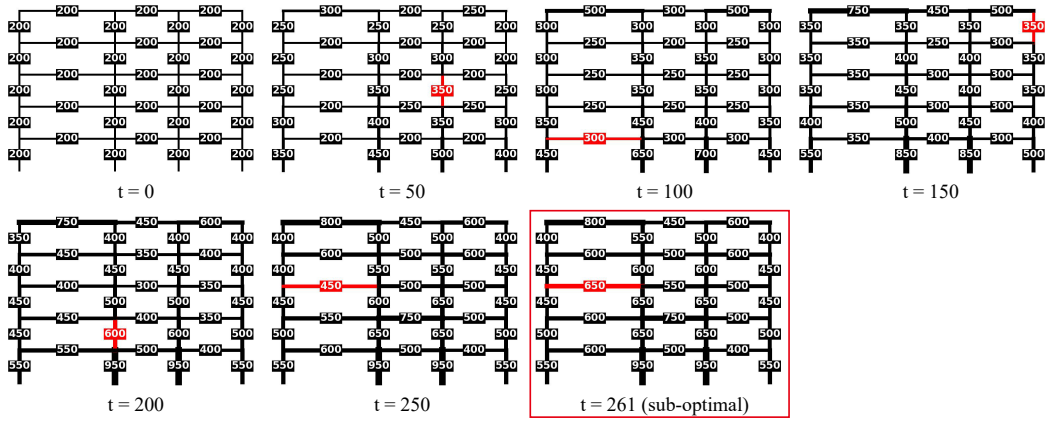


Figure 5.11: Incremental sequence of cross-sections (reward setting 1)

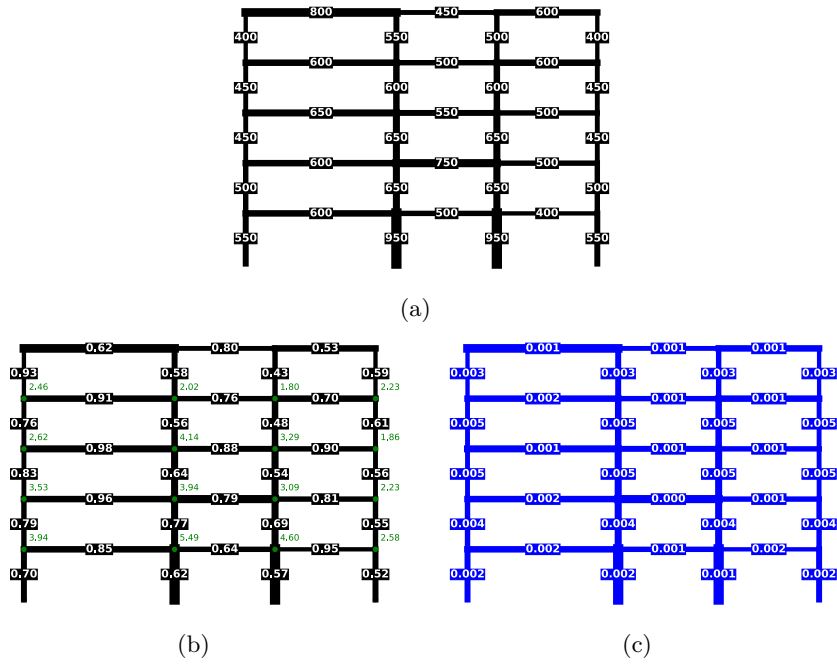


Figure 5.12: The result at step 261 (increasing cross-sections, reward setting 1, feasible solution). (a) Cross-section index J_i ($V_s = 7.000[\text{m}^3]$). (b) The maximum stress ratio in the member and the COF at the node. (c) The inter-story drift ratio of the column and the deformation at the mid-point of the beam over its length.

Reward setting 2

The training result for reward setting 2 is given in the following. First, the training histories for reducing the cross-sections with $\alpha = 5.0 \times 10^{-5}$ and $\alpha = 1.0 \times 10^{-5}$ are provided in Fig. 5.13. There is no significant difference in the history between the two learning rates. In order to match the parameter setting that gives better training results in the task of increasing cross-sections later explained in Fig. 5.17, the result of the training with the learning rate $\alpha = 5.0 \times 10^{-5}$ will be described in detail below.

The best test simulation at the 490th episode is illustrated in Fig. 5.14. In the early phase of the episode, the cross-sections of upper beams are intensively reduced to avoid exceeding the COF constraints. After that, the cross-sections of the columns and beams of the entire frame are reduced in a well-balanced manner, and finally, the sub-optimal cross-section design shown in Fig. 5.15 is obtained at the 290th step.

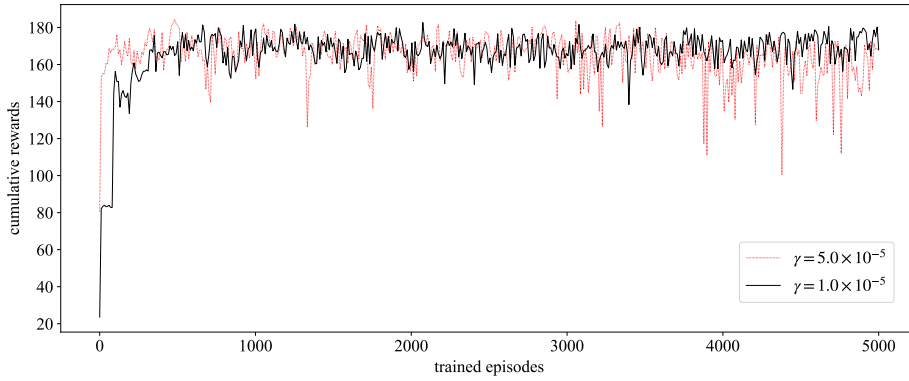


Figure 5.13: Histories of the cumulative reward of each test measured every 10 episodes (reducing cross-sections, reward setting 2)

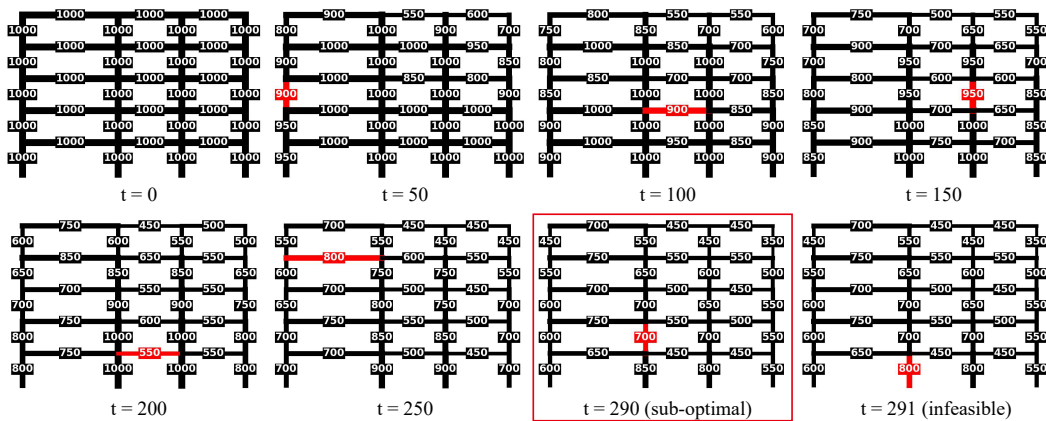
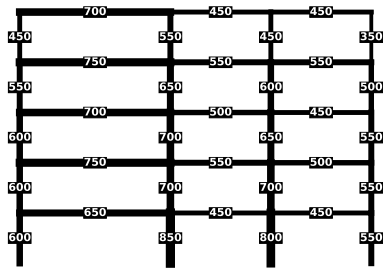
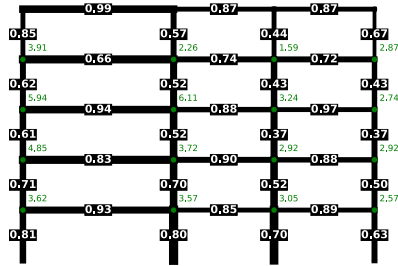


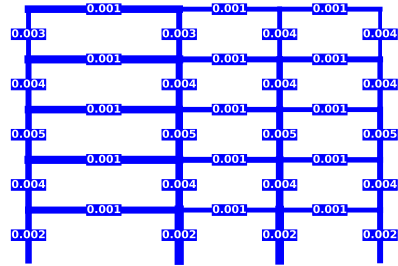
Figure 5.14: Decremental sequence of cross-sections (reward setting 2)



(a)



(b)



(c)

Figure 5.15: The result at step 290 (reducing cross-sections, reward setting 2, feasible solution). (a) Cross-section index J_i ($V_s = 7.058[\text{m}^3]$). (b) The maximum stress ratio in the member and the COF at the node. (c) The inter-story drift ratio of the column and the deformation at the mid-point of the beam over its length.

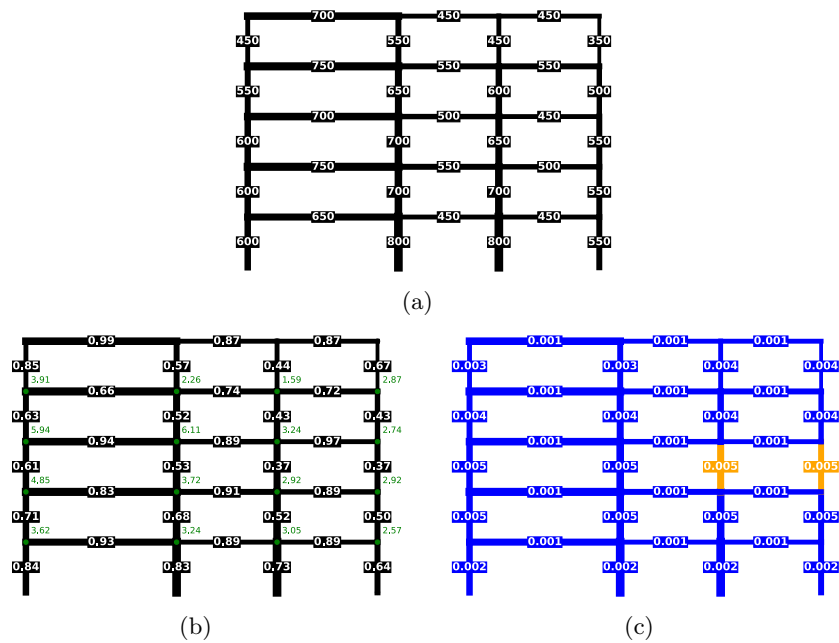


Figure 5.16: The result at step 291 (reducing cross-sections, reward setting 2, infeasible solution). (a) Cross-section index J_i ($V_s = 7.032[\text{m}^3]$). (b) The maximum stress ratio in the member and the COF at the node. (c) The inter-story drift ratio of the column and the deformation at the mid-point of the beam over its length. The column highlighted in yellow exceeds the displacement constraint.

Next, the result of increasing the cross-sections with reward setting 2 is provided. The training histories with learning rates $\alpha = 5.0 \times 10^{-5}$ and $\alpha = 1.0 \times 10^{-5}$ are shown in Fig. 5.17. Although the agent performance can be stably improved by using either learning rate, it has become possible to learn a larger cumulative reward when the learning rate is $\alpha = 5.0 \times 10^{-5}$ than when $\alpha = 1.0 \times 10^{-5}$. The elapsed CPU time for the training with $\alpha = 5.0 \times 10^{-5}$ is 18.7 hours using the CPU and GPU, while it takes 43.4 hours when using the CPU only.

Figure 5.18 shows the sequence of design changes at the time of 4650-episode training, in which the maximum cumulative reward was obtained when the training was performed at a learning rate of $\alpha = 5.0 \times 10^{-5}$. The bottom columns have intensively been enlarged, and then the columns and beams of the entire frame have been changed in a well-balanced manner.

The resulting feasible cross-section design is shown in Fig. 5.19. The total structural volume of the sub-optimal solution obtained by this simulation is the smallest among the various training results shown so far. On the other hand, the cross-section of the central beam that supports the 3rd floor is over-designed at 800, and there is still room for improvement toward a more slender cross-section design.

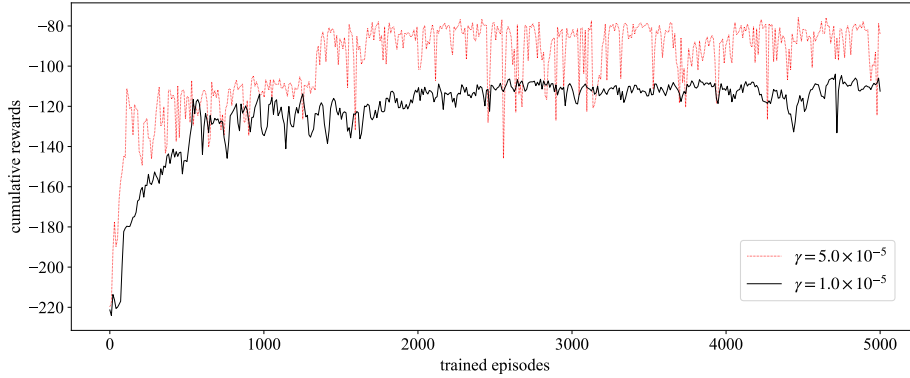


Figure 5.17: Histories of the cumulative reward of each test measured every 10 episodes (increasing cross-sections, reward setting 2)

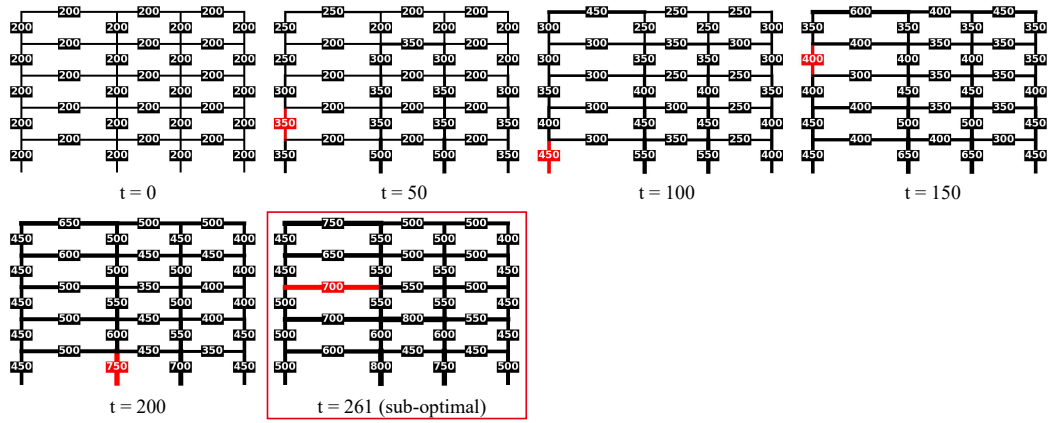


Figure 5.18: Incremental sequence of cross-sections (reward setting 2)

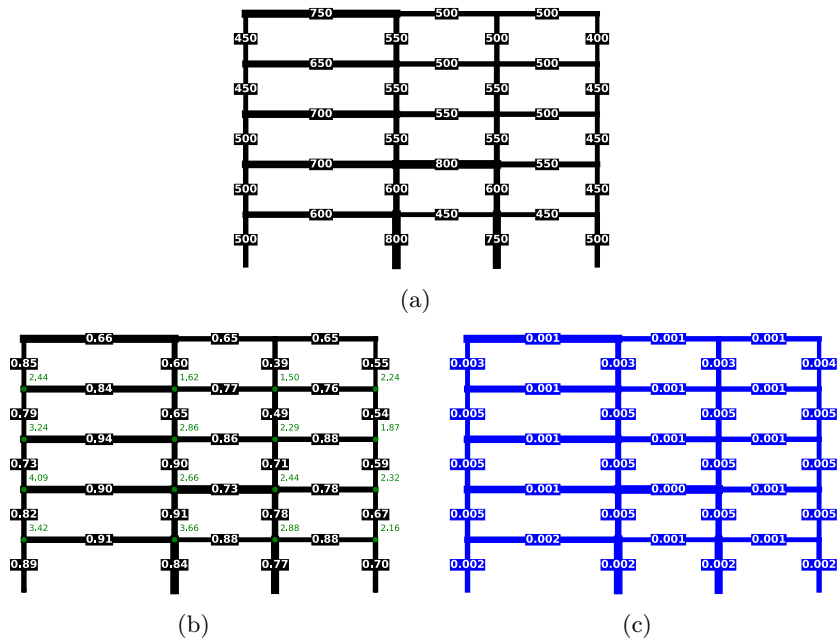


Figure 5.19: The result at step 247 (increasing cross-sections, reward setting 2, feasible solution). (a) Cross-section index J_i ($V_s = 6.418[\text{m}^3]$). (b) The maximum stress ratio in the member and the COF at the node. (c) The inter-story drift ratio of the column and the deformation at the mid-point of the beam over its length.

5.4.2 Investigation of generalization performance 1: 5×3-grid frame

The trained RL agents in Sec. 5.4.1 are applied to another frame with 5 spans and 3 stories, as shown in Fig. 5.20. The second span from the left is 12m, and the remaining spans are all 8m and the floor height is 4m for each story.

The cross-section design change to this frame is implemented using the following trained agents which are already trained in the previous section.

Agent(-) reducing size, reward setting 1, $\alpha = 1.0 \times 10^{-5}$, Figs. 5.6 - 5.9

Agent(+) increasing size, reward setting 2, $\alpha = 5.0 \times 10^{-5}$, Figs. 5.17 - 5.19

First, the result of reducing the cross-sections using Agent(-) is explained. All cross-sections start from $J_i = 1000$, and the cross-section design becomes infeasible at the 203rd step. Therefore, the cross-sections at 202nd step is regarded as the sub-optimal cross-section design as shown in Fig. 5.21. There is relatively little variation in the resulting cross-sections; the chosen cross-sections are between $N_i = 450$ and $N_i = 850$ for columns and $N_i = 550$ and $N_i = 850$ for beams.

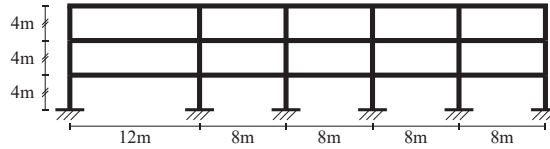


Figure 5.20: 5 × 3-grid steel frame

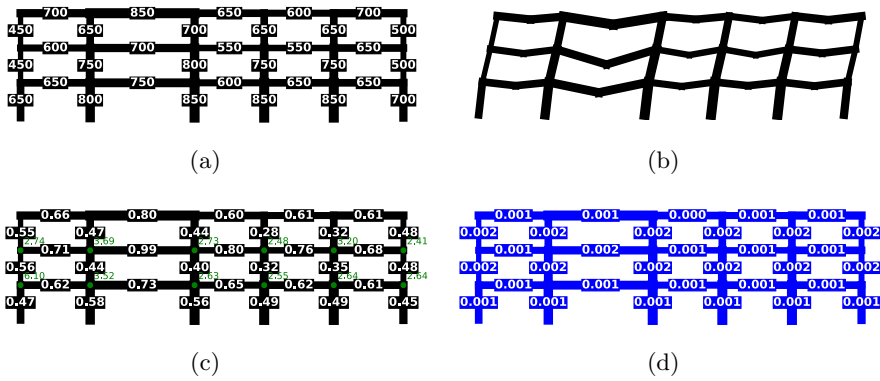


Figure 5.21: The result at step 202 (using Agent(-)). (a) Cross-section index J_i ($V_s = 7.995[\text{m}^3]$). (b) Deformation against short-term load (scaled by $\times 100$) (c) The maximum stress ratio in the member and the COF at the node. (d) The inter-story drift ratio of the column and the deformation at the mid-point of the beam over its length.

Next, Agent(+) is applied to the same 5×3 -grid frame. The cross-sections start from the minimum $J_i = 200$. The cross-sections are sequentially increased by the agent and the feasible solution is obtained at step 288, as shown in Fig. 5.22. Although the cross-section of the bottom-layer beams is designed to be small, this result is reasonable considering the boundary conditions and displacements; the supported nodes are strictly not allowed to be displaced, so that the inter-story drift of the columns in the bottom layer becomes small as shown in Fig. 5.22(b), and the stress of the beams in the bottom layer also becomes small.

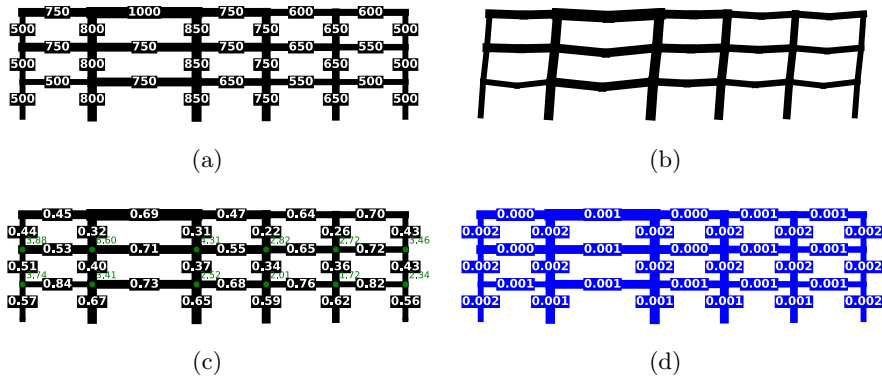


Figure 5.22: The result at step 288 (using Agent(+)). (a) Cross-section index J_i ($V_s = 8.207[\text{m}^3]$). (b) Deformation against short-term load (scaled by $\times 100$) (c) The maximum stress ratio in the member and the COF at the node. (d) The inter-story drift ratio of the column and the deformation at the mid-point of the beam over its length.

5.4.3 Investigation of generalization performance 2: 4×6-grid frame

Next, Agent(-) and Agent(+) are applied to a larger scale frame as shown in Fig. 5.23. The frame has 4 spans with lengths of 10m, 10m, 8m, and 8m from the left. The uniform floor height of 4m is assigned to each floor.

The cross-sections, deformed shape, stress ratios, COFs and displacements of the sub-optimal solution obtained by Agent(-) are shown in Fig. 5.24. 10m-span beams have a larger cross-section than 8m-span beams, and both columns and beams are designed in a well-balanced manner. The total structural volume is $V_s = 12.940[\text{m}^3]$.

Similarly, the sub-optimal cross-sectional design obtained by Agent(+) is shown in Fig. 5.25. The total structural volume is $V_s = 13.554[\text{m}^3]$ which is larger than the solution obtained by Agent(-), due to the over-design of beams with $J_i = 1000$ and upper columns. On the other hand, the deformation of the entire structure is kept small due to the high rigidity, as shown in Figs. 5.24(b) and 5.24(d).

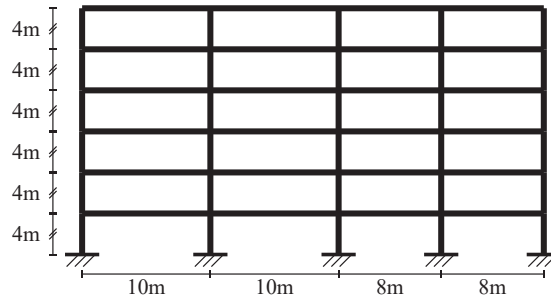


Figure 5.23: 5 × 3-grid steel frame

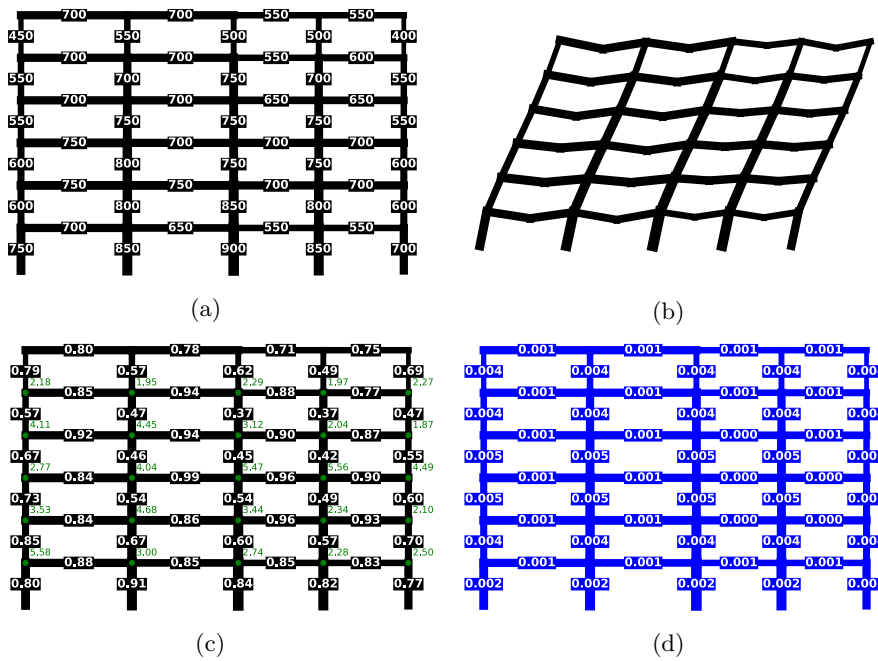


Figure 5.24: The result at step 322 (using Agent(-)). (a) Cross-section index J_i ($V_s = 12.940[m^3]$). (b) Deformation against short-term load (scaled by $\times 100$). (c) The maximum stress ratio in the member and the COF at the node. (d) The inter-story drift ratio of the column and the deformation at the mid-point of the beam over its length.

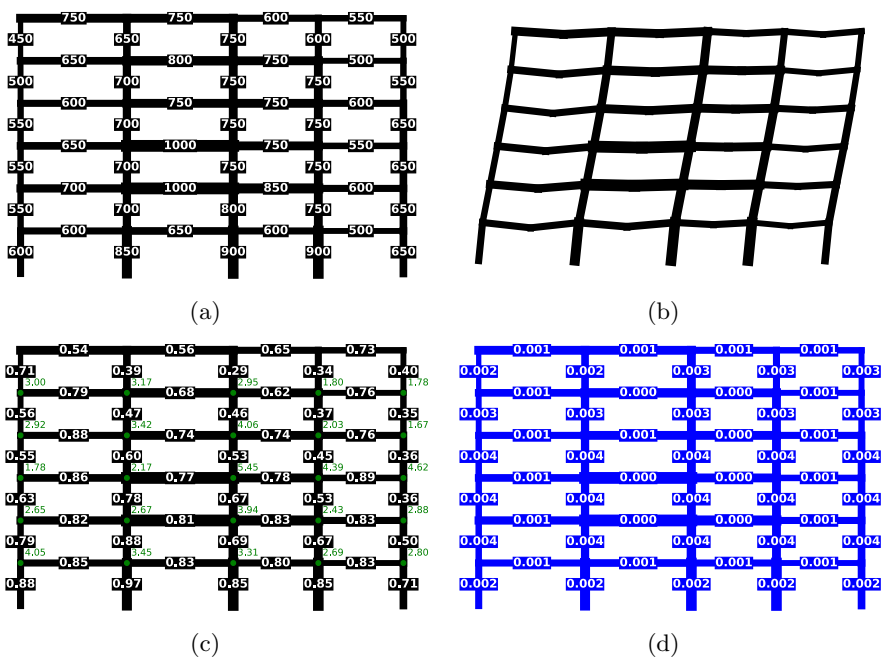


Figure 5.25: The result at step 499 (using Agent(+)). (a) Cross-section index J_i ($V_s = 13.554[m^3]$). (b) Deformation against short-term load (scaled by $\times 100$) (c) The maximum stress ratio in the member and the COF at the node. (d) The inter-story drift ratio of the column and the deformation at the mid-point of the beam over its length.

5.4.4 Comparison of efficiency and accuracy with particle swarm optimization

Finally, the performance of Agent(+) and Agent(-) are compared with PSO in view of the computational cost and the quality of the solution. Like GAs, PSO is also one of the most popular metaheuristics. PSO is first developed to simulate the movement of a bird flock and a fish school [148]. PSO algorithms utilize a population of candidate solutions. Each particle's position is repeatedly updated based on its position, velocity, and the best-known positions of the particle and the entire population.

The PSO algorithm used in this study is described in Table 5.6. The momentum factor is fixed to $\xi = 0.7$ and the social coefficient c_1 and the cognitive coefficient c_2 are both set to be 2.0. The maximum number of iterations and the number of particles are set to be 5000 and 10, respectively. The lower and upper bounds are set to be $x_i^L = 0.0$ and $x_i^U = 1.0$ ($i = 1, \dots, n_m$). In PSO, the variables take continuous values. The continuous variables are converted to cross-section index J_i using the following equation.

$$J_i = 200 + 50 \times \text{TR} (17x_{j,i} - 1.0 \times 10^{-10}) \quad (5.16)$$

where TR is a truncation operator. 17 is the number of possible cross-sections $J_i \in \{200, 250, \dots, 1000\}$. In Eq. (5.16), a very small value 1.0×10^{-10} is used to avoid the non-existing index $J_i = 1050$ when $x_{j,k} = 1.0$. Note that the negative value within $(-1.0, 0.0)$ is truncated to 0; this means that $J_i = 200$ when $x_{j,k} = 0.0$. To ensure the fairness of the problem definition, if the cross-section of a column is smaller than that of upper columns in the same axis, the cross-section is modified so as to become equal to the maximum cross-section among the upper columns. For example, if the bottom column's cross-section is $J_i = 550$ and the cross-section of the upper columns in the same axis is $J_i = 700$ or $J_i = 750$, the bottom column's cross-section is modified to $J_i = 750$. The initial variables are all set to be $\mathbf{x}_j, i = 0.5$ ($i \in \{1, \dots, n_m\}$) for all the particles $j \in \{1, \dots, n_p\}$. The initial velocity of each variable is provided from a uniform distribution of $[-0.05, 0.05]$.

The cost function to be minimized by the PSO algorithm is defined as

$$F(\mathbf{J}) = V_s(\mathbf{J}) + b_1 C_1(\mathbf{J}) + b_2 C_2(\mathbf{J}) + b_3 C_3(\mathbf{J}) \quad (5.17a)$$

$$C_1(\mathbf{J}) = \max \left\{ \max_{i \in \{1, \dots, n_m\}} (\tilde{\sigma}_i - 1), 0 \right\} \quad (5.17b)$$

$$C_2(\mathbf{J}) = \max \left\{ \max_{i \in \{1, \dots, n_m\}} (\tilde{d}_i - 1), 0 \right\} \quad (5.17c)$$

$$C_3(\mathbf{J}) = \max \left\{ \max_{k \in \Omega_\beta} (1.5 - \beta_k), 0 \right\} \quad (5.17d)$$

where b_1 , b_2 and b_3 are the penalty coefficients for the stress, displacement and COF constraints, respectively. b_1 , b_2 and b_3 are set to be 1000 in this study. This algorithm is terminated if the least cost function value $F(\mathbf{x}_b)$ is not updated for $n_t = 10$ consecutive cycles. To consider the variation of the solutions due to randomness, the PSO algorithm is executed 10 times, and their median value is extracted.

Table 5.6: PSO method used in this study

input: F : cost function, \mathbf{x}_j : initial solution of particle j , \mathbf{v}_j : initial velocity of particle j , \mathbf{x}^U : upper bounds, \mathbf{x}^L : lower bounds, $v^U(= 0.1)$ maximum velocity, $n_p(= 50)$: number of particles, $n_t(= 10)$: stopping criterion, $\xi(= 0.7)$: momentum factor, $c_1(= 2.0)$: social coefficient, $c_2(= 2.0)$: cognitive coefficient

output: \mathbf{x}_b : best solution

$x_b \leftarrow x_1$

for $j \leftarrow 1$ **to** n_p **do**

- $\mathbf{x}_{pb,j} \leftarrow \mathbf{x}_j$
- if** $F(\mathbf{x}_j) < F(\mathbf{x}_b)$ **then**
 - $\mathbf{x}_b \leftarrow \mathbf{x}_j$


while $I \leq n_t$ **do**

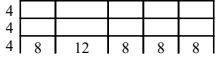
- $I \leftarrow I + 1$
- for** $j \leftarrow 1$ **to** n_p **do**
 - $\mathbf{x}_j \leftarrow \mathbf{x}_j + \mathbf{v}_j$
 - foreach** $x_{j,i} \in \mathbf{x}_j$ **do**
 - if** $x_{j,i} < x_i^L$ **then**
 - $x_{j,i} \leftarrow x_i^L$
 - if** $x_{j,i} > x_i^U$ **then**
 - $x_{j,i} \leftarrow x_i^U$
 - if** $F(\mathbf{x}_j) < F(\mathbf{x}_b)$ **then**
 - $\mathbf{x}_b \leftarrow \mathbf{x}_j$
 - $I \leftarrow 0$
 - if** $F(\mathbf{x}_j) < F(\mathbf{x}_{pb,j})$ **then**
 - $\mathbf{x}_{pb,j} \leftarrow \mathbf{x}_j$
- for** $j \leftarrow 1$ **to** n_p **do**
 - $\mathbf{v}_j \leftarrow \xi \mathbf{v}_j + c_1(\mathbf{x}_b - \mathbf{x}_j) + c_2(\mathbf{x}_{pb,j} - \mathbf{x}_j)$
 - foreach** $v_{j,i} \in \mathbf{v}_j$ **do**
 - if** $v_{j,i} < -v^U$ **then**
 - $v_{j,i} \leftarrow -v^U$
 - if** $v_{j,i} > v^U$ **then**
 - $v_{j,i} \leftarrow v^U$

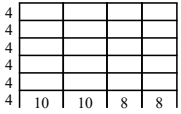
return x_b

The comparative result is described in Table 5.7, in which the elapsed CPU time t_c using Intel(R) Core(TM) i7-8850H CPU @ 2.60GHz and the total structural volume V_s are summarized. Note that the CPU time $t_c[s]$ for RL+GE includes initializing the frame model, importing the trained RL agent, and simulating the design changes of the members until the terminal state. Owing to the penalty terms, the resulting solutions obtained by the PSO algorithm are all feasible that satisfies all the constraints on stress, displacement, and COF. The elapsed CPU time t_c of PSO is the average of 10 optimizations. For 3-span, 5-story frame and 4-span, 6-story frame examples, it is noteworthy that the computational cost of RL+GE is lower than PSO, while the proposed RL+GE method achieves the solution comparable to the solution obtained by PSO with $n_p = 50$ particles in terms of the total structural volume V_s . Although the computational cost of RL+GE is also low in the 5-span, 3-story example, the total structural volume is larger than the median solution obtained by PSO. From this comparison result, it was found that there is still room for further improvement in the accuracy of the training RL agent.

Table 5.7: Comparison between proposed method (RL+GE) and PSO in view of total structural volume $V_s[m^3]$ and CPU time for one optimization $t_c[s]$

problem	n_m	RL+GE Agent(-)	RL+GE Agent(+)	PSO
	35	$\begin{cases} t_c = 2.6 \\ V_s = 6.778 \end{cases}$	$\begin{cases} t_c = 2.3 \\ V_s = 6.418 \end{cases}$	$\begin{cases} t_c = 6.2 \\ V_s = 6.642 \end{cases}$

	33	$\begin{cases} t_c = 1.7 \\ V_s = 7.995 \end{cases}$	$\begin{cases} t_c = 2.4 \\ V_s = 8.207 \end{cases}$	$\begin{cases} t_c = 8.0 \\ V_s = 6.220 \end{cases}$

	54	$\begin{cases} t_c = 3.9 \\ V_s = 12.940 \end{cases}$	$\begin{cases} t_c = 6.0 \\ V_s = 13.554 \end{cases}$	$\begin{cases} t_c = 18.3 \\ V_s = 12.999 \end{cases}$

5.5 Conclusion

In this chapter, the proposed RL method has been demonstrated through the cross-section optimization of steel frames under constraints on stress, displacement, and COF, where the cross-section of each member is selected from a set of standard dimensions. The agent learns sequences of design changes of the member cross-section of the 3-span, 5-story frame through a number of simulations in which the spans between the columns and the floor heights vary for each episode.

Two RL agents are trained separately depending on the task. One agent is

trained to reduce the size of members from the maximum cross-section and the other is trained to increase the size of members from the minimum cross-section. The training is implemented for different learning rates α and reward functions r . The trained RL agents that recorded a higher cumulative reward are capable of changing the cross-sections so as to achieve a minimum-volume design while satisfying the stress, displacement, and COF constraints. The trained RL agents are also applied to 5-span, 3-story and 4-span, 6-story steel frames without re-training and are found to design the cross-sections in a well-balanced manner. The trained RL agent is further compared with PSO in accuracy and efficiency. The obtained feasible design through the trained agent is comparable to the design obtained by the PSO algorithm in the total structural volume, and the computational cost of using the agent is much lower than implementing the PSO algorithm.

Chapter 6

Conclusion

6.1 Summary

This study addresses a combined method of GE and RL for the optimal design of skeletal structures. Although it requires a high computational cost for the training, the trained agent is capable of finding sub-optimal solutions for a skeletal structure with different geometry, topology, and support and loading conditions without re-training. Moreover, the computational cost of using the trained agent is much lower than using metaheuristic methods to obtain an optimal solution. The agent trained by the proposed method exhibited satisfactory performance for simplified structural design problems that requires complex decision making. The background of this study is explained in Chapter 1 as an introduction. The results obtained in Chapters 2-5 are summarized below.

Chapter 2 introduces previous researches on RL, which is the basis of the proposed method. First, the outline of ML methods and the characteristics of RL are explained. Next, the definition and the components of an MDP, which is a decision-making process targeted by RL, are described. The value functions and policies which are important concepts in RL are further introduced. The recent RL methods are based on TD learning. In order to show the importance of TD learning, the DP method and the MC method, which are the basis of TD learning, were explained, and their advantages and disadvantages were summarized. Two representative RL methods SARSA and Q-Learning applying the concept of TD learning are also explained. When using tabular representations, it is difficult to calculate and store the state and action values for all possible combinations of states and actions, which is called the *curse of dimensionality*. To deal with this problem, a function approximation method using an NN is introduced to reduce the size of the training parameters. Finally, a DQN was introduced as an example of combining the NN and RL.

In Chapter 3, The general procedure for converting a structural optimization problem into an RL task was first explained. Since the RL task is formulated as an MDP, the method of defining its components (states, actions, and rewards) are described. In particular, the state is represented by member features expressed as vectors, which are computed from node and member inputs through GE. Here, skeletal structures are regarded as graphs and numerical data about neighbor nodes and members are aggregated to each member. The detail of the

GE formulation for each member was first explained, and another GE formulation that simultaneously computes all the member features using a connectivity matrix was explained next. Although the operations of the above two formulations yield the same result, the latter is much more computationally efficient than the former, owing to efficient matrix multiplication algorithms. Using the extracted member features, the action value to change the design of each member is computed. In the same manner as the member feature extraction, the equation to compute the action values was re-formulated using matrix operations. A simple example with a 5-bar truss was presented to make the above procedure easier to understand. The loss function to train the RL agent for more accurate estimation of action values is defined, and the overall training workflow for minimization of the loss was finally described. The definition of an MDP depends on the structural optimization problem to be solved, and thus cannot be fully generalized. In this study, the definition of an MDP is demonstrated through two examples and the proposed method is applied to them to confirm the validity of the MDP settings and the proposed method.

In Chapter 4, the proposed method was applied to topology optimization of planar trusses for volume minimization under stress and displacement constraints. Note that the proposed method belongs to a binary type approach where the cross-sectional area of each member either keeps its initial value or obtains a very small value. A variety of load and support conditions were provided for an initial GS, and the removal of unnecessary members and the update of the trainable parameters were implemented simultaneously. It was shown that the trained agent is not only capable of obtaining sub-optimal solutions at a low computational cost, but also reusable to other trusses with different geometry, topology, and boundary conditions at a low computational cost. The effects of hyper-parameters such as the learning rate, the size of feature vectors, and the discount rate on training results were also discussed. Furthermore, the action values of removing each member estimated by the trained agent were quantitatively evaluated to confirm that the trained agent correctly evaluates the structural importance of each member. In addition, the trained agent is utilized for generating initial topology for simultaneously optimizing topology and geometry using FDM. By introducing FDM, the nodal coordinates of the equilibrium shape can be obtained as a function of force densities only. The optimization problem to minimize the compliance subject to a total structural volume constraint is also formulated using functions of force densities only. The agent is utilized to reduce the number of design variables by removing unnecessary members before the optimization. This way, a more sparse optimal topology can be obtained at a lower computational cost.

In Chapter 5, the proposed method was applied to another optimization problem: discrete cross-section optimization of planar steel frames. The cross-sections of the steel frames are designed from a prescribed set of standard dimensions so as to minimize the total structural volume while satisfying constraints on stress, displacement, and column-to-beam over-strength factor. For such a difficult problem, the trained agent is capable of finding good solutions comparable to those obtained by metaheuristics. As well as in the previous chapter, the trained agent can be used for frames with different nodes and members at a low computational cost, and it was confirmed that the agent acquired a versatile policy to change the member cross-sections even if the numbers of layers and spans are changed.

6.2 Future outlook

The agent trained with the proposed method is expected to become a supporting tool that instantly feedbacks a sub-optimal structural design. Unlike supervised learning, the trained model learns the sequence towards a desirable state instead of the desirable state only. Therefore, any step of the sequence towards the sub-optimal design can be queried by structural designers, which improves the degree of freedom and flexibility in the interactive design between the RL agent and the designers. This improvement is also expected to enhance our design exploration towards better structural designs.

Although the trained RL model in this study can be applied to more various skeletal structures compared with previous studies, there are still some cases that the trained agent may not apply due to the change of the size of trainable parameters; for example,

- when the objective or the constraints of structural design changes
- when the structural type changes; for example, from a truss to a frame
- when the dimension of the structural model changes between 2D and 3D

For such cases, it is necessary to set up a new agent and train it from the beginning. Although it might also be difficult to apply the trained agent if the scale of the target structure is significantly different from that during the training, it is possible to re-train the agent inheriting the parameters as the initial values. Furthermore, in order to take the advantage of the trained agent at the maximum, it is also necessary to design a graphical user interface (GUI) for the users to utilize the agent.

The architecture, engineering, and construction (AEC) industry has drastically changed along with the advent of digital technology. In the 1990s, software such as AutoCAD [149] has realized the concept of computer-aided design (CAD), which has reduced development costs and shortened design cycles of the buildings by smoothing and visualizing design inputs, analysis processes, and feedback from the computer. The success of the CAD is succeeded by a building information modeling (BIM) in the 2000s. BIM software such as Revit [150] has facilitated exchanges and inter-operations of building information like materials and dimensions by centralizing it into a 3D model-based common framework. As a result, the processes of planning, design, construction and management of buildings by AEC experts in multiple disciplines have become more efficient. The recent trend of parametric and generative design using Grasshopper [151] or Dynamo [152] saw another phase of digitalization in the AEC industry.

ML is widely recognized as a breakthrough of current problems in AI and is also expected as an extension of the AEC digitalization described above. However, there are still excessive expectations for current AI technologies. Frey and Osborne [153] quantified the effect of the recent technological progress on the future of employment and reported that the role of drafting for architectural and civil drawings will be computationalized with the probability of 52% and that of construction and related works with the probability of 71%. The problem is that such shocking numbers come first, and it is not fully understood how the working style of engineers will change due to the current AI technologies.

It must not be ignored that designers have flexibly adapted to the progress of technology and have increased their literacy for new technologies. Nowadays, most architectural designers have become familiar with CAD and BIM

software and know its technological aspects to some extent. In order to benefit from recent AI technologies, it is of great significance at first to know what the currently proposed ML methods can and cannot do in the AEC industry. The author hopes that this doctoral thesis will help all the people involved in architectures to gain an understanding and a perspective towards better collaboration between structural design and AI.

Appendix A

GPU implementation for accelerating the training

The core operation of the learning process by the proposed method is large-scale matrix multiplication, which requires a high computational cost. Therefore, how to improve the efficiency of large-scale matrix operations greatly affects the learning efficiency. This appendix discusses in detail the techniques used in Chapter 5 to improve the efficiency of matrix operations with the GPU from both hardware and software perspectives.

A central processing unit (CPU) is a part of electronic circuits in a computer to execute a sequence of instructions called program. The CPU is responsible for the entire calculation process of the computer, and exhibits excellent performance in sequentially processing complex tasks.

On the other hand, a graphics processing unit (GPU) is originally designed for rapid manipulation of graphics and their outputs onto a display device. The GPUs are roughly divided into those built into the CPU called Integrated GPU (IGPU) and those having their own source of random access memory (RAM) independent from the CPU called dedicated (or discrete) GPU. Although IGPU are usually cheaper and uses less power compared with dedicated GPUs, since only a small percentage of memory can be assigned for image processing tasks, the performance of IGPU is limited for intensive tasks. On the other hand, dedicated GPUs can allocate intensive tasks to their own memory and thus superior to IGPU in terms of performance. GPUs generally have a much larger number of cores than CPUs. While each core of the GPU is operated at lower frequencies compared with the CPU, the GPU cores work in parallel to handle large-scale data more quickly.

Although GPUs typically handle computations for graphics, there is an increasing number of applications of general-purpose computing on graphics processing units (GPGPU), in which the computation traditionally handled by a CPU is performed by a GPU. The GPGPU is possible to allocate the calculation processing to the processors utilizing each ability so that a GPU is in charge of the calculation process with a large computational load and a CPU is in charge of other sequential processing.

The utilization of GPGPU requires an application programming interface (API) that allows software developers to build their own algorithms and pass

intensive kernels to the GPU. Nvidia corporation developed the compute unified device architecture (CUDA) that is a parallel computing platform and an API for GPGPU. By using CUDA, instructions of high-speed arithmetic processing using GPU can be easily written using general-purpose programming languages such as C, C++, and Fortran without requiring programming knowledge about graphics programming.

For numerical computing over multi-dimensional arrays and matrices, NumPy [154] is the most prevalent library for the Python programming language. CuPy [155] is an open-source library for accelerated computing with NVIDIA GPUs and CUDA platform. CuPy was born as a back-end of Chainer [156, 157, 158], which is an open-source framework for training NNs using back-propagation methods. CuPy became separated from Chainer as an independent library in 2017. CuPy automatically wraps and compiles the code to make CUDA kernels, using C++ codes. As shown in Codes A.1 and A.2, CuPy's syntax is highly compatible with NumPy; both operations yield the same result $\mathbf{Y} = \mathbf{W}\mathbf{X} = \{10, 10, 10\}$. The array is transferred from a GPU to a CPU in the second last line of Code A.2, and the transferred array is re-transferred from the CPU to the GPU in the last line of Code A.2.

Listing A.1: matrix operation using NumPy

```

1 import numpy as np
2
3 X_cpu = np.ones((10,))
4 W_cpu = np.ones((10,3))
5 Y_cpu = np.dot(X_cpu, W_cpu)

```

Listing A.2: matrix operation using CuPy

```

1 import cupy as cp
2
3 X_gpu = cp.ones((10,))
4 W_gpu = cp.ones((10,3))
5 Y_gpu = cp.dot(X_gpu, W_gpu)
6
7 Y_cpu = cp.asarray(Y_gpu)
8 Y_gpu2 = cp.asnumpy(Y_cpu)

```

In the example of truss topology optimization in Chapter 4, the training is performed by a CPU only using NumPy and Chainer. In the example of frame cross-section optimization in Chapter 5, the training was accelerated by using CuPy and Chainer for weighting input values by trainable parameters Θ and tuning trainable parameters using the RMSProp. Note that NumPy instead of CuPy is used to weight the input values by trainable parameters Θ after the training so that the trained agent becomes available on a PC that does not have a dedicated GPU or does not install CUDA.

Bibliography

- [1] W. Hsu, B. Liu, Conceptual design: issues and challenges, *Computer-Aided Design* 32 (2000) 849 – 850.
- [2] C. Mueller, J. Ochsendorf, From analysis to design: A new computational strategy for structural creativity, in: *2nd International Workshop on Design in Civil and Environmental Engineering*, pp. 1–11.
- [3] C. Cui, M. Sasaki, Creation of three-dimensional structural form by extended eso method (application to structural design), 2003. (in Japanese).
- [4] T. Kimura, Y. Hiraiwa, M. Sasaki, Structural design of the crematorium in kawaguchi, 2016.
- [5] T. Zegard, C. Hartz, A. Mazurek, W. F. Baker, Advancing building engineering through structural and topology optimization, *Structural and Multidisciplinary Optimization* 62 (2020) 915–935.
- [6] M. Banachowicz, Nonlinear shaping architecture designed with using evolutionary structural optimization tools, *IOP Conference Series: Materials Science and Engineering* 245 (2017) 1–9.
- [7] K. S. S. Co.Ltd., Kawaguchi city megurino-mori, 2020 (accessed September 27, 2020). <http://www.kawaguchishi-megurinomori.jp/>.
- [8] H. Busta, SOM and Oak Ridge National Lab debut 3D-printed shelter and car, 2020 (accessed October 1, 2020). https://www.architectmagazine.com/technology/som-and-oak-ridge-national-lab-debut-3d-printed-shelter-and-car_o.
- [9] D. Luenberger, *Linear and Nonlinear Programming*, Addison-Wesley, 1984.
- [10] N. Karmarkar, A new polynomial-time algorithm for linear programming, in: *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing, STOC '84*, Association for Computing Machinery, New York, NY, USA, 1984, p. 302–311.
- [11] H. B. Curry, The method of steepest descent for non-linear minimization problems, *Quarterly of Applied Mathematics* 2 (1944) 258–261.

- [12] P. E. Gill, W. Murray, M. A. Saunders, M. H. Wright, Sequential quadratic programming methods for nonlinear programming, in: E. J. Haug (Ed.), *Computer Aided Analysis and Optimization of Mechanical System Dynamics*, Springer Berlin Heidelberg, Berlin, Heidelberg, 1984, pp. 679–700.
- [13] M. Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, MA, USA, 1998.
- [14] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, Optimization by simulated annealing, *Science* 220 (1983) 671–680.
- [15] J. Kennedy, R. Eberhart, Particle swarm optimization, in: *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pp. 1942–1948.
- [16] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattemberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, 2020 (accessed September 3, 2020). <https://www.tensorflow.org/>.
- [17] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic differentiation in pytorch, in: *NIPS-W*, pp. 1–4.
- [18] MathWorks, *Statistics and machine learning toolbox*, 2020 (accessed September 3, 2020). <https://uk.mathworks.com/products/statistics.html>.
- [19] D. Prayogo, M.-Y. Cheng, Y.-W. Wu, D.-H. Tran, Combining machine learning models via adaptive ensemble weighting for prediction of shear capacity of reinforced-concrete deep beams, *Engineering with Computers* (2019).
- [20] J.-S. Chou, A.-D. Pham, Enhanced artificial intelligence for ensemble approach to predicting high performance concrete compressive strength, *Construction and Building Materials* 49 (2013) 554 – 563.
- [21] M. Khandelwal, Blast-induced ground vibration prediction using support vector machine, *Engineering with Computers* 27 (2011) 193–200.
- [22] T. Tamura, M. Ohsaki, J. Takagi, Machine learning for combinatorial optimization of brace placement of steel frames, *Japan Architectural Review* 1 (2018) 419–430.
- [23] M. Papadrakakis, N. D. Lagaros, Y. Tsompanakis, Structural optimization using evolution strategies and neural networks, *Computer Methods in Applied Mechanics and Engineering* 156 (1998) 309 – 333.

- [24] A. A. Markov, N. M. Nagorny, *The Theory of Algorithms*, Springer Publishing Company, Incorporated, 1st edition, 2010.
- [25] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, D. Hassabis, Mastering the game of go without human knowledge, *Nature* 550 (2017) 354–359.
- [26] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, *Nature* 518 (2015) 529–533.
- [27] S. Nakamura, T. Suzuki, High-speed calculation in structural analysis by reinforcement learning, *The 32nd Annual Conference of the Japanese Society for Artificial Intelligence JSAI2018* (2018) 3K1OS18a01. (in Japanese).
- [28] D. Chiba, M. Suzuki, M. Hayashi, K. Watanabe, Development of active response control system using ai (part 2. creation of ai for response control by deep reinforcement learning), 2018. (in Japanese).
- [29] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, Backpropagation applied to handwritten zip code recognition, *Neural Computation* 1 (1989) 541–551.
- [30] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, J. Schmidhuber, High-performance neural networks for visual object classification, *CoRR* abs/1102.0183 (2011).
- [31] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: *the 25th International Conference on Neural Information Processing Systems - Volume 1, NIPS'12*, Curran Associates Inc., USA, 2012, pp. 1097–1105.
- [32] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, Y. Bengio, Generative adversarial networks, *ArXiv* abs/1406.2661 (2014).
- [33] M. Radovic, O. Adarkwa, Q. Wang, Object recognition in aerial images using convolutional neural networks, *Journal of Imaging* 3 (2017).
- [34] A. Uçar, Y. Demir, C. Güzeliundefined, Object recognition and detection with deep learning for autonomous driving applications, *Simulation* 93 (2017) 759–769.
- [35] S. Hershey, S. Chaudhuri, D. P. W. Ellis, J. F. Gemmeke, A. Jansen, C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, M. Slaney, R. Weiss, K. Wilson, Cnn architectures for large-scale audio classification, in: *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 1–5.

- [36] Y. Kim, Convolutional neural networks for sentence classification, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, Doha, Qatar, 2014, pp. 1746–1751.
- [37] Y. Yu, T. Hur, J. Jung, Deep learning for topology optimization design, ArXiv abs/1801.05463 (2018).
- [38] S. Banga, H. Gehani, S. Bhilare, S. Patel, L. Kara, 3d topology optimization using convolutional neural networks, arXiv abs/1808.07440 (2018).
- [39] Non-iterative structural topology optimization using deep learning, Computer-Aided Design 115 (2019) 172–180.
- [40] Y. Zhang, A. Chen, B. Peng, X. Zhou, D. Wang, A deep convolutional neural network for topology optimization with strong generalization ability, arXiv abs/1901.07761 (2019).
- [41] I. Sosnovik, I. Oseledets, Neural networks for topology optimization, Russian Journal of Numerical Analysis and Mathematical Modelling 34 (2019) 215 – 223.
- [42] Z. Nie, T. Lin, H. Jiang, L. B. Kara, TopologyGAN: Topology optimization using generative adversarial networks based on physical fields over the initial domain, ArXiv abs/2003.04685 (2020).
- [43] Z. Nie, H. Jiang, L. Kara, Stress field prediction in cantilevered structures using convolutional neural networks, Journal of Computing and Information Science in Engineering 20 (2019).
- [44] H. Jiang, Z. Nie, R. Yeo, A. B. Farimani, L. B. Kara, StressGAN: A Generative Deep Learning Model for 2D Stress Distribution Prediction, in: International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, volume Volume 11B: 46th Design Automation Conference (DAC).
- [45] A. Liew, R. Avelino, V. Moosavi, T. Van Mele, P. Block, Optimising the load path of compression-only thrust networks through independent sets, Structural and Multidisciplinary Optimization 60 (2019) 231–244.
- [46] H. Cai, V. W. Zheng, K. Chang, A comprehensive survey of graph embedding: Problems, techniques, and applications, IEEE Transactions on Knowledge and Data Engineering 30 (2018) 1616–1637.
- [47] F. A. Faber, L. Hutchison, B. Huang, J. Gilmer, S. S. Schoenholz, G. E. Dahl, O. Vinyals, S. Kearnes, P. F. Riley, O. A. von Lilienfeld, Machine learning prediction errors better than DFT accuracy, CoRR abs/1702.05532 (2017).
- [48] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, G. E. Dahl, Neural message passing for quantum chemistry, CoRR abs/1704.01212 (2017).

- [49] B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: Online learning of social representations, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, Association for Computing Machinery, New York, NY, USA, 2014, p. 701–710.
- [50] I. Bello, H. Pham, Q. V. Le, M. Norouzi, S. Bengio, Neural combinatorial optimization with reinforcement learning, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Workshop Track Proceedings, OpenReview.net, 2017, pp. 1–5.
- [51] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, L. Song, Learning combinatorial optimization algorithms over graphs, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, pp. 6351–6361.
- [52] J. Jiang, C. Dun, Z. Lu, Graph convolutional reinforcement learning for multi-agent cooperation, CoRR abs/1810.09202 (2018).
- [53] A. Malysheva, T. T. K. Sung, C. Sohn, D. Kudenko, A. Shpilman, Deep multi-agent reinforcement learning with relevance graphs, CoRR abs/1811.12557 (2018).
- [54] W. Zheng, D. Wang, F. Song, Opengraphgym: A parallel reinforcement learning framework for graph optimization problems, in: V. V. Krzhizhanovskaya, G. Závodszy, M. H. Lees, J. J. Dongarra, P. M. A. Sloot, S. Brissos, J. Teixeira (Eds.), Computational Science - ICCS 2020 - 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings, Part V, volume 12141 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 439–452.
- [55] A. Grover, J. Leskovec, Node2vec: Scalable feature learning for networks, in: KDD : proceedings. International Conference on Knowledge Discovery and Data Mining, KDD '16, Association for Computing Machinery, New York, NY, USA, 2016, p. 855–864.
- [56] N. Zhao, H. Zhang, M. Wang, R. Hong, T.-S. Chua, Learning content–social influential features for influence analysis, *International Journal of Multimedia Information Retrieval* 5 (2016) 137–149.
- [57] C. Wang, C. Wang, Z. Wang, X. Ye, P. S. Yu, Edge2vec: Edge-based social network embedding 14 (2020).
- [58] R. Furuta, M. Yamakawa, N. Katoh, K. Araki, M. Ohsaki, A design method for optimal truss structures with redundancy based on combinatorial rigidity theory, *Journal of Structural and Construction Engineering* 79 (2014) 583–592. (in Japanese).
- [59] R. Razani, Behavior of fully stressed design of structures and its relationship to minimum-weight design, *AIAA Journal* 3 (1965) 2262–2268.
- [60] U. Kirsch, Optimal topologies of truss structures, *Computer Methods in Applied Mechanics and Engineering* 72 (1989) 15–28.

- [61] M. P. Bendsøe, O. Sigmund, *Topology Optimization: Theory, Methods and Applications*, Springer, 2004.
- [62] W. S. Dorn, R. E. Gomory, H. J. Greenberg, Automatic design of optimal structures, *Journal de Mecanique* 3 (1964) 25–52.
- [63] M. Ohsaki, Genetic algorithm for topology optimization of trusses, *Computers and Structures* 57 (1995) 219–225.
- [64] G. Cheng, X. Guo, ε -relaxed approach in structural topology optimization, *Structural Optimization* 13 (1997) 258–266.
- [65] W. Achtziger, M. Stolpe, Global optimization of truss topology with discrete bar areas—part ii: Implementation and numerical results, *Computational Optimization and Applications* 44 (2009) 315–341.
- [66] C. Y. Sheu, L. A. Schmit, Jr., Minimum Weight Design of Elastic Redundant Trusses under Multiple Static Loading Conditions, *AIAA Journal* 10 (1972) 155–162.
- [67] U. T. Ringertz, A branch and bound algorithm for topology optimization of truss structures, *Engineering Optimization* 10 (1986) 111–124.
- [68] M. Ohsaki, N. Katoh, Topology optimization of trusses with stress and local constraints on nodal stability and member intersection, *Structural and Multidisciplinary Optimization* 29 (2005) 190–197.
- [69] P. Hajela, E. Lee, Genetic algorithms in truss topological optimization, *International Journal of Solids and Structures* 32 (1995) 3341 – 3357.
- [70] B. Topping, A. Khan, J. Leite, Topological design of truss structures using simulated annealing, *Structural Engineering Review* 8 (1996) 301–304.
- [71] T. Hagishita, M. Ohsaki, Topology optimization of trusses by growing ground structure method, *Structural and Multidisciplinary Optimization* 37 (2009) 377–393.
- [72] M. Ohsaki, K. Hayashi, Force density method for simultaneous optimization of geometry and topology of trusses, *Structural and Multidisciplinary Optimization* 56 (2017) 1157–1168.
- [73] K. Hayashi, M. Ohsaki, FDMopt: Force density method for optimal geometry and topology of trusses, *Advances in Engineering Software* 133 (2019) 12–19.
- [74] P. Hajela, L. Berke, Neural network based decomposition in optimal structural synthesis, *Computing Systems in Engineering* 2 (1991) 473 – 481.
- [75] S. Lee, J. Ha, M. Zokhirova, H. Moon, J. Lee, Background information of deep learning for structural engineering, *Archives of Computational Methods in Engineering* 25 (2018) 121–129.
- [76] R. Swift, S. Batill, Application of neural networks to preliminary structural design.

- [77] Z. Szewczyk, P. Hajela, Neural network approximations in a simulated annealing based optimal structural design, *Structural optimization* 5 (1993) 159–165.
- [78] R. Hecht-Nielsen, Counterpropagation networks, *Applied Optics* 26 (1987) 4979–4984.
- [79] M. J. D. Powell, Algorithms for nonlinear constraints that use lagrangian functions, *Mathematical Programming* 14 (1978) 224–248.
- [80] T. Kimura, M. Ohsaki, R. Okazaki, Simultaneous optimization of brace locations and cross-sections of beams and columns of steel frames, *Journal of Structural and Construction Engineering* 83 (2018) 1445–1454. (in Japanese).
- [81] N. Tamura, H. Ohmori, Supporting system for structural design of steel frame structures by using multi-objective optimization method, *Journal of Structural and Construction Engineering* 73 (2008) 891–897. (in Japanese).
- [82] K. Hager, R. Balling, New approach for discrete structural optimization, *Journal of Structural Engineering* 114 (1988) 1120–1134.
- [83] S. Yoshitomi, M. Yamakawa, K. Uetani, A method for selecting optimum discrete sections of steel frames using two-step relaxation, *Journal of Structural and Construction Engineering* 69 (2004) 95–100. (in Japanese).
- [84] A. L. Samuel, Some studies in machine learning using the game of checkers, *IBM Journal of Research and Development* 3 (1959) 210–229.
- [85] M. Mohri, A. Rostamizadeh, A. Talwalkar, *Foundations of Machine Learning*, The MIT Press, 2012.
- [86] E. Togootogtokh, A. Amartuvshin, Deep learning approach for very similar objects recognition application on chihuahua and muffin problem, *ArXiv abs/1801.09573* (2018).
- [87] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: *International Conference on Learning Representations*.
- [88] L. van der Maaten, G. Hinton, Visualizing data using t-SNE, *Journal of Machine Learning Research* 9 (2008) 2579–2605.
- [89] S. Gu, E. Holly, T. Lillicrap, S. Levine, Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates, in: *Proceedings 2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, Piscataway, NJ, USA, 2017.
- [90] IBM, Models for machine learning, 2020 (accessed September 7, 2020). <https://developer.ibm.com/articles/cc-models-machine-learning/>.
- [91] B. F. Skinner, *Science and human behavior*, New York, Macmillan, 1953.

- [92] D. Schacter, D. T. Gilbert, D. M. Wegner, *Psychology* (2nd Edition), Worth, New York, 2011.
- [93] E. L. Thorndike, Animal intelligence, *Psych Revmonog* 8 (1911).
- [94] R. S. Sutton, A. G. Barto, *Introduction to Reinforcement Learning*, MIT Press, Cambridge, MA, USA, 1st edition, 1998.
- [95] R. Bellman, On the theory of dynamic programming, *Proceedings of the National Academy of Sciences* 38 (1952) 716–719.
- [96] R. Bellman, *Dynamic Programming*, Rand Corporation research study, Princeton University Press, 1957.
- [97] R. Bellman, A markovian decision process, *Indiana Univ. Math. J.* 6 (1957) 679–684.
- [98] R. A. Howard, *Dynamic Programming and Markov Processes*, Technology Press and Wiley, 1960.
- [99] D. Michie, R. A. Chambers, BOXES: An experiment in adaptive control, in: E. Dale, D. Michie (Eds.), *Machine Intelligence*, Oliver and Boyd, Edinburgh, UK, 1968.
- [100] A. G. Barto, R. S. Sutton, C. W. Anderson, Neuronlike adaptive elements that can solve difficult learning control problems, *IEEE Transactions on Systems, Man, and Cybernetics SMC-13* (1983) 834–846.
- [101] OpenAI, Cartpole-v1, 2020 (accessed October 1, 2020). <https://gym.openai.com/envs/CartPole-v1/>.
- [102] C. J. C. H. Watkins, *Learning from Delayed Rewards*, Ph.D. thesis, King’s College, Cambridge, UK, 1989.
- [103] P. Maes, I. behavior, J. Meyer, M. Mataric, S. Wilson, J. Pollack, *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, Bradford book, MIT Press, 1996.
- [104] O. Berger-Tal, J. Nathan, E. Meron, D. Saltz, The exploration-exploitation dilemma: A multidisciplinary framework, *PLOS ONE* 9 (2014) 1–8.
- [105] M. A. Wiering, *Explorations in efficient reinforcement learning*, Ph.D. thesis, University of Amsterdam, 1999.
- [106] J. Langford, *Efficient Exploration in Reinforcement Learning*, Springer US, Boston, MA, pp. 389–392.
- [107] M. Tokic, Adaptive ϵ -greedy exploration in reinforcement learning based on value differences, in: R. Dillmann, J. Beyerer, U. D. Hanebeck, T. Schultz (Eds.), *KI 2010: Advances in Artificial Intelligence*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 203–210.
- [108] A. Mignon, R. L. A. Rocha, An adaptive implementation of ϵ -greedy in reinforcement learning, *Procedia Computer Science* 109 (2017) 1146–1151.

- [109] R. Bellman, The theory of dynamic programming, *Bulletin of the American Mathematical Society* 60 (1954) 503–515.
- [110] S. P. Singh, R. S. Sutton, Reinforcement learning with replacing eligibility traces, *Machine Learning* 22 (1996) 123–158.
- [111] R. S. Sutton, Learning to predict by the methods of temporal differences, *Machine Learning* 3 (1988) 9–44.
- [112] G. A. Rummery, M. Niranjan, On-Line Q-Learning Using Connectionist Systems, Technical Report, Cambridge University, 1994.
- [113] C. J. C. H. Watkins, P. Dayan, Technical note: Q-learning, *Machine Learning* 8 (1992) 279–292.
- [114] M. Roderick, J. MacGlashan, S. Tellex, Implementing the Deep Q-Network, *CoRR abs/1711.07478* (2017).
- [115] R. Bellman, *Adaptive Control Processes*, Princeton University Press, 1961.
- [116] N. J. van Eck, M. van Wezel, Application of reinforcement learning to the game of othello, *Computers and Operations Research* 35 (2008) 1999–2017.
- [117] C. E. Shannon, XXII. Programming a computer for playing chess, *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 41 (1950) 256–275.
- [118] B. T. Polyak, A. B. Juditsky, Acceleration of stochastic approximation by averaging, *SIAM Journal on Control and Optimization* 30 (1992) 838–855.
- [119] L. Bottou, On-line learning and stochastic approximations, in: *On-line Learning in Neural Networks*, Cambridge University Press, 1998, pp. 9–42.
- [120] T. Tieleman, G. Hinton, Lecture 6e - rmsprop: Divide the gradient by a running average of its recent magnitude, *COURSERA: Neural Networks for Machine Learning*, 2012.
- [121] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning representations by back-propagating errors, *Nature* 323 (1986) 533–536.
- [122] L.-J. Lin, *Reinforcement Learning for Robots Using Neural Networks*, Ph.D. thesis, USA, 1992.
- [123] J. Sola, J. Sevilla, Importance of input data normalization for the application of neural networks to complex industrial problems, *IEEE Transactions on Nuclear Science* 44 (1997) 1464–1468.
- [124] M. Puheim, L. Madarász, Normalization of inputs and outputs of neural network based robotic arm controller in role of inverse kinematic model, in: *SAMI 2014 - IEEE 12th International Symposium on Applied Machine Intelligence and Informatics*, p. 4.

- [125] J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, in: Proceedings of the 24th International Conference on Neural Information Processing Systems, NIPS'11, Curran Associates Inc., Red Hook, NY, USA, 2011, p. 2546–2554.
- [126] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, *Journal of Machine Learning Research* 13 (2012) 281–305.
- [127] J. Snoek, H. Larochelle, R. P. Adams, Practical bayesian optimization of machine learning algorithms, in: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2, NIPS'12, Curran Associates Inc., Red Hook, NY, USA, 2012, p. 2951–2959.
- [128] N. Vithayathil Varghese, Q. H. Mahmoud, A survey of multi-task deep reinforcement learning, *Electronics* 9 (2020) 21.
- [129] W. Miller, Computational complexity and numerical stability, *STOC '74*, Association for Computing Machinery, New York, NY, USA, 1974, p. 317–322.
- [130] S. S. Skiena, *Sorting and Searching*, Springer London, London, pp. 103–144.
- [131] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, Third Edition, The MIT Press, 3rd edition, pp. 75–59.
- [132] F. Le Gall, Powers of tensors and fast matrix multiplication, in: Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Association for Computing Machinery, New York, NY, USA, 2014, p. 296–303.
- [133] I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*, MIT Press, 2016.
- [134] W. McGuire, R. Gallagher, R. Ziemian, *Matrix Structural Analysis*, Wiley, 2015.
- [135] X. Guo, Z. Du, G. Cheng, C. Ni, Symmetry properties in structural optimization: Some extensions, *Structural Multidisciplinary Optimization* 47 (2013) 783–794.
- [136] H.-J. Schek, The force density method for form finding and computation of general networks, *Computer Methods in Applied Mechanics and Engineering* 3 (1974) 115–134.
- [137] J. Zhang, M. Ohsaki, *Tensegrity Structures: Form, Stability, and Symmetry*, Mathematics for Industry, Springer Japan, 2015.
- [138] A. Tibert, S. Pellegrino, Review of form-finding methods for tensegrity structures, *International Journal of Space Structures* 18 (2003) 209–223.
- [139] J. Zhang, M. Ohsaki, Adaptive force density method for form-finding problem of tensegrity structures, *International Journal of Solids and Structures* 43 (2006) 5658 – 5673.

- [140] W. S. Hemp, *Optimum structures*, Clarendon Press Oxford, 1973.
- [141] W. Achtziger, On simultaneous optimization of truss geometry and topology, *Structural and Multidisciplinary Optimization* 33 (2007) 285–304.
- [142] P. E. Gill, W. Murray, M. A. Saunders, SNOPT: An SQP algorithm for large-scale constrained optimization, *SIAM JOURNAL ON OPTIMIZATION* 12 (1997) 979–1006.
- [143] J. Stoer, Principles of sequential quadratic programming methods for solving nonlinear programs, in: K. Schittkowski (Ed.), *Computational Mathematical Programming*, Springer Berlin Heidelberg, Berlin, Heidelberg, 1985, pp. 165–207.
- [144] S. K. Eldersveld, *Large-Scale Sequential Quadratic Programming Algorithms*, Ph.D. thesis, Stanford University, Stanford, CA, USA, 1992.
- [145] Ministry of Land, Infrastructure, Transport and Tourism of Japan, *Technical Standards Manual on Building Structures: 2007 Edition*, National Marketing Cooperative of Official Gazettes, second edition, 2008. (in Japanese).
- [146] Ministry of Land, Infrastructure, Transport and Tourism of Japan, Notice of milt no.1024: on determining special allowable stress and special material strength, 2001. (in Japanese).
- [147] M. Nakashima, S. Sawaizumi, Column-to-beam strength ratio required for ensuring beam-collapse mechanism in earthquake responses of steel frames, in: *Proceedings of the 12 th World Conference on Earthquake Engineering*.
- [148] J. Kennedy, The particle swarm: social adaptation of knowledge, in: *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97)*, pp. 303–308.
- [149] B.-C. Björk, M. Laakso, CAD standardisation in the construction industry —a process view, *Automation in Construction* 19 (2010) 398 – 406.
- [150] M. Imtaar, *Complete Technical BIM Project Using Autodesk Revit: Architecture - Structure - MEP*, CreateSpace Independent Publishing Platform, North Charleston, SC, USA, 2016.
- [151] R. McNeel, *Grasshopper - Algorithmic modeling for rhino*, <http://www.grasshopper3d.com/>, 2020. Accessed: 2020-10-27.
- [152] Autodesk, *Open source graphical programming for design*, 2020 (accessed September 17, 2020). <https://dynamobim.org/>.
- [153] C. B. Frey, M. A. Osborne, *The Future of Employment: How Susceptible are Jobs to Computerisation?*, Working Paper, Oxford Martin School, University of Oxford, Oxford, UK, 2013.

- [154] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del R'io, M. Wiebe, P. Peterson, P. G'erald-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, T. E. Oliphant, Array programming with NumPy, *Nature* 585 (2020) 357–362.
- [155] R. Okuta, Y. Unno, D. Nishino, S. Hido, C. Loomis, Cupy: A numpy-compatible library for nvidia gpu calculations, in: Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS).
- [156] S. Tokui, K. Oono, S. Hido, J. Clayton, Chainer: a next-generation open source framework for deep learning, in: Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS).
- [157] T. Akiba, K. Fukuda, S. Suzuki, ChainerMN: Scalable Distributed Deep Learning Framework, in: Proceedings of Workshop on ML Systems in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS).
- [158] S. Tokui, R. Okuta, T. Akiba, Y. Niitani, T. Ogawa, S. Saito, S. Suzuki, K. Uenishi, B. Vogel, H. Yamazaki Vincent, Chainer: A deep learning framework for accelerating the research cycle, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, pp. 2002–2011.

List of presentations related to this study

Peer-reviewed journal

1. Kazuki Hayashi and Makoto Ohsaki. Reinforcement learning for optimum design of a plane frame under static loads. *Engineering with Computers*, 2020. (published online)
2. Kazuki Hayashi and Makoto Ohsaki. Reinforcement learning and graph embedding for binary truss topology optimization under stress and displacement constraints. *Frontiers in Built Environment*, 6(59), Apr 2020
3. Kazuki Hayashi and Makoto Ohsaki. FDMopt: Force density method for optimal geometry and topology of trusses. *Advances in Engineering Software*, 133:12–19, Jul 2019
4. Makoto Ohsaki and Kazuki Hayashi. Force density method for simultaneous optimization of geometry and topology of trusses. *Structural and Multidisciplinary Optimization*, 56(5):1157–1168, Nov 2017

International conference with full paper

1. Kazuki Hayashi, Makoto Ohsaki, and Caitlin Mueller. FDMOPT: A new tool for simultaneous optimization of geometry and topology of truss structures. In *IASS symposium 2018, Boston, USA*. International Association of Shell and Spatial Structures, Jul 2018
2. Kazuki Hayashi, Makoto Ohsaki, and Caitlin Mueller. Force density method for simultaneous optimization of geometry and topology of spatial trusses. In *IASS symposium 2017, Hamburg, Germany*. International Association of Shell and Spatial Structures, Sep 2017

Other oral presentations

1. Kazuki Hayashi and Makoto Ohsaki. Minimum-volume design of steel frames using reinforcement learning. In *ECCOMAS Congress 2020 and 14th WCCM*, number P00322. International Association for Computational Mechanics (IACM) and European Community on Computational Methods in Applied Sciences (ECCOMAS), Jan 2021

2. Kazuki Hayashi and Makoto Ohsaki. Development of an agent for discrete cross-section design of planar steel frames using graph embedding and reinforcement learning. In *43rd Symposium on Computer Technology of Information, Systems and Applications*, number H03. Architectural Institute of Japan, Dec 2020. (in Japanese)
3. Kazuki Hayashi and Makoto Ohsaki. Reinforcement learning and graph embedding for sequential optimal design of plane trusses and frames. In *Asian Congress of Structural and Multidisciplinary Optimization(ACSMO), Seoul, Korea*, number P00322. Asian Society of Structural and Multidisciplinary Optimization (ASSMO) and Korean Society for Design Optimization (KSDO), Nov 2020
4. Kazuki Hayashi and Makoto Ohsaki. Reinforcement learning and graph embedding for cross-sectional design of planar steel frames. In *Summaries of technical papers of annual meeting (Structure I)*. Architectural Institute of Japan, Jul 2020. (in Japanese)
5. Kazuki Hayashi and Makoto Ohsaki. Graph embedding and reinforcement learning for topology optimization of planar trusses with stress and displacement constraints. In *Summaries of technical reports of annual meeting (Structure)*, number 2044, pages 193–196. Architectural Institute of Japan Kinki Branch, Jun 2020. (in Japanese)
6. Kazuki Hayashi and Makoto Ohsaki. Deep-Q network for truss topology optimization with stress constraints. In *IASS symposium 2019, Barcelona, Spain*, number 249. Internatinal Assosication of Shell and Spatial Structures, Oct 2019
7. Kazuki Hayashi and Makoto Ohsaki. Shape optimization of frame structures using dynamic programming. In *the 41st Symposium on Computer Technology of Information, Systems and Applications*, number H14. Architectural Institute of Japan, Dec 2018. (in Japanese)
8. Kazuki Hayashi and Makoto Ohsaki. Force density method for simultaneous optimization of geometry and topology for latticed shells with free-form design surface. In *Summaries of technical papers of annual meeting (Structure I)*, number 20180, pages 359–360. Architectural Institute of Japan, Jul 2018. (in Japanese)
9. Kazuki Hayashi and Makoto Ohsaki. Force density method for simultaneous optimization of geometry and topology of trusses. In *Summaries of technical papers of annual meeting (Structure I)*, number 20514, pages 1027–1028. Architectural Institute of Japan, Jul 2017. (in Japanese)
10. Kazuki Hayashi and Makoto Ohsaki. Simultaneous topology and shape optimization by force density method - Analysis of constraint method for member length and nodal position. In *the 40th Symposium on Computer Technology of Information, Systems and Applications*, number H82. Architectural Institute of Japan, Dec 2017. (in Japanese)
11. Kazuki Hayashi and Makoto Ohsaki. Simultaneous optimization of topology and geometry of planar truss using force density as design variable. In *the 11th colloquium analysis and generation of structural shapes and systems*, number 16. Architectural Institute of Japan, Oct 2016. (in Japanese)

List of other presentations

Peer-reviewed journal

1. Kazuki Hayashi, Yoshiaki Jikumaru, Makoto Ohsaki, Takashi Kagaya, and Yohei Yokosuka. Discrete Gaussian curvature flow for piecewise constant Gaussian curvature surface. *Computer-Aided Design*, Jan 2021. (published online)
2. Hiroto Ota, Takuya Ito, and Kazuki Hayashi. Design review system with “Live AHP” to visualize and share jury’s own decision evaluation. *The AIJ Journal of Technology and Design*, 27(65):562–567, Feb 2021. (in Japanese, accepted for publication)

International conference with full paper

1. Kazuki Hayashi and Makoto Ohsaki. Structural performance of triangular latticed shells with regularized panels for bézier design surfaces. In *12th Asian Pacific Conference on Shell and Spatial Structures (APCS2018)*, pages 337–347, Oct 2018
2. Kazuki Hayashi and Makoto Ohsaki. Regularization of triangular latticed shell panels for bézier surfaces. In *IASS symposium 2018, Boston, USA*. International Association of Shell and Spatial Structures, Jul 2018

Other oral presentations

1. Hiroto Ota, Takuya Ito, and Kazuki Hayashi. Study on communication environment to visualize and share jury’s own ”design evaluation” - Design review system with AHP application. In *Summaries of technical papers of annual meeting (education)*, number 50476. Architectural Institute of Japan, Jul 2020. (in Japanese)
2. Hiroto Ota and Kazuki Hayashi. Development of an application to visualize and share user’s own design review expressed as AHP - A report of experiment in the review-event “Diploma x Kyoto”. In *Summaries of presentations of Design symposium*, pages 143–148, Nov 2019. (in Japanese)
3. Kazuki Hayashi and Makoto Ohsaki. Regularization of edge length of triangular panels for latticed shells with free-form surfaces. In *Summaries of technical reports of annual meeting (Structure)*, number 58, pages 293–296. Architectural Institute of Japan Kinki Branch, Jun 2018. (in Japanese)

4. Kazuki Hayashi and Hiroto Ota. A study on critique of architectural and urban design by the method of using ahp. In *Summaries of technical papers of annual meeting (architectural planning)*, number 5448, pages 895–896. Architectural Institute of Japan, Sep 2015. (in Japanese)

Acknowledgement

This research was kindly sponsored by Grant-in-Aid for JSPS Research Fellow No.JP18J21456, JSPS Overseas Challenge Program for Young Researchers, and JSPS KAKENHI No.JP18K18898.

I would like to express my heartfelt gratitude to my supervisor, Professor Makoto Ohsaki, whose expertise in structural optimization has been invaluable in formulating the research questions and methodology. Not only did he teach me countless lessons about how to work in the academic world, but he always understood what I wanted to do and believed in my decision.

I would like to acknowledge the vice-chairs in my dissertation committee, Professor Izuru Takewaki, and Associate Professor Masahiro Kurata. Their expertise, guidance, and encouragement have been crucial for the work of this thesis. I also thank Professor Kei Senda, an expert on reinforcement learning, for his valuable comments on applying structural design problems to reinforcement learning. I would like to thank Professor Yoshikazu Araki and Associate Professor Jingyao Zhang, for their insightful feedback in the meetings that have brought my work to a higher level. I give sincere thanks to Dr. Toshiaki Kimura for his many forms of support in my research life. He taught me a great deal about what a structural engineer should be. I would also like to express my gratitude to the laboratory members for their assistance at every research phase. I am very thankful to Safumi Saiki for her administrative assistance that has accelerated my efforts over the years.

My research has also been greatly improved thanks to the stimuli from my experience at the Massachusetts Institute of Technology (MIT) in the United States and École Polytechnique Fédérale de Lausanne (EPFL) in Switzerland. My sincere thanks go to Associate Professor Caitlin Mueller in MIT for welcoming me as a visiting student in her laboratory when I was in the master course. Her enthusiasm in connecting architectural design and structural engineering is still a source of my passion for research. During at MIT, I also had the great pleasure to have worked with Yijiang Huang, Jingwen Wang, Dr. Paul Mayencourt, Assistant Professor Nathan Brown, Dr. Pierre Cuvilliers, Dr. Renaud Danhaive, and Dr. Samuel Letellier-Duchesne. I would also like to extend my deepest gratitude to Emeritus Professor Ian Smith and Dr. Gennaro Senatore for giving me an invaluable experience at EPFL. Their constructive advice and insightful suggestions have been influential in shaping my methods. I also wish to thank Dr. Arka Reksowardojo, Dr. Yafeng Wang, and Dr. Slah Drira for their great amount of assistance and patience that cannot be overestimated.

Finally, I cannot forget to thank my family for their love and unwavering trust in me. I am grateful for everything they have done for me and for making my life so wonderful.

