Optimization of Trusses and Frames using Reinforcement Learning

Chi-tathon Kupwiwat

Contents

List of Tables

List of Figures

List of Abbreviations

DDPG	Deep Deterministic Policy Gradient
DEAP	Distributed Evolutionary Algorithms in Python
DoF	Degrees of Freedom
EM	Enumeration Method
GA	Genetic Algorithm
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GSP	Global Sum Pooling
HV	Hypervolume
MADDPG	Multi-agent Deep Deterministic Policy Gradient
MARL	Multi-agent Reinforcement Learning
MDP	Markov Decision Process
ML	Machine Learning
MLP	Multi-layer Perceptron
MSE	Mean Squared Error
NSGA-II	Non-dominated Sorting Genetic Algorithm II
PSO	Particle Swarm Optimization
RL	Reinforcement Learning
SA	Simulated Annealing
SGD	Stochastic Gradient Descent

Chapter 1

Introduction and Background

1.1 Overview of structural optimization

Structural design requires certain levels of optimization. From the perspective of architects/designers, the objective is to obtain structures with high performance while engineers pursue those that minimize the cost under constraints. The structural cost consists of those in design, construction, and maintenance of the structure. Sometimes structural optimization aims to improve both the structural cost and design preference, which can conflict with each other. This research focuses on optimizing the structural performance, cost, and trade-off between cost and design preference which are the main concerns of both architects and structural engineers in the preliminary design phase of truss and frame structures. The reader is encouraged to consult the work by Ohsaki [1] for technical aspects of optimization of these types of structures.

Structural optimization aims to obtain the best design variables that minimize/maximize an objective function under specified constraints [2]. For discrete structures, such as trusses and frames, typically, the design variables are nodal connectivity, cross-sectional properties, and/or nodal locations [3]. The best nodal connectivity, cross-sectional properties, and nodal locations are obtained by solving topology, sizing, and geometry optimization problems, respectively. Objective functions of trusses and frames could be defined using compliance or strain energy (i.e., maximize the stiffness against static loads) under constraints on structural weight, as shown in Refs. [4-6], or the structural weight under constraints on structural response, as shown in Refs. [7-9]. The general structural optimization is formulated as follows:

$$\begin{array}{lll} \text{minimize} & f(\mathbf{x}) & \\ \text{subject to} & g_j(\mathbf{x}) \geq 0, & j = 1,2,\ldots,J; \\ & h_k(\mathbf{x}) = 0, & k = 1,2,\ldots,K; \\ & \mathbf{x} \in [\mathbf{x}_{\mathrm{L}}, \mathbf{x}_{\mathrm{U}}] & \end{array} \tag{1.1}$$

where $\mathbf{x}$, $\mathbf{x}_\mathrm{L}$, and $\mathbf{x}_\mathrm{U}$ are the vector of design variables, the lower bound, and the upper bounds of design variables, respectively. $f(\mathbf{x})$ is the objective function to evaluate the design variable $\mathbf{x}$. $g_j(\mathbf{x})$ and $h_k(\mathbf{x})$ denote inequality and equality constraint functions, respectively. J and K are the numbers of inequality constraints and equality constraints, respectively.

A multi-objective optimization problem has multiple objective functions that can be maximized or minimized, equality and/or inequality constraints, and design variable bounds. The general multi-objective optimization problem, where M objective functions $\mathbf{F}(\mathbf{x}) = (F_1(\mathbf{x}), \dots, F_M(\mathbf{x}))$ are to be minimized is formulated as

$$\begin{array}{lll} \text{minimize} & \mathbf{F}(\mathbf{x}) & \\ \text{subject to} & g_j(\mathbf{x}) \geq 0, & j = 1,2,\dots,J; \\ & h_k(\mathbf{x}) = 0, & k = 1,2,\dots,K; \\ & \mathbf{x} \in [\mathbf{x}_\mathrm{L}, \mathbf{x}_\mathrm{U}] & \end{array} \tag{1.2}$$

In a multi-objective optimization scenario, feasible solutions satisfying bounds and constraints can be projected onto an M-dimensional objective function space, referred to as the objective space. Here, a mapped solution $\mathbf{x}$ is represented as a vector $\mathbf{F}(\mathbf{x}) = (F_1(\mathbf{x}), \dots, F_M(\mathbf{x}))$ in the objective space. It is important to note that in certain cases, objective functions cannot be simultaneously minimized, leading to potential conflicts where improving one objective function may come at the cost of worsening others.

To deal with this conflict, the concept of Pareto optimum [10] comes into play. Pareto optimum is also known as the noninferior solution, non-dominated solution, or compromise solution, and it represents optimal solutions of the multi-objective optimization problem. Given feasible solutions $\mathbf{x}^1$ and $\mathbf{x}^2$ that satisfy all constraints, we consider $\mathbf{x}^1$ to be dominated by $\mathbf{x}^2$ if $F_i(\mathbf{x}^2) \leq F_i(\mathbf{x}^1)$ for all $i \in \{1, \dots, M\}$ and $F_i(\mathbf{x}^2) < F_i(\mathbf{x}^1)$ for at least one $i \in \{1, \dots, M\}$. By applying this definition, a solution $\mathbf{x}^*$ is considered Pareto optimal (or Pareto solution) if no other feasible solutions dominate it. In other words, there are no solutions that are better for all objectives. It is essential to acknowledge that the objective space may contain multiple Pareto optimal solutions, forming a set of non-dominated solutions known as the Pareto front. The Pareto front bounds the feasible region in the objective space. If we specify a reference point in the objective space, we can calculate a generalized volume that covers the Pareto front and the reference point in the M-dimensional space. This volume is known as the *hypervolume* and serves as a metric representing the diversity of the Pareto solutions. Examples of multi-objective optimization of trusses and frames can be found in Refs. [11-13]

Structural optimization approaches can be categorized into two main classes: optimality criteria approaches and search approaches. The optimality criteria approaches focus on finding solutions that meet specific criteria for optimality when the objective function is minimized under the given constraints. Search approaches aim to numerically find solutions starting from an initial design and use different approaches to explore the design variable space, finally minimizing the objective function within the feasible region. A brief overview of common search approaches and their applicability to solving structural optimization problems is as follows:

- Gradient-based approaches depend on local models constructed from the gradient and/or Hessian of the objective and constraint function and the design variables to iteratively improve a solution until a termination criterion is met. This class includes various optimization algorithms, each based on different approaches for determining the descent direction and step size using the local model. Gradient-based methods are well-suited for continuous optimization problems, where evaluating the gradient and/or Hessian matrices is possible. Some common algorithms in this category include gradient descent [14], quasi-Newton [15], sequential quadratic programming [16], and interior-point methods for constrained problems [17]. Examples of such approaches for structural optimizations can be found in Refs. [18-21].
- Stochastic-based approaches rely on pseudo-random number generation during the optimization process to explore the design variable space. Randomness is desirable because it increases the likelihood of finding a global solution and effectively explores the design space by assigning more weight to the regions likely to contain good solutions based on past information. However, excessive randomness can be counterproductive, as it hinders the use of the past information to guide the search. Stochastic-based approaches can also enhance the performance of gradient-based methods and locate approximate global minima for both continuous and discrete mathematical programming problems a limited computational time. Common stochastic search approaches include Stochastic Gradient Descent (SGD) [22] and

Simulated Annealing (SA) [23]. Examples of these approaches for structural optimizations are presented in Refs. [24-27].

- Population-based approaches utilize a population of candidate solutions to explore the design variable space. After each iteration, these candidate solutions are diversified through a stochastic process. By leveraging a large number of candidate solutions for exploration, these methods can reduce the risk of premature termination or getting trapped in a local minimum. Population-based methods are suitable for both continuous and discrete optimization problems, where calculating the objective and constraint functions does not entail significant computational costs. Genetic Algorithm (GA) [28], and Particle Swarm Optimization algorithm (PSO) [29] for single-objective problems, and non-dominated sorting genetic algorithm (NSGA-II) [30] for multi-objective problems fall under these approaches. Refs. [31-34] show examples of these approaches for structural optimizations.

1.2 Machine learning approach and reinforcement learning approaches for structural optimization

In recent years, machine learning (ML) has emerged as a powerful tool in engineering problem-solving. Machine learning is a branch of artificial intelligence (AI) that focuses on developing algorithms and models capable of learning from data and making predictions based on that acquired knowledge. Machine learning can be classified into three categories: supervised learning, unsupervised learning, and reinforcement learning (RL). In supervised learning, the ML model is trained using labeled data, where the inputs and corresponding outputs are provided during the learning process. The model learns to generalize patterns and relationships within the data to make accurate predictions on new, unseen inputs. Unsupervised learning, on the other hand, deals with unlabeled data, aiming to discover inherent structures or patterns in the dataset without specific guidance. Recent advances in ML have gained popularity for applications in engineering fields [35-37], especially for the optimization of structures [38] as presented in the following examples. Mirra and Pugnele [39] used a Variational Autoencoder (VAE) [40] to design spatial structures. Samaniego et al. [41] utilized an ML method to approximate the mechanical response of plates and shells. Zheng et al. [42] and Fuhrimann et al. [43] utilized ML to explore designs of spatial structures. Xie et al. [44] proposed a Bayesian network to obtain the deviation between designed shapes and constructed shapes of the 3D-printed lattice structures. In research on braced frames, Tamura et al. [45] combined ML with SA to optimize brace locations of building frames. Sakaguchi et al. [46] offered methods for extracting important features and converting the features of small frames to those of large frames.

In this research, our main focus lies on RL, a distinct and powerful branch of ML that draws inspiration from optimal control and dynamic programming. In RL, an agent interacts with the environment and learns to make decisions by taking actions to achieve specific goals. The agent receives feedback in the form of rewards or penalties from the environment based on its actions, which helps it to improve its decision-making policy over time. Through this iterative learning process, RL allows the agent to explore various actions and learn how to make optimal decisions to maximize rewards and achieve desirable outcomes in complex and dynamic environments.

In the context of structural optimization, RL holds the promise of revolutionizing the way architects and engineers discover optimal solutions for structural engineering challenges. By leveraging the ability of the agent to interact with the environment and receive feedback, RL can efficiently explore the solution space and find innovative designs for complex engineering structures. Its dynamic nature makes RL particularly well-suited for addressing real-world optimization problems, where deterministic solutions are pursued, constrained by computational time. This research seeks to harness the potential of RL to enhance and expedite the process of finding efficient solutions in the realm of structural optimization.

1.3 Why use the RL approach?

Reinforcement learning can also be viewed as a universal approximation function that takes a structure as input and returns a better structure (Figure 1.1). During the optimization process, the structure is iteratively modified using the agent until the solution of the structural optimization problem is found. A reinforcement learning approach is useful when the structural optimization problem cannot be solved using a gradient-based approach (i.e., discrete optimization problem). The trained RL agent is more deterministic in creating optimum solutions compared to stochastic-based approaches, and also more computationally efficient compared to population-based approaches because only one solution is evaluated at each optimization iteration.

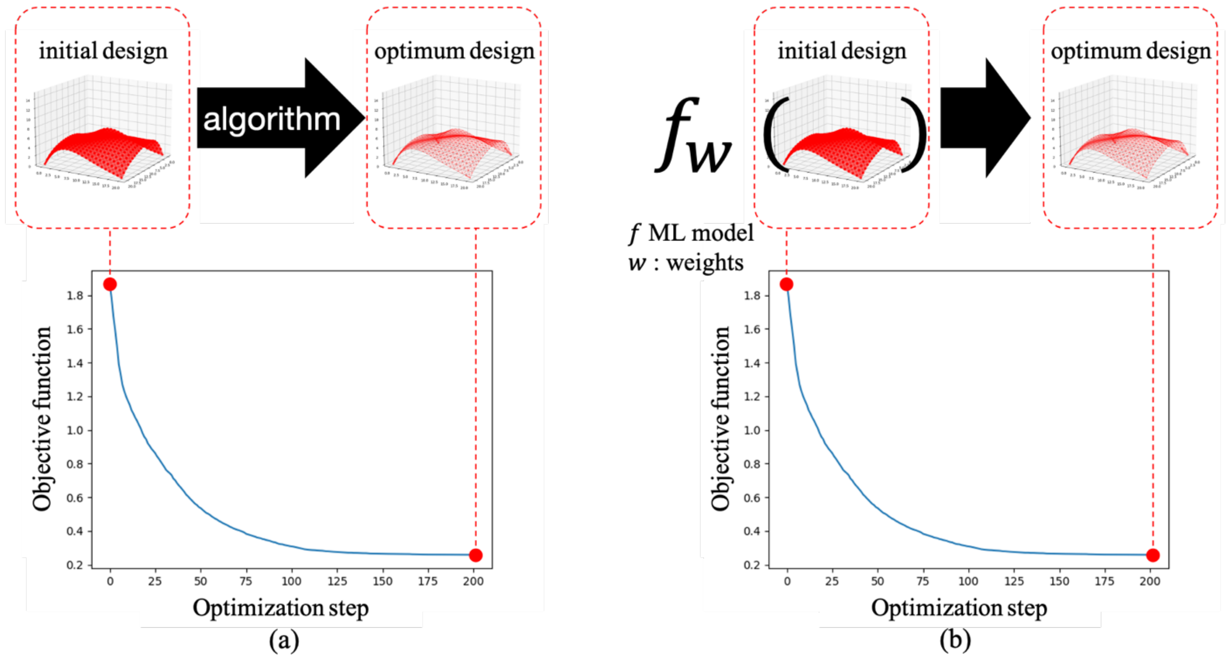


Figure 1.1: RL as a universal approximation function that improves structure; (a) optimization algorithm, (b) RL

One argument about the inferiority of the ML approaches is the high computational cost required for training. To address this problem, this dissertation shows that the RL approach for structural optimization can be trained using some small structures and applied to much larger structures. This fact results in considerable small computational cost during the training because small structures require small computational time to evaluate structural responses using finite element analysis. Since the RL approach is more efficient than to other approaches, it is suitable for real-time structural optimization during the designing phase of buildings. The trained RL agents can also optimize any given structural input. Therefore, the approach is more controllable than stochastic- and population-based approaches, resulting in an approximate optimal solution but close to the initial structure given by architects/structural engineers.

1.4 How is RL utilized?

The potential of RL in Section 1.3 motivates the author to investigate the applicability of RL in structural optimization. Reinforcement learning utilizes state data to do actions (i.e., modifying the structures). Therefore, it is also important to represent the structural data so that the RL agent gains a sufficient understanding of the overall structure. Kupwiwat and Yamamoto [47] applied various RL algorithms to the geometry optimization of small lattice shells using a vector that represents the local information of the structure. Lee et al. [48] and Hoyer et al. [49] also applied a matrix representation for beams and building frames. However, these data representations cannot effectively be utilized for integrating information about the whole structure (i.e., nodal locations, properties of the elements, internal forces, etc.) and solving problems with different structural sizes and configurations. The neural network agents that resort to the representation also struggle when applied to larger structures where local information is not sufficient to represent the overall structure. Hayashi and Ohsaki [50] proposed graph representation to represent the truss information to an RL agent. By transforming the structure into graph data consisting of nodes (vertices) and edges and implementing repetitive graph embedding operations to transmit signals of adjacent nodes and edges for estimating accumulated rewards associated with each action, an RL agent can observe the entire structure and capture the global information of the structural responses and configurations. Note that RL approaches investigated in this literature should not only be applicable to different structures and objective functions but also are robust when applied to structures not utilized for the training.

This study extends the graph representation methods and combines them with RL algorithms that not only can observe, but also modify the whole design variables for solving structural optimization problems. Graph representation methods presented in this study are as follows:

(a) Structural node-wise representation

    In this graph representation, structural nodes and structural elements are represented as graph nodes and graph edges, respectively.

(b) Structural element-wise representation

    In this graph representation method, structural elements are represented as graph nodes. Graph edges are the adjacency of the structural element.

(c) Control point-wise representation

    In this representation, Bézier control points and Bézier grids are represented as graph nodes and graph edges, respectively.

(d) Pareto-wise representation

In this graph representation, non-dominated solutions in the objective space and the proximities of each neighboring solutions pair are represented as graph nodes and graph edges, respectively.

The field of structural design necessitates a deliberate pursuit of optimization. Architects and designers strive for structures that seamlessly blend aesthetics with exceptional structural performance, while structural engineers are focused on achieving cost-efficient solutions within stringent structural constraints. This dissertation leverages RL as a potent computational tool to address these intricate structural optimization challenges. Through RL, we aim to uncover optimal solutions to a range of structural optimization problems, navigating the delicate balance between architectural finesse and economic efficiency. This research explores the transformative potential of RL in enhancing the practice of structural design and engineering in the following structural optimization problems:

(1) Single-objective topology optimization of lattice shell to minimize the strain energy
(2) Single-objective sizing optimization of lattice shell to minimize the structural volume under stress constraint
(3) Single-objective simultaneous topology and sizing optimization of building structure to minimize the structural volume under multiple constraints
(4) Single-objective geometry optimization of lattice shell to minimize the strain energy
(5) Multi-objective sizing optimization of truss structure to minimize structural volume and deformation under stress constraint
(6) Multi-objective simultaneous geometry and sizing optimization of truss structure to minimize structural volume and geometry preference under stress and deformation constraints

1.5 Objectives and contributions

This study aims at developing RL approaches for solving structural optimization problems of trusses and frames. In each chapter, the detail of the optimization problem is formulated, and RL approach is presented, followed by the verification of its performance against benchmark algorithms and its application to solving optimization problems of interest. In particular, we address the following truss and frame structural optimization frameworks:

(1) Topology and sizing optimization problems of lattice shell using structural node-wise representation in problems (1) and (2).
(2) A simultaneous topology and sizing optimization problem of building frame utilizing structural element-wise representation, which is a unification of problems 1 and 2, i.e., problem (3).
(3) Geometry optimization problem of lattice shell using multi-information representations of structural node-wise and control point-wise representations, i.e., problem (4).
(4) Multi-objective problem of truss with single discrete variables using multi-information representations of structural node-wise and Pareto-wise representations in problem (5).
(5) Multi-objective problem of truss with mixed discrete-continuous variables utilizing multi-information representations of structural node-wise and Pareto-wise representations in problem (6).

The main contributions of this study to the structural engineering and optimization communities are twofold:

(1) Effective RL frameworks for structural optimization

    These frameworks can be extended for solving diverse structural optimization scenarios, such as optimizing building designs, bridges, or other complex structures, each with specific requirements and performance criteria. By adopting RL, engineers can transcend the limitations of traditional optimization methods and effectively address discrete and continuous optimization problems that may pose significant challenges to conventional gradient-based approaches.
(2) Useful structural optimization algorithms that are efficient and reliable

    The deterministic nature of RL, presented in this dissertation, empowers it to create optimal solutions with confidence, outperforming stochastic-based methods in producing high-quality outcomes. Furthermore, the computational efficiency of RL is evident when compared to population-based approaches. The evaluation of a single solution in each optimization iteration significantly reduces computational costs, streamlining the optimization process and enabling real-time applications during the designing phase of buildings and other structures. This level of control and efficiency renders RL-based structural optimization approaches more reliable, empowering architects and structural engineers to converge on highly optimized designs while preserving key elements of the original concepts.

1.6 Dissertation outline

The remaining chapters of this dissertation are outlined as follows:

Chapter 2 presents brief introductions to RL approaches. The basic concepts of ML and data structures are also included in this chapter. Two RL algorithms including Deep Deterministic Policy Gradient (DDPG) and Multi-Agent Deep Deterministic Policy Gradient (MADDPG) are presented. The chapter also introduces the computational parts of RL including the neural networks, Graph Convolutional Networks (GCN), and Graph Attention Networks (GAT), respectively.

Chapter 3 exhibits the applicability of the proposed RL frameworks for solving two single objective optimization problems of lattice shells. The first problem is the topology optimization of shell structures where the brace directions and strain energy are design variables and the objective function, respectively. The second problem is the sizing optimization of lattice shells where the sectional area of the structural element, structural volume, and internal stress ratio are design variable, objective function, and constraint, respectively. Structural data is represented using the node-wise method. This chapter utilizes the graph representation method (a) from Section 1.4, for creating optimization framework (1) in Section 1.5.

Chapter 4 shows a case of RL frameworks for solving simultaneous topology and sizing optimization of building frames under seismic loads. Unlike the representation in Chapter 3, the structural data is modeled by an element-wise representation. In this chapter, graph representation method (b) from Section 1.4 is utilized for creating optimization framework (2) in Section 1.5.

Chapter 5 initiates an RL framework for geometry optimization of lattice shells whose nodal locations are determined by Bézier control points. The design variables and objective function are the locations of the control points and strain energy, respectively. The framework in this chapter unifies data from both information from the structures and Bézier control points. This chapter combines graph representation methods (a) and (c) from Section 1.4, for creating optimization framework (3) in Section 1.5.

Chapter 6 presents a novel multi-agent reinforcement learning (MARL) framework for solving multi-objective optimization of planar trusses. The proposed framework is verified utilizing a mathematical multi-objective optimization problem and a discrete variable multi-objective optimization of truss which has known and approximated Pareto solutions, respectively. After the verifications, the proposed method is applied to a mixed discrete-continuous variable multi-objective optimization of the trusses where Pareto solutions change depending on predetermined settings. This chapter combines graph representation methods (a) and (d) from Section 1.4, for creating optimization frameworks (5) and (6) in Section 1.5.

Chapter 7 summarizes all presented frameworks and discusses the future research direction.

Figure 1.2 illustrates the relations and dependencies between each chapter in this dissertation. In this Figure, the arrows and dashed arrows indicate dependencies and weak dependencies between the chapters, respectively.

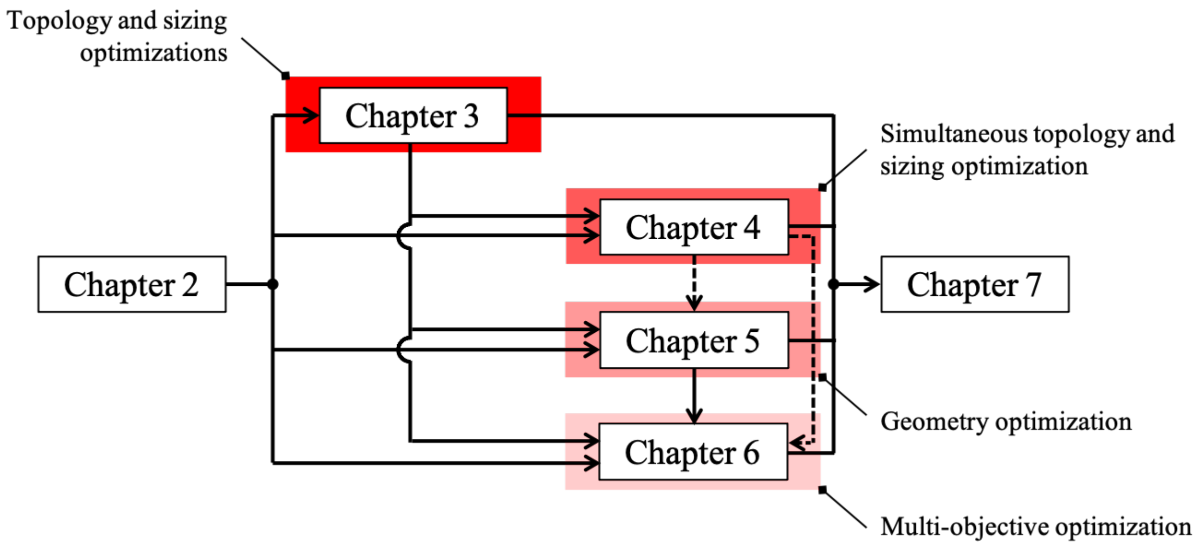

Topology and sizing optimizations
Chapter 3
Simultaneous topology and sizing optimization
Chapter 4
Chapter 2
Chapter 7
Chapter 5
Geometry optimization
Chapter 6
Multi-objective optimization

Figure 1.2: Relations and dependencies between chapters

1.7 Published works included in the dissertation

This dissertation is a collection of several published papers by the author and co-authors. The detailed information of these papers are organized in this dissertation are organized as follows:

Chapter 3

Kupwiwat C, Hayashi K, Ohsaki M (2022) Deep deterministic policy gradient and graph convolutional network for bracing direction optimization of grid shells. Frontiers in Built Environment, Vol.8(899072), no pages.

Kupwiwat C, Hayashi K, Ohsaki M (2023) Sizing optimization of free-form lattice shells using deep deterministic policy gradient and graph convolutional networks. *In.* Proceeding of International Association for Shell and Spatial Structures 2023. pp. 1458−1468.

Chapter 4

Kupwiwat C, Iwagoe Y, Hayashi K, Ohsaki M (2023) Deep deterministic policy gradient and graph convolutional network for topology optimization of braced steel frames. Journal of Structural Engineering B. Architectural Institute of Japan. Vol.69B, pp. 129−139.

Chapter 5

Kupwiwat C, Hayashi K, Ohsaki M (2023) Deep deterministic policy gradient and graph attention network for geometry optimization of latticed shells. Applied Intelligence. Vol.53, pp. 19809−19826.

Chapter 6

Kupwiwat C, Hayashi K, Ohsaki M (2024) Multi-objective optimization of truss structure using multi-agent reinforcement learning and graph representation. Engineering Applications of Artificial Intelligence. Vol.129(107594), no pages.

Chapter 2

Reinforcement learning approaches: A brief review

2.1 Introduction

This chapter briefly reviews the basic concepts of ML and RL together with their applications in structural engineering domains. The dissertation utilizes two RL algorithms including DDPG and MADDPG. Both of which utilize value function approximation, policy gradient method, and actor-critic method. All key concepts and the computational aspects of these methods and algorithms are reviewed in this chapter. Variables in structural problems are represented by vector and graph data which are also introduced in this chapter. RL agents observe these representations to find optimal solutions. At the end of this chapter, the approximation functions or neural networks utilized for making agents in this research are explained.

2.2 Basic concepts of ML

Machine learning is a collection of algorithms that have the ability to learn to do tasks, such as classification and regression, from data [51]. In the classification task, the computational part, denoted as a *model*, learns to classify input data $\mathbf{x}$ into a class utilizing the vector of outputs or *labels*. In the regression task, the model learns to approximate the output value $\boldsymbol{y}$ from input data $\mathbf{x}$, given that there is a relationship $f_{\mathbf{w}}(\mathbf{x}) = \boldsymbol{y}$. Let $f_{\mathbf{w}}(\mathbf{x})$ be a parameterized machine learning model that takes input data $\mathbf{x}$. The output of the model or the prediction $\hat{\boldsymbol{y}}$ is as follows:

$$f_{\mathbf{w}}(\mathbf{x}) = \hat{\boldsymbol{y}} \tag{2.1}$$

Suppose there is a vector of optimum parameterized weights $\mathbf{w}^*$ that minimizes a difference measuring value between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ denoted as loss or error $\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})$. The aim of machine learning algorithms is to minimize this loss, and this task is formulated as

$$\min \mathcal{L}(\boldsymbol{y}, f_{\mathbf{w}}(\mathbf{x}))|(\mathbf{x}, \boldsymbol{y}) \tag{2.2}$$

where $(\mathbf{x}, \boldsymbol{y})$ in Eq. (2.2) is the *dataset* or *training data* that is utilized for training the model.

Structural optimization in structural engineering is a critical endeavor aimed at designing efficient structures capable of withstanding various loads while minimizing material usage and costs. Traditional approaches to structural optimization often involve intricate mathematical models and iterations, which can be time-consuming and limited in their ability to explore the extensive design space. However, the advent of ML techniques has ushered in a new era, offering efficient and innovative avenues for optimizing truss and frame structures.

One of the notable applications of ML in structural optimization is the development of surrogate models, which can be based on various ML algorithms, and trained using existing structural data to approximate the complex relationships between design parameters, external loads, and performance metrics. By utilizing these surrogate models, engineers can significantly expedite the optimization process. These models enable predictions of the performance of numerous design configurations, allowing for rapid exploration of the design space and the identification of optimal solutions as shown in the research by Mai [52] which utilizes surrogate models for optimizing truss structures. This approach can also be combined with other optimization approaches and enables broad spectrums of the design method, as shown in the works by Li [53] and Luo [54] which combines the surrogate models with the bi-directional evolutionary structure optimization method and heuristic optimization method, respectively.

Another promising application of ML in structural optimization is generative design. Generative design harnesses ML algorithms such as Variational Autoencoder [40] or Convolutional Autoencoder [55] to explore diverse design possibilities. Engineers can specify design constraints and objectives, and generative design tools can automatically generate numerous design proposals. These proposals can then undergo further refinement and optimization based on engineering preferences and project-specific requirements. Examples of ML in the generative design of structure can be found in recent works by Palmeri et al. [56], Mirra et al. [39], Zheng et al. [42], and Fuhrimann et al. [43] which explores design option utilizing ML, studies AI-generative design in the shell structures, investigates the relationship between form and force in shells using ML, and utilized ML for form-finding tasks, respectively.

Additionally, ML can be employed for structure health monitoring and damage detection in structural engineering. The ML model can be formulated to predict the conditions of structures, detect anomalies, and forecast potential failures before they occur. This capability holds particular significance for critical infrastructure like bridges and buildings, where early detection of structural issues can prevent catastrophic failures, ensuring both safety and cost savings. Examples of applications of ML in the area being researched by Cheung et al. [57], Ng et al. [58], Hwang et al. [59], and Tibaduiza et al. [60] which utilize autoregressive ML models [61] for the structure health monitoring of bridge structures, Bayesian ML approach [62] for the structure health monitoring of aluminium beams, ML model approach for damage detection (i.e., pattern recognition) of building structures, and Principal Component Analysis ML model [63,64] for damage detection of aircraft shell structure, respectively.

In conclusion, ML offers promising applications in civil and mechanical engineering. It enables the creation of surrogate models, and design exploration, and contributes to predictive maintenance for structural health monitoring. As ML techniques continue to advance, structural optimization stands to benefit from faster, more efficient approaches to designing cost-effective and resilient structures.

2.3 Basic concepts of RL

Reinforcement learning is a type of ML that trains a model or an *agent* to perform *actions* in an environment using reward signals. The core of RL is learning what to do in given situations or to map situations to actions using functions (i.e., classification of an action among $\boldsymbol{y}$ sets of action given a situation observed as $\mathbf{x}$). This learning is guided by numerical reward signal and the objective of the agents is to maximize this reward. Unlike other machine learning algorithms, RL has no initial training dataset. Thus, the algorithm has to collect the dataset for optimizing its models during the training.

An RL algorithm consists of three main elements: a *policy* that determines the behavior of the agent, a *reward signal* that defines how good/bad the behavior is, according to the policy, and a *value function* that predicts how the agent performs based on the policy [65]. The interaction of an agent and the environment is formulated using a Markov Decision Process (MDP) [66,67] as follows:

In a discrete time step t:

The agent receives a representation of the environment's *state* S_t (situations).

The agent performs actions A_t.

The agent receives quantitative reward R_{t+1} and next state S_{t+1} from the environment.

The diagram of the MDP can be represented as shown in Figure 2.1. This cycle keeps on until a terminal state is reached.

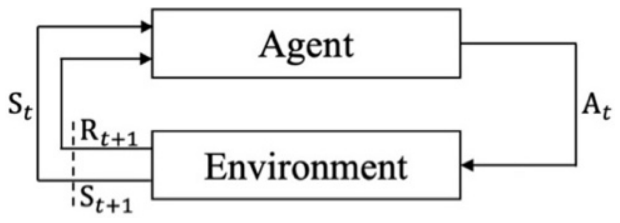


Figure 2.1: Diagram of the MDP

Considering an MDP where the agent obtains a reward R_{t+1} in a step t, the agent aims to maximize the accumulated rewards G_t computed as

$$\mathrm{G}_t = \mathrm{R}_{t+1} + \gamma \mathrm{R}_{t+2} + \gamma^2 \mathrm{R}_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k \mathrm{R}_{t+\mathrm{k}+1} \tag{2.3}$$

where $0 \leq \gamma \leq 1$ is the discount rate, normally set as 0.99.

Given that a policy $\mathrm{q}_{\boldsymbol{\pi}}$ is a mapping function from S_t to an action A_t, we can quantitatively represent how good it is for an agent to do A_t in a given state S_t in a form of action-value function using state-action pairs (s, a) under parameter $\boldsymbol{\pi}$. This function aims at computing the expected reward (i.e., *return*), defined as

$$\mathrm{q}_{\boldsymbol{\pi}}(s, a) \doteq \mathbb{E}_{\boldsymbol{\pi}}[\mathrm{G}_t | \mathrm{S}_t = s, \mathrm{A}_t = a] = \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_{k=0}^{\infty} \gamma^k \mathrm{R}_{t+\mathrm{k}+1} | \mathrm{S}_t = s, \mathrm{A}_t = a\right] \tag{2.4}$$

where $\mathbb{E}_{\pi}[\cdot]$ is the expected value.

Reinforcement learning algorithms aim to find a policy that yields the highest expected reward. In theory, there is at least a policy with parameter $\boldsymbol{\pi}_*$ that is as good as or better than any other policies. These *optimal policies* have the same optimal action-value function denoted as q_* and defined as

$$\mathrm{q}_*(s,a) \doteq \max\left(\mathrm{q}_{\boldsymbol{\pi}}(s,a)\right) \tag{2.5}$$

Thus, the policy that yields the highest expected return of an optimal action-value function can also be defined in a Bellman optimality equation for q_* [68,69] as

$$\mathrm{q}_*(s,a) \doteq \mathbb{E}[\mathrm{R}_{t+1} + \gamma \max\left(\mathrm{q}_*(\mathrm{S}_{t+1}, \mathrm{A}_{t+1})\right) | \mathrm{S}_t = s, \mathrm{A}_t = a] \tag{2.6}$$

Note that there are various methods for the RL algorithm to utilize the optimal action-value function. For example, Q-learning [70] utilizes the optimal action-value function to evaluate all actions in a given state s. From this evaluation, the agent selects the action that yields the maximum expected reward (i.e., *greedy* policy). In policy gradient method [71], the optimal action-value function aims to approximate the expected reward from the given state s and actions from another parameterized policy function as explained in the next section.

2.3.1 Policy gradient method

Policy gradient methods learn to modify a parametrized policy that chooses actions in a given state to maximize the return. The parametrized policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}$ has a policy parameter $\boldsymbol{\theta}_t$ in a timestep t. The parametrized policy is optimized to maximize a scalar performance measurement $\mathrm{J}(\boldsymbol{\theta})$. Policy parameters can be updated to maximize the scalar performance measure by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla \mathrm{J}(\boldsymbol{\theta}_t) \tag{2.6}$$

where α is the learning rate.

According to the policy gradient theorem [71], the scalar performance $\mathrm{J}(\theta)$ of a policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}$ can be obtained from summarizing the evaluation of the policy, parameterized by policy parameter θ, that do action a in the given state s ($\boldsymbol{\pi}_{\boldsymbol{\theta}}(a|s)$) using the value function $\mathrm{q}(s,a)$ defined as

$$\mathrm{J}(\theta) \doteq \mathrm{v}_{\boldsymbol{\pi}_{\boldsymbol{\theta}}}(s) = \sum \boldsymbol{\pi}_{\boldsymbol{\theta}}(a|s)\,\mathrm{q}(s,a) \tag{2.7}$$

where $\mathrm{v}_{\pi_\theta}(s)$ is the value of the state computed using a policy function $\boldsymbol{\pi}_\theta(a|s)$ and action-value function $\mathrm{q}(s,a)$. The gradient of $\mathrm{v}_{\pi_\theta}(s)$ can be defined as

$$\nabla_\theta \mathrm{v}_{\boldsymbol{\pi}_\theta}(s) = \nabla_\theta \left[\sum \boldsymbol{\pi}_{\boldsymbol{\theta}}(a|s)\,\mathrm{q}(s,a)\right] \tag{2.8}$$

Utilizing Eqs. (2.7) and (2.8), the gradient of $\mathrm{J}(\theta)$ can be defined as

$$\nabla_\theta \mathrm{J}(\theta) = \nabla_\theta \mathrm{v}_{\boldsymbol{\pi}_\theta}(s) = \nabla_\theta \left[\sum \boldsymbol{\pi}_\theta(a|s)\,\mathrm{q}(s,a)\right] \tag{2.9}$$

or, in the policy gradient theorem, as

$$\nabla_\theta \mathrm{J}(\theta) = \mathbb{E}[\nabla_\theta \log \boldsymbol{\pi}_\theta(a|s)\mathrm{q}(s,a)] \tag{2.10}$$

Eq. (2.10) can be extended to a deterministic policy [72] which uses policy distribution over states $\boldsymbol{\pi}_\theta(s) = \sum \boldsymbol{\pi}_\theta(a|s)$ defined as

$$\nabla_\theta \mathrm{J}(\theta) = \mathbb{E}[\nabla_\theta \mathrm{q}(s, \boldsymbol{\pi}_\theta(s))] \tag{2.11}$$

or, using the chain rule, as

$$\nabla_\theta \mathrm{J}(\theta) = \mathbb{E}\left[\nabla_\theta \boldsymbol{\pi}_\theta(s) \nabla_a \mathrm{q}(s,a)|_{a=\boldsymbol{\pi}_\theta(s)}\right] \tag{2.12}$$

Note that $\boldsymbol{\pi}_\theta(s)$ can be represented using approximation functions such as neural networks.

2.3.2 Actor-Critic method

Actor-Critic [73] is a variance of the policy gradient method. In Actor-Critic, a neural network is utilized for representing a policy $\boldsymbol{\pi}_\theta(s)$ with weights and bias as policy parameter θ. This neural network is called an *actor*. Another important part is the *critic*, a neural network that predicts the return from s and actions from $\boldsymbol{\pi}_\theta(s)$. In general, the approximation function has parameterized weight w and also needs to be trained to predict the return. Let $\mathrm{q}_w(s, a)$ be the critic. Eq. (2.12) becomes

$$\nabla_\theta \mathrm{J}(\theta) = \mathbb{E}\left[\nabla_\theta \boldsymbol{\pi}_\theta(s) \nabla_a \mathrm{q}_w(s, a)|_{a=\boldsymbol{\pi}_\theta(s)}\right] \tag{2.13}$$

2.3.3 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient [72] is a type of actor-critic algorithm, characterized by the ability of the agent to do multiple actions in an MDP which can reduce the number of optimization steps and computational cost in structural optimization. The DDPG agent utilizes a policy function $\boldsymbol{\pi}_{\theta_1}$(*actor network*) and a value function Q_{θ_2} (*critic network*) parametrized by trainable parameters θ_1 and θ_2, respectively, and defined as

$$\boldsymbol{\pi}_{\theta_1}(\mathrm{S}_t) = \mathrm{P}(\mathrm{A}_t^i|\mathrm{S}_t) \tag{2.14}$$

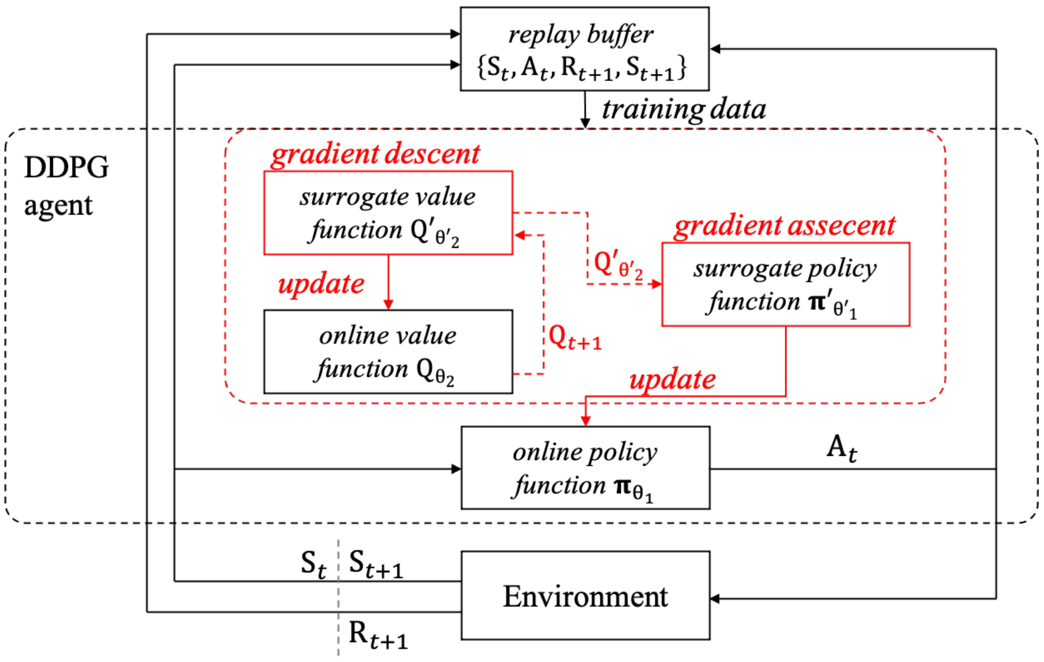
$$\mathrm{Q}_{\theta_2}\left(\mathrm{S}_t, \boldsymbol{\pi}_{\theta_1}(\mathrm{S}_t)\right) = \sum_{v=1}^{\infty} \gamma^{v-1} \mathrm{R}_{t+v} \tag{2.15}$$

The objective of the policy function is to obtain high rewards by determining the probability $\mathrm{P}(\mathrm{A}_t^i|\mathrm{S}_t)$ of taking an action i in a state S_t. The agent interacts with the environment during the training phase to gather training data $\{\mathrm{S}_t, \mathrm{A}_t, \mathrm{R}_{t+1}, \mathrm{S}_{t+1}\}$ and store them in a *replay buffer*. These training data are necessary for training both policy and value functions, respectively, to improve the behavior of the agent, in order to obtain high reward, and increase the accuracy of accumulated reward prediction. However, simultaneously collecting training data and training functions makes learning unstable because the functions could become diverged [72]. To stabilize the training process, a *tau updating* technique was developed by Haarnoja et al. [74]. This technique trains $\boldsymbol{\pi}'_{\theta'_1}$ and $Q'_{\theta'_2}$ as surrogate policy function and value function, respectively, in place of $\boldsymbol{\pi}_{\theta_1}$ and Q_{θ_2}. Trainable parameters of these surrogate functions are gradually updated into those of policy and value functions that interact with the environment to collect the training data, named as *online* policy and value functions (i.e., $\boldsymbol{\pi}_{\theta_1}$ and Q_{θ_2}). The trainable parameters are also updated at a constant interval utilizing a small amount of updating weight determined by $\tau(\ll 1)$ to stabilize the learning process. The details of the DDPG algorithm are explained as follows:

DDPG algorithm:

1. Sample n_{batch} data $\{S_t, A_t, R_{t+1}, S_{t+1}\}$ from the replay buffer and turn them into training dataset $\{\mathbf{S}_t, \mathbf{A}_t, \mathbf{R}_{t+1}, \mathbf{S}_{t+1}\}$.
2. Make the following parameter updates:
   $\boldsymbol{\pi}_{\theta'_1}(\mathbf{S}_t) = \widehat{\mathbf{A}}_t$
   $\boldsymbol{\pi}_{\theta_1}(\mathbf{S}_{t+1}) = \widehat{\mathbf{A}}_{t+1}$
   $Q'_{\theta'_2}(\mathbf{S}_t, \mathbf{A}_t) = \widehat{\mathbf{Q}}_t$
   $Q_{\theta_2}(\mathbf{S}_{t+1}, \widehat{\mathbf{A}}_{t+1}) = \mathbf{Q}_{t+1}$
   $\nabla Q'_{\theta'_2} = \nabla_{\theta'_2} Q'_{\theta'_2}(\mathbf{S}_t, \mathbf{A}_t) \nabla_{\widehat{\mathbf{Q}}_t} \mathcal{L}(\mathbf{R}_{t+1} + \mathbf{Q}_{t+1}, \widehat{\mathbf{Q}}_t)$; *gradient descent*
   $\nabla J'_{\theta'_1} = -\mathbb{E}\left[\nabla_{\theta'_1} \boldsymbol{\pi}'_{\theta'_1}(\mathbf{S}_t) \nabla_{\widehat{\mathbf{A}}_t} Q'_{\theta'_2}(\mathbf{S}_t, \widehat{\mathbf{A}}_t)|_{\widehat{\mathbf{A}}_t = \boldsymbol{\pi}'_{\theta'_1}(\mathbf{S}_t)}\right]$; *gradient ascent*
   By utilizing $\nabla Q'_{\theta'_2}$, update $\boldsymbol{\theta}'_2$ in $Q'_{\theta'_2}$
   By utilizing $\nabla J'_{\theta'_1}$, update $\boldsymbol{\theta}'_1$ in $\boldsymbol{\pi}'_{\theta'_1}$
   Upon reaching the tau update interval:
      $\boldsymbol{\theta}_1 = (1-\tau)\boldsymbol{\theta}_1 + \tau\boldsymbol{\theta}'_1$
      $\boldsymbol{\theta}_2 = (1-\tau)\boldsymbol{\theta}_2 + \tau\boldsymbol{\theta}'_2$

Among a number of optimization algorithms for surrogate functions such as SGD [75,76] and Adam [77], Adam is utilized for the parameter updating of $\boldsymbol{\theta}'_1$ and $\boldsymbol{\theta}'_2$. By directly introducing Ornstein-Uhlenbeck noise [78] into the output value of the policy function, the exploration of the agent is triggered [72]. Figure 2.2 illustrates the MDP which has a DDPG agent interacting with the environment to collect training data in the replay buffer, which are utilized in the training and updating process of the functions of DDPG described in this section.



replay buffer
$\{S_t, A_t, R_{t+1}, S_{t+1}\}$
training data
DDPG agent
gradient descent
surrogate value function $Q'_{\theta'_2}$
update
online value function Q_{θ_2}
$Q'_{\theta'_2}$
Q_{t+1}
gradient assecent
surrogate policy function $\pi'_{\theta'_1}$
update
online policy function π_{θ_1}
A_t
S_t
S_{t+1}
R_{t+1}
Environment

Figure 2.2: Diagram of the MDP with the DDPG agent

When applied to single objective structural optimization problems, the RL agent can be trained to modify the design variables of the structure to obtain a desirable structural configuration measured by the objective function. After training, the agent can also be deployed to solve similar problems as presented by Gambardella et al. [79] and Huynh et al. [80].

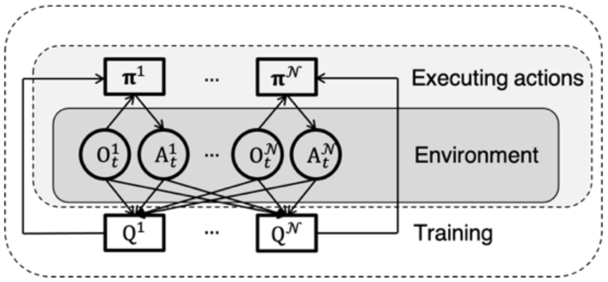
2.3.4 Multi-agent DDPG

MARL is an extension of RL. Unlike RL, MARL formulates interactions of multiple agents in an environment using Markov games [81], instead of the MDP. A Markov game that has $\mathcal{N}$ agents is formulated by a set of states $\mathcal{S}$ indicating the possible configuration of the environment by all agents, a set of actions $A^1, \ldots, A^{\mathcal{N}}$, and a set of observations of the environment $O^1, \ldots, O^{\mathcal{N}}$. Note that for a discrete time step t the state transition is a product of the current state S_t and all actions in that time step, i.e., $S_t \times A_t^1 \times \ldots \times A_t^{\mathcal{N}} \rightarrow S_{t+1}$. Each agent u receives its observation, does actions, and obtains its reward as $\mathcal{S} \times A_t^u \rightarrow R_{t+1}^u$. Similar to a standard RL, all agents in MARL aim to maximize their rewards. However, since the actions of other agents hinder the agent from learning effectively, some MARL algorithms, proposed by Tesauro [82] and Foerster et al. [83], address the formulation so that agents can observe the actions of other agents as well. Unlike a conventional RL, MARL for structural optimization is still a frontier research field. A few examples of MARL for optimization are investigated by Nasir et al. [84] and Yu et al. [85].

Multi-agent Deep Deterministic Policy Gradient (MADDPG) [86] is a type of MARL algorithm that has multiple agents, each of which has a parametrized policy function $\boldsymbol{\pi}^u$ (*actor network*) and a parametrized value function Q^u (*critic network*). MADDPG utilizes the multi-agent decentralized actor and centralized critic approach [83]. In this approach, each agent executes an action independently, but their value functions are trained collectively using actions from all agents, as illustrated in Figure 2.3. Let u be an index of the agent while $\mathbf{P}^u(\mathrm{A}_t^u|\mathrm{O}_t^u)$ is the probability of making action A_t^u using observation O_t^u, the policy and value functions of agent u are given as follows:

$$\boldsymbol{\pi}^u(\mathrm{O}_t^u) = \mathbf{P}^u(\mathrm{A}_t^u|\mathrm{O}_t^u) \tag{2.16}$$

$$\mathrm{Q}^u\left(\mathrm{O}_t^1,\ldots,\mathrm{O}_t^{\mathcal{N}},\boldsymbol{\pi}^1(\mathrm{O}_t^1),\ldots,\boldsymbol{\pi}^{\mathcal{N}}\left(\mathrm{O}_t^{\mathcal{N}}\right)\right) = \sum_{v=1}^{\infty}\gamma^{v-1}\mathrm{R}_{t+v}^u \tag{2.17}$$



$\boldsymbol{\pi}^1$
...
$\boldsymbol{\pi}^{\mathcal{N}}$
Executing actions
O_t^1
A_t^1
$\mathrm{O}_t^{\mathcal{N}}$
$\mathrm{A}_t^{\mathcal{N}}$
Environment
Q^1
$\mathrm{Q}^{\mathcal{N}}$
Training

Figure 2.3: Multi-agent deep deterministic gradient

Similar to DDPG, no training data is available at the beginning of the MARL, all agents interact with the environment during the training phase to collect training data $\{\mathrm{O}_t^u,\mathrm{A}_t^u,\mathrm{R}_{t+1}^u,\mathrm{O}_{t+1}^u; u \in \{1,\ldots,\mathcal{N}\}\}$ and store them in the replay buffer. Each agent utilizes this training data to improve its behavior to obtain high rewards and to predict the rewards more accurately. This research utilizes a tau updating similar to the DDPG algorithm.

During training, the policy and value functions process dataset $\{\mathbf{O}_t^u,\mathbf{A}_t^u,\mathbf{R}_{t+1}^u,\mathbf{O}_{t+1}^u\}$ from the replay buffer to predict action $\widehat{\mathbf{A}}_t^u$ and expected reward $\widehat{\mathbf{Q}}_t^u$, respectively. $\widehat{\mathbf{Q}}_t^u$ is utilized for computing a loss function for training the value function. The trainable parameters of the value function are modified (i.e., trained) using gradients of the loss function and the parameters, denoted as $\nabla \mathrm{Q}'^u$, utilizing the chain rule. Note that the trainable parameters of the value function are adjusted to minimize the loss function, using a *gradient descent* algorithm. In each agent, the policy function is trained using the gradient of $\widehat{\mathbf{Q}}_t^u$ with respect to its trainable parameters to maximize the expected reward using *gradient ascent*, denoted as $\nabla \mathrm{J}'^u$. The MADDPG algorithm is summarized as follows:

<u>MADDPG algorithm</u>

```
1. Sample $n_{\text{batch}}$ sets of data $\{\mathrm{O}_t^u, \mathrm{A}_t^u, \mathrm{R}_{t+1}^u, \mathrm{O}_{t+1}^u\}$ from the replay buffer and change them into
   training dataset $\{\mathbf{O}_t^u, \mathbf{A}_t^u, \mathbf{R}_{t+1}^u, \mathbf{O}_{t+1}^u\}$ where $u \in \{1, \dots, \mathcal{N}\}$.
2. Train policy and value functions of the agents, where $u \in \{1, \dots, \mathcal{N}\}$
      $\boldsymbol{\pi}'^u(\mathbf{O}_t^u) = \widehat{\mathbf{A}}_t^u$
      $\boldsymbol{\pi}^u(\mathbf{O}_{t+1}^u) = \widehat{\mathbf{A}}_{t+1}^u$
      $\mathrm{Q}'^u(\mathbf{O}_t^1, \dots, \mathbf{O}_t^{\mathcal{N}}, \widehat{\mathbf{A}}_t^1, \dots, \widehat{\mathbf{A}}_t^{\mathcal{N}}) = \widehat{\mathbf{Q}}_t^u$
      $\mathrm{Q}^u(\mathbf{O}_{t+1}^1, \dots, \mathbf{O}_{t+1}^{\mathcal{N}}, \widehat{\mathbf{A}}_{t+1}^1, \dots, \widehat{\mathbf{A}}_{t+1}^{\mathcal{N}}) = \mathbf{Q}_{t+1}^u$
      $\nabla \mathrm{Q}'^u = \nabla \mathrm{Q}'^u(\mathbf{O}_t^1, \dots, \mathbf{O}_t^{\mathcal{N}}, \widehat{\mathbf{A}}_t^1, \dots, \widehat{\mathbf{A}}_t^{\mathcal{N}}) \nabla_{\widehat{\mathbf{Q}}_t^u} \mathcal{L}(\mathbf{R}_{t+1}^u + \mathbf{Q}_{t+1}^u, \widehat{\mathbf{Q}}_t^u)$ ; *gradient descent*
      $\nabla \mathrm{J}'^u = -\mathbb{E}\left[\nabla \boldsymbol{\pi}'^u(\mathbf{O}_t^u) \nabla_{\widehat{\mathbf{A}}_t^u} \mathrm{Q}'^u(\mathbf{O}_t^1, \dots, \mathbf{O}_t^{\mathcal{N}}, \widehat{\mathbf{A}}_t^1, \dots, \widehat{\mathbf{A}}_t^{\mathcal{N}})|_{\widehat{\mathbf{A}}_t^u = \boldsymbol{\pi}'^u(\mathbf{O}_t^u)}\right]$ ; *gradient ascent*
      Update trainable parameters in $\mathrm{Q}'^u$ using $\nabla \mathrm{Q}'^u$
      Update trainable parameters in $\boldsymbol{\pi}'^u$ using $\nabla \mathrm{J}'^u$
      If reach the tau update interval:
         $\boldsymbol{\theta}_1^u = (1 - \tau)\boldsymbol{\theta}_1^u + \tau \boldsymbol{\theta}'^u_1$
         $\boldsymbol{\theta}_2^u = (1 - \tau)\boldsymbol{\theta}_2^u + \tau \boldsymbol{\theta}'^u_2$
```

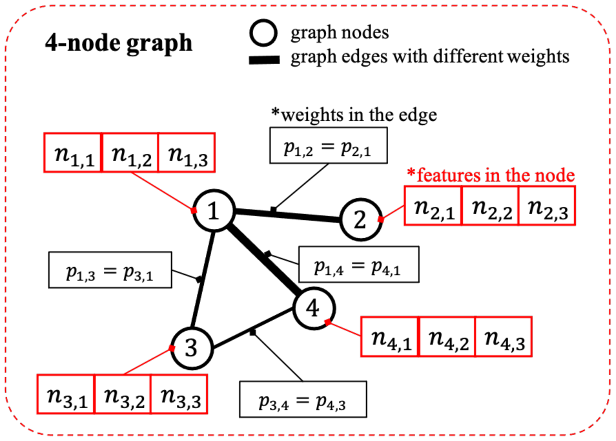
Similar to DDPG, Adam is utilized for the parameter updating of $\boldsymbol{\theta}'_1$ and $\boldsymbol{\theta}'_2$. Explorations of all agents are activated by adding small Ornstein-Uhlenbeck noise to the outputs of all policy functions when collecting training data.

The application of RL to the structural optimization of trusses and frames represents a new approach in engineering. Reinforcement learning techniques offer the potential to enhance the efficiency and adaptability of these structures, optimizing their design in response to varying conditions. Examples of applications of RL in this area are work by Huynh et al. [87], Hayashi et al. [88], Sahachaisaree et al. [89], and Kupwiwat et al. [90] which combines Q-learning and evolutionary algorithm for the optimization of truss, proposes combined RL-SA and RL-PSO methods for plane frames optimization, proposes truss design method using RL, and investigates the applicability of MADDPG for the geometry optimization of lattice shell, respectively.

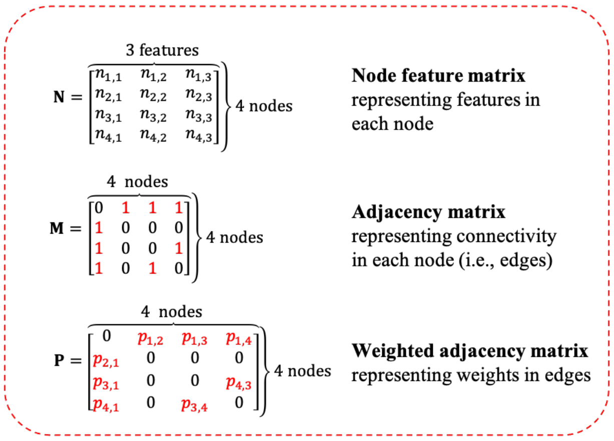
2.4 Graph data structure and representation

The success of ML requires properly representing the data to train the model effectively. There are multiple methods to represent data depending on their types such as a vector and a two-dimensional matrix for tabular and image data, respectively. In applications of ML to structural optimization problems, there are multiple approaches to represent data of building structures. Hayashi and Ohsaki [50] showed that it is effective to represent a truss structure as a graph and then process the graph data using fixed-size matrices. This method allows the RL agent to capture the properties of the whole structure and modify the topology of trusses with different connectivity.

A graph is a type of data structure that has nodes connected by edges. Note that nodes contain data indicated as *features* and edges contain data about the connectivity, weights of the connectivity, or relationship between nodes. This connectivity can be either directed or undirected indicating the directions or no direction of these edges, respectively. A graph, having n nodes and g features in each node, can be represented using a node feature matrix $\mathbf{N} \in \mathbb{R}^{n \times g}$. An adjacency matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$, a weighted adjacency matrix $\mathbf{P} \in \mathbb{R}^{n \times n}$, and a degree matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$ represent the data of nodal connections, weights of nodal connections, and the number of connections at each node, respectively. Figure 2.4 illustrates undirected graph data (a) and its representation (b). Note that all research in this dissertation utilizes only undirected graph data which does not indicate the directions of the edge connections, because all edge information in this dissertation are internal forces and structural properties such as sectional areas which have no direction.


4-node graph
graph nodes
graph edges with different weights
*weights in the edge
*features in the node
$n_{1,1}$ $n_{1,2}$ $n_{1,3}$
$n_{2,1}$ $n_{2,2}$ $n_{2,3}$
$n_{3,1}$ $n_{3,2}$ $n_{3,3}$
$n_{4,1}$ $n_{4,2}$ $n_{4,3}$
$p_{1,2} = p_{2,1}$
$p_{1,3} = p_{3,1}$
$p_{1,4} = p_{4,1}$
$p_{3,4} = p_{4,3}$
1
2
3
4

(a)


3 features
4 nodes
Node feature matrix representing features in each node
Adjacency matrix representing connectivity in each node (i.e., edges)
Weighted adjacency matrix representing weights in edges

$$\mathbf{N} = \begin{bmatrix} n_{1,1} & n_{1,2} & n_{1,3} \\ n_{2,1} & n_{2,2} & n_{2,3} \\ n_{3,1} & n_{3,2} & n_{3,3} \\ n_{4,1} & n_{4,2} & n_{4,3} \end{bmatrix}$$

$$\mathbf{M} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{P} = \begin{bmatrix} 0 & p_{1,2} & p_{1,3} & p_{1,4} \\ p_{2,1} & 0 & 0 & 0 \\ p_{3,1} & 0 & 0 & p_{4,3} \\ p_{4,1} & 0 & p_{3,4} & 0 \end{bmatrix}$$

(b)

Figure 2.4: Graph data and graph representation; (a) Graph data, (b) Graph representation from the graph data

In the context of engineering and structural analysis, graph data provides an effective means of representing and understanding the interconnections between different elements within a structure, such as nodal heights and internal forces. At its core, a graph consists of nodes that represent entities or elements, and edges that signify the relationships or connections between these entities.

In structural engineering, nodes often correspond to key points within a physical structure, such as joints or connections where beams, trusses, or columns meet. These nodes are crucial as they define the structural geometry which determines an overall structural performance. By representing nodal positions as attributes associated with each node in the graph, we can effectively capture the structural configuration, and how elements are positioned relative to one another, aiding in tasks like load distribution, deflection calculations, and assessing the strain energy with design specifications, and optimize the structure. Examples of the graph representation for the structural nodes are shown in Refs. [91-93].

Internal forces within a structure, such as axial forces, shear forces, and bending moments, are fundamental indicators of structural behavior and integrity. These forces are transmitted through the various elements of a structure, affecting its stability and safety. Graph data can also be employed to depict these internal forces by incorporating attributes associated with the edges of the graph. Each edge can be encoded with attributes representing the magnitude and direction of internal forces experienced by the structural components it connects. As a result, engineers can create a comprehensive representation of how forces flow through a structure, facilitating the assessment of load-bearing capacity, structural stresses, the identification of potential failure points, and optimization of the structure. Examples of the graph representation for the internal forces are shown in Refs. [94-96].

It is important to note that, in practical structural design optimization, there can also be non-structural components that are related to the structures such as functionality of each area in the structures, parametric lines/surfaces, and solutions in the objective space in the case of multi-objective optimization. Graph representation also offers several approaches to represent these components and combines to foster a more complete representation of the structural designs. Examples of these representations can be seen in the research in building information modeling (BIM) which utilized graph representation for each element in the architecture and structural models, as shown in Refs. [97] and [98].

2.5 Neural networks

Neural networks as shown in Figure 2.5, also known as artificial neural networks [99, 100], are a subset of ML and are at the basis of deep learning algorithms. Neural networks comprise node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, having an associated weight and bias, can compute the output signal from the input it receives. Multiple nodes can be connected, which means some nodes take outputs from other nodes as their inputs. In general, the output signal is transformed by an activation function which acts as a signal threshold. If the output of the node is larger than the threshold value, that signal is activated and sent to the next connected node. Otherwise, no data is passed through. Typically, layers in a neural network can be divided into 3 parts consisting of an input layer, hidden layers, and an output layer. Neural networks are capable of handling multi-dimensional input. Thus, in recent research and real-world applications, they are widely used as nonlinear approximation models. Neural networks depend on training data to learn and improve their accuracy. Once these networks are fine-tuned for accuracy, they become powerful predicting machines that excel in various domains.

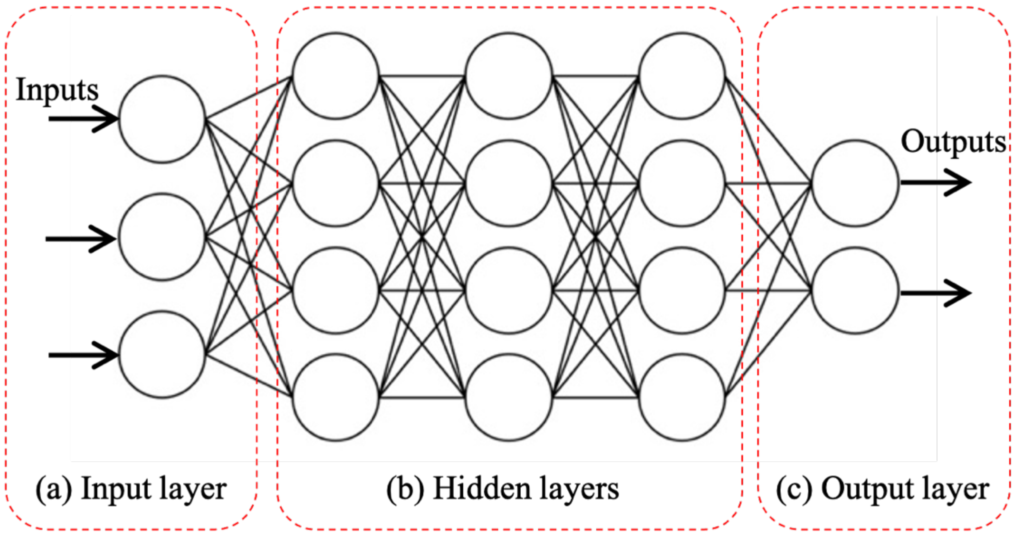


Figure 2.5: Neural Network;
(a) Input layer, (b) Hidden layers, (c) Output layer

A node or neuron in the neural networks has weight (w), bias (b), and activation function $(\sigma(\,))$. In each of these neurons, an input (x) will be passed to each specified neuron. These neurons will calculate their outputs (y') and pass them to specified neurons in the next layer. Note that these w and b are randomly initialized and will be fine-tuned while the activation function is pre-determined and will be the same during the training process. This operation of a neuron can be mathematically written as

$$y' = \sigma(z) = \sigma(wx + b) \tag{2.18}$$

2.5.1 Activation functions

Activation functions are non-linear functions that transform the output of each neuron in neural networks. By incorporating the non-linear functions into neural networks with more than two layers, the networks become a universal function approximator [101]. It should also be noted that these functions have computable gradients of the output with respect to the internal parameters, enabling the gradient-based optimization method to be performed during the training. Some activation functions used in this research are as follows:

<u>Sigmoid activation function</u> [102]

$$Sigmoid(z) = \frac{1}{1 + e^{-z}} \tag{2.19}$$

<u>Rectified Linear Unit (ReLU)</u> [103]

$$ReLU(z) = \max(0, z) \tag{2.20}$$

<u>Leaky Rectified Linear Unit (LeakyReLU)</u> [104]

$$LeakyReLU(z) = \max(\alpha z, z)\colon 0 < \alpha < 1 \tag{2.21}$$

<u>Tanh activation function</u> [105]

$$tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2.22}$$

2.5.2 Training neural networks

In practical implementation, the process of training a neural network involves precise tuning of the weights w and biases b within each neuron of the network according to the corresponding gradient components of the error function, adapting to the discrepancies detected by the network. Given a collection of n pairs of real input and output data (x_i, y_i), the adjustment of w and b in each neuron is methodically carried out. The neural network, defined as $f_{w,b}(\mathbf{x}) = \hat{\boldsymbol{y}}$, computes the predicted output $\hat{y}_i$ from input x_i and then evaluates the disparity between y_i and $\hat{y}_i$. Subsequently, the parameters w and b within each neuron are optimized to minimize the disparity between y_i and $\hat{y}_i$, a quantity referred to as the Loss ($\mathcal{L}(y_i, \hat{y}_i)$). This loss is determined through various methods, one of which is the employment of a regression loss function like the Mean Squared Error (MSE) in RL contexts, where y_i and $\hat{y}_i$ represented expected reward (i.e., Q-value) and the real obtained accumulated reward. The MSE loss function, applied to n data pairs (x_i, y_i), is mathematically expressed as:

$$\mathcal{L}(y_i, \hat{y}_i) = MSE = \frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n} \tag{2.23}$$

Note that we are interested in adjusting w and b in the neural network to reduce the loss defined in Eq. (2.23). The magnitudes of these adjustments are the gradients of the loss function with respect to w and b which can be calculated, using the chain rule of calculus defined in Eqs. (2.24) and (2.25), respectively, as

$$\frac{\partial \mathcal{L}(y_i, \hat{y}_i)}{\partial f_w(x_i)} \frac{\partial f_w(x_i)}{\partial w} = \frac{\partial \mathcal{L}(y_i, \hat{y}_i)}{\partial w} \tag{2.24}$$

$$\frac{\partial \mathcal{L}(y_i, \hat{y}_i)}{\partial f_b(x_i)} \frac{\partial f_b(x_i)}{\partial b} = \frac{\partial \mathcal{L}(y_i, \hat{y}_i)}{\partial b} \tag{2.25}$$

Let α be a learning rate parameter. The adjustments of w and b are as follows:

$$w \leftarrow w - \alpha \frac{\partial \mathcal{L}(y_i, \hat{y}_i)}{\partial w} \tag{2.26}$$

$$b \leftarrow b - \alpha \frac{\partial \mathcal{L}(y_i, \hat{y}_i)}{\partial b} \tag{2.27}$$

Note that the process that propagates input information x_i along the neural network to obtain the final product $\hat{y}_i$ is called forward propagation [51]. After that the Loss is computed from $\hat{y}_i$, and the information from this loss flows backward through the neural network to obtain the gradients. This information is called the Backpropagation algorithm [106], and (x_i, y_i) is commonly called *training set* or *training data*. If the amount of training set is large, the computational time is also large. In modern neural network training, this training time can be optimized by sampling a small amount from the whole training set, i.e., Batch gradient descent, or by sampling only one sample from the whole training set, i.e., SGD. Training a neural network using Batch gradient descent or SGD can be further improved using optimization methods in the following section.

2.5.3 Optimization methods of gradient descent

The original SGD requires enormous optimization steps to tune up the parameters of the approximation function. Therefore, variants of the SGD algorithm have been developed to increase the speed of optimization. This research utilizes the Adam algorithm to optimize the weights in the approximation function in RL. The Adam algorithm is based on two algorithms; Momentum [107] and RMSProp [108], all of which are explained as follows:

Momentum

Since, in some cases, SGD oscillates around the local optima of each time step of SGD, utilizing momentum can help to move SGD toward the local optima faster using the weighted average. The parameter θ is not being updated using its gradient $\nabla_\theta f(\theta)$ directly, but using the weighted average v_t of its gradient in the past time steps computed as

$$v_t = \gamma v_{t-1} - \alpha \nabla_\theta f(\theta) \tag{2.28}$$

$$\theta = \theta + v_t \tag{2.29}$$

where γ and α are the parameters for momentum and learning rate. v_t is initialized as $v_0 = 0$.

RMSProp

RMSProp (Root Mean Square Propagation) is a type of adaptive learning rate algorithm that scales gradient update $\Delta\theta$ of each parameter θ inversely to the square root of the exponentially weighted moving average of the accumulated squared value of the gradient. Gradient update $\Delta\theta$ is defined as

$$r_t = \rho r_{t-1} + (1-\rho)\nabla_\theta f(\theta) \odot \nabla_\theta f(\theta) \tag{2.30}$$

$$\Delta\theta = -\frac{1}{\sqrt{\delta + r_t}} \odot \nabla_\theta f(\theta) \tag{2.31}$$

where ρ and δ are the decay rate and the small constant to stabilize the algorithm. r_t is initialized as $r_0 = 0$. $\odot$ is an element-wise product (Hamadard product). The parameter θ is updated with learning rate α by

$$\theta = \theta + \alpha\Delta\theta \tag{2.32}$$

Adam

Adam (Adaptive Moments Estimation) is a type of adaptive learning rate algorithm that combines RMSProp and Momentum algorithms. Adam algorithm calculates gradient update Δθ using both scaled gradient and momentum of the gradient with bias corrections in both terms. The gradient update Δθ is defined as follows:

$$s_t = \rho_1 s_{t-1} + (1-\rho_1)\nabla_\theta \mathrm{f}(\theta) \tag{2.33}$$

$$r_t = \rho_2 r_{t-1} + (1-\rho_2)\nabla_\theta \mathrm{f}(\theta)\odot\nabla_\theta \mathrm{f}(\theta) \tag{2.34}$$

$$\hat{s} = \frac{s_t}{1-\rho_1^t} \tag{2.35}$$

$$\hat{r} = \frac{r_t}{1-\rho_2^t} \tag{2.36}$$

$$\Delta\theta = -\frac{\hat{s}}{\sqrt{\delta + \hat{r}}} \tag{2.37}$$

where ρ_1, ρ_2, $\hat{s}$, and $\hat{r}$ are a decay rate for momentum, a decay rate for RMSProp, a correct bias for momentum, and a correct bias for RMSProp. The parameter θ is updated with learning rate α by

$$\theta = \theta + \alpha\Delta\theta \tag{2.38}$$

2.5.4 Variants of the Neural Networks

Note that suitable ML models depend on the data type to be processed; for example, the neural networks are applicable to the tabular data while the convolutional neural networks [109] are suitable for processing image data. This research utilizes multiple variants of neural networks for making the RL agent. The basic idea of choosing these variants depends on the type of input data and the desirable output data. In general, if the input data are the graph representation, then graph neural networks will be utilized, whereas if the input data are a vector or the output should be a scalar, then the multi-layer perceptron (MLP) will be utilized instead. Note that the same actor or critic model can be made of both graph neural networks and MLP. Graph neural networks utilized in this research include Graph Convolutional Networks (GCNs) [110] and Graph Attention Networks (GATs) [111]. The detailed explanation of MLP, GCN, and GAT are as follows:

Multi-layer perceptron

A multi-layer perceptron is a name for a neural network that consists of fully connected neurons with nonlinear activation functions. The illustration of the MLP is similar to Figure 2.5. This research utilizes the MLPs for making the critics in DDPG and MADDPG agents because the critics aim to compute the expected rewards which are scalar values. Note some examples in this research also utilize MLPs when the input data are vectors.

Graph Convolutional Networks

Graph convolutional network is a type of neural network that can process graph representations. Integrated within the RL framework, GCNs are utilized for making policy and value functions. Utilizing GCNs, the policy function can compute action on the graph (i.e., embedded node signal) from a given observation through graph representations. The value function made of GCNs can predict rewards from the embedded node signal and graph representations of the observation. Let $\mathbf{I} \in \mathbb{R}^{n\times n}$ and $\mathbf{D}^{-1/2}$ be the identity matrix and the inverse of the matrix $\mathbf{D}^{1/2}$ satisfying $\mathbf{D}^{1/2}\mathbf{D}^{1/2} = \mathbf{D}$, respectively. A GCN computing unit is referred to as a *layer* that takes $\mathbf{N}$ and either a normalized adjacency matrix $\mathbf{M}$ denoted as $\widetilde{\mathbf{M}} = \mathbf{D}^{-1/2}[\mathbf{M} + \mathbf{I}]\mathbf{D}^{-1/2}$ or weighted adjacency matrix $\mathbf{P}$ as inputs to compute embedded node signal $\mathbf{N}'$ as follows:

$$\mathbf{N}' = \sigma\big(\widetilde{\mathbf{M}}\mathbf{N}\mathbf{w}\big) \text{ or } \mathbf{N}' = \sigma(\mathbf{P}\mathbf{N}\mathbf{w}) \tag{2.39}$$

where $\sigma(\cdot)$ denotes a nonlinear activation function. $\mathbf{w} \in \mathbb{R}^{g\times h}$ is an internal weight in the GCN layer with h filters or feature maps [110]. Note that multiple GCN layers can be stacked and the next layer can take the embedded node signal $\mathbf{N}'$ from the previous layer as one of the inputs instead of the original $\mathbf{N}$.

Graph convolutional network is applicable in various graph problems including node classification [112], link prediction [113], and graph classification [114]. This research utilizes the GCNs for making the actors and critics in DDPG and MADDPG agents when the input data are graph representations in Chapters 3, 4, and 6.

Graph Attention Networks

Graph attention network utilizes a parameterized weight $\mathbf{w}$ and attention weight (i.e., attention coefficient) $\boldsymbol{\alpha}$ to indicate the importance of node features and neighboring nodes connected to the interested node, respectively. Graph attention network computes embedded node signal $\mathbf{N}'$ by taking the node feature matrix $\mathbf{N}$ and either the adjacency matrix $\mathbf{M}$ or weighted adjacency matrix $\mathbf{P}$ as inputs as follows:

$$\mathbf{N}' = \sigma(\boldsymbol{\alpha}\mathbf{MNw}) \text{ or } \mathbf{N}' = \sigma(\boldsymbol{\alpha}\mathbf{PNw}) \tag{2.40}$$

where $\sigma(\cdot)$ is a non-linear activation function. A *layer* can be considered as a GAT computing unit, and layers can be stacked to construct a computation model. When stacked, the next layer can take the embedded node signal $\mathbf{N}'$ of the previous layer as one of the inputs.

The attention coefficient of node i that has $\mathcal{N}_i$ adjacent nodes and its neighboring node $j \in \mathcal{N}_i$ can be computed as follows:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^{\mathrm{T}}\left[\mathbf{w}h_i \parallel \mathbf{w}h_j\right]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}(\mathbf{a}^{\mathrm{T}}[\mathbf{w}h_i \parallel \mathbf{w}h_k])\right)} \tag{2.41}$$

where $\mathbf{a}$ and h_i are the trainable attention weight and the features of node i, respectively. exp (x) and $\parallel$ denote the exponential function of x and concatenation which horizontally joins two or more matrices to make a new matrix, respectively. The subscript ij indicates the (i, j) component of a matrix.

Similar to GCN, GAT is also applicable in various domains including node classification [115], link prediction [116], and graph classification [117]. This research also utilizes the GATs for making the DDPG agent in Chapter 5.

2.6 Graph and matrix operations

In order to train RL agents, it is important to represent adequate state information for the agent so that it can learn about the relationship between the state and its decisions on doing action. One innovation in this dissertation is to utilize graph representation in Section 2.4 on multiple types of information (i.e., *domain*) such as the architectural structure, parametric control points for the structural geometry, and non-dominated solution in the objective space in the bi-objective optimization case. To combine information from different domains, this research utilizes two operations to transform a matrix into a vector and to transform a vector into a matrix as follows:

2.6.1 Global sum pooling operation

Global sum pooling operation (GSP) [118] transforms the matrix into a vector by summing up all entries in each column of the output matrix. Let $\mathbf{V} \in \mathbb{R}^{n\times g}$ be a matrix, the GSP operation to transform $\mathbf{V}$ into a vector can be represented as

$$\mathrm{GSP}(\mathbf{V}) = \left[\sum_{i=1}^{n} \mathrm{v}_{i,1} \quad \cdots \quad \sum_{i=1}^{n} \mathrm{v}_{i,g}\right] \in \mathbb{R}^{1\times g} \tag{2.42}$$

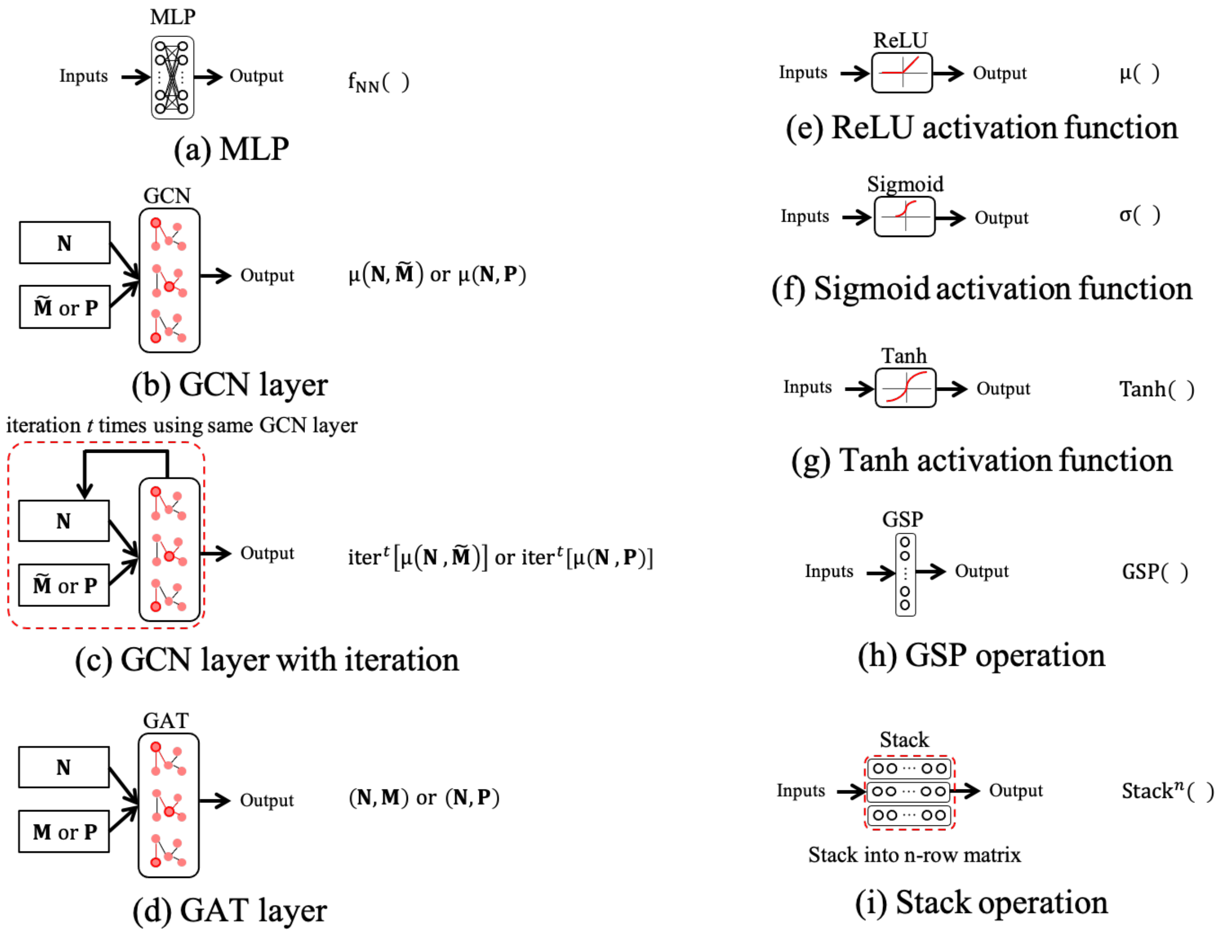
2.6.2 Stack operation

This operation transforms a row vector into a matrix with desirable number of rows and the same number of columns. The stack operation that transforms a vector $\mathbf{a} \in \mathbb{R}^{1\times g}$ into a matrix $\mathbf{A} \in \mathbb{R}^{n\times g}$, is represented as

$$\mathrm{Stack}^n(\mathbf{a}) = \left.\begin{bmatrix}\mathbf{a}\\ \vdots \\ \mathbf{a}\end{bmatrix}\right\} n \text{ times} = \mathbf{A} \in \mathbb{R}^{n\times g} \tag{2.43}$$

2.7 Computational blocks

This dissertation utilizes variants of the neural networks in Section 2.5.4, and operations in Section 2.6 for making policy and value functions for the RL agents. These neural networks and operations are defined as *computational blocks* and connected to become the functions. Figure 2.6 illustrates the symbols of computational blocks and functions of the MLPs, GCNs, GATs, GSP, and stack operation utilized throughout this dissertation. Figure 2.7 illustrates an example of an ML model represented as a group of computational blocks and functions. In this Figure, (a) illustrates the MLP that takes *Input* and returns *Output*, denoted as $\mathrm{f}_{\mathrm{NN}}(\ )$. Figure 2.6 (b) shows the GCN layer that takes $\mathbf{N}$ and $\widetilde{\mathbf{M}}$ or $\mathbf{P}$ as inputs, denoted as $\mu(\mathbf{N}, \widetilde{\mathbf{M}})$ or $\mu(\mathbf{N}, \mathbf{P})$. Figure 2.6 (c) illustrates the GCN layer that takes $\mathbf{N}$ and $\widetilde{\mathbf{M}}$ or $\mathbf{P}$ as inputs, and, after obtaining the output, utilizes its own output as $\mathbf{N}$ to compute the output again for t times, denoted as $\mathrm{iter}^t[\mu(\mathbf{N}, \widetilde{\mathbf{M}})]$ or $\mathrm{iter}^t[\mu(\mathbf{N}, \mathbf{P})]$. Figure 2.6 (d) shows the GAT layer that takes $\mathbf{N}$ and $\widetilde{\mathbf{M}}$ or $\mathbf{P}$ as inputs, denoted as $\mu(\mathbf{N}, \widetilde{\mathbf{M}})$ or $\mu(\mathbf{N}, \mathbf{P})$. Note that the μ, in Figures 2.6 (a)-(d), denotes the activation functions that are selected from activation functions in Figures 2.6 (e), (f), (g), indicating ReLU, Sigmoid, and Tanh, respectively. Figures 2.6 (h) and (i) represent the GSP operation and Stack operation, respectively.



MLP
Inputs
Output
$f_{NN}(\ )$
(a) MLP
GCN
N
$\tilde{\mathbf{M}}$ or P
Output
$\mu(\mathbf{N},\tilde{\mathbf{M}})$ or $\mu(\mathbf{N},\mathbf{P})$
(b) GCN layer
iteration t times using same GCN layer
N
$\tilde{\mathbf{M}}$ or P
Output
$\mathrm{iter}^t[\mu(\mathbf{N},\tilde{\mathbf{M}})]$ or $\mathrm{iter}^t[\mu(\mathbf{N},\mathbf{P})]$
(c) GCN layer with iteration
GAT
N
M or P
Output
$(\mathbf{N},\mathbf{M})$ or $(\mathbf{N},\mathbf{P})$
(d) GAT layer
ReLU
Inputs
Output
$\mu(\ )$
(e) ReLU activation function
Sigmoid
Inputs
Output
$\sigma(\ )$
(f) Sigmoid activation function
Tanh
Inputs
Output
Tanh( )
(g) Tanh activation function
GSP
Inputs
Output
GSP( )
(h) GSP operation
Stack
Inputs
Output
$\mathrm{Stack}^n(\ )$
Stack into n-row matrix
(i) Stack operation

Figure 2.6: Symbols of computational blocks and functions; (a) MLP, (b) GCN layer, (c) GCN layer with iteration, (d) GAT layer, (e) ReLU activation function, (f) Sigmoid activation function, (g) Tanh activation function, (h) GSP operation, (i) Stack operation

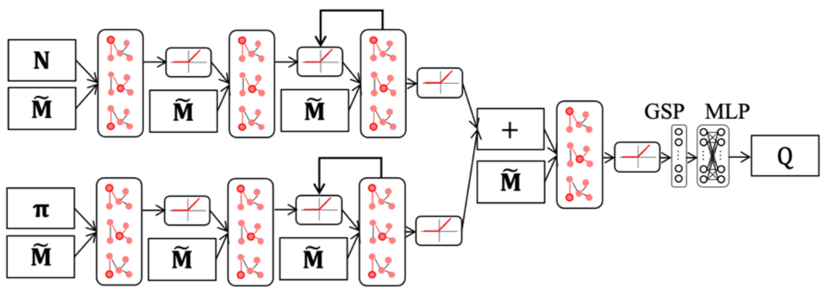


Figure 2.7: Example of an ML model represented as a group of computational blocks and functions

Neural networks, including specialized variants like MLPs, GCNs, and GATs, can be applied to architectural and engineering problems by utilizing them as ML models. With sufficient training methods and training data, these computational models can be leveraged to enhance diverse tasks such as predictive or surrogate models [119-121], pattern recognitions in structural engineering [122-124], and designs and optimizations [49,125,126].

2.8 Conclusion

This chapter has presented a brief review of machine learning, especially reinforcement learning, together with their applications in structural engineering. It covers the DDPG and MADDPG algorithms and computational ingredients including MLP, GCN, and GAT that are utilized for constructing the RL agents proposed in the subsequent chapters.

Chapter 3
Topology optimization and sizing optimization of lattice shells using DDPG and GCN

3.1 Introduction

Optimization of lattice and grid shell structures presents an important task in the field of structural engineering, with topology and sizing of structural elements being the primary considerations for achieving optimal designs. These optimization problems are inherently categorized as discrete combinatorial optimization, characterized by the need to select the most suitable arrangement and dimensions of structural elements within the lattice shell. The discrete nature of these problems renders them to be solved using gradient-based optimization techniques, requiring alternative approaches for their resolution.

In practice, the solutions to topology and sizing optimization problems are often sought through stochastic- or population-based algorithms such as SA and GA. These methods, while effective in some cases, come with certain drawbacks. Notably, they tend to require a significant computational burden due to their iterative nature. Moreover, the heuristic elements inherent in these algorithms can lead to diverse outcomes across multiple trials, introducing uncertainty and requiring additional computational resources for statistical evaluation. Consequently, there exists a compelling need for innovative techniques that can address these challenges by reducing computational costs and stabilizing optimization outcomes.

This chapter is dedicated to exploring the application of RL as a potential solution to address these problems in the context of topology and sizing optimizations for lattice shell structures. Section 3.1 sets the stage by explaining the fundamental topology and sizing optimization problems encountered in lattice shell design. These problems are framed as discrete combinatorial optimization problems, where the objective is to obtain the optimal configuration and dimensions of structural elements within the lattice shell. The chapter then proceeds to introduce an approach in Section 3.2, employing a specific RL framework known as DDPG together with a graph-based structural representation. This approach enhances an RL agent, constructed using GCNs, to learn and optimize the bracing direction within the lattice shell structure with the goal of minimizing the total strain energy. The utilization of graph representations enables the agent to navigate the intricate and discrete optimization landscape inherent to topology optimization.

The extension of the RL methodology to sizing optimization is detailed in Section 3.3, where the capabilities of the agent are broadened to optimize structural element dimensions. Sizing optimization is a complex multi-dimensional problem, and the RL agent is equipped to navigate this discrete parameter space while respecting design constraints. The reward function for this task is constructed based on changes in structural volume, aligning with the objective of sizing optimization. Finally, Section 3.4 synthesizes and presents the key findings of this chapter, highlighting the potential of RL, specifically the DDPG framework, to solve the problems of lattice shell engineering design by reducing computational cost and enhancing the stability of optimization outcomes.

3.2 Formulation of optimization problems

3.2.1 Topology optimization

The topology optimization problem aims to minimize the total strain energy of lattice shells subjected to static loads. The lattice shells consist of main grid elements and bracing elements which are modeled as 3-dimensional beam elements with 12 degrees of freedom (DoFs) and 3-dimensional truss elements with 6-DoFs, respectively. The stiffness matrices, in the local coordinate system, of the beam element and bracing element are defined as $\mathbf{k}_{\mathrm{f}} \in \mathbb{R}^{12\times 12}$ and $\mathbf{k}_{\mathrm{e}} \in \mathbb{R}^{6\times 6}$, respectively. The global stiffness matrix $\mathbf{K} \in \mathbb{R}^{n_{\mathrm{D}}\times n_{\mathrm{D}}}$ is assembled from these local stiffness matrices with respect to the global coordinate system, with n_{D} as the number of DoFs of the shell after assigning the boundary conditions. Nodes and elements in the shell are subjected to a point load and self-weight per unit length, respectively. These loads and weights are aggregated into the load vector $\mathbf{p} \in \mathbb{R}^{n_{\mathrm{D}}}$, with respect to the global coordinates. The nodal displacement vector $\mathbf{d} \in \mathbb{R}^{n_{\mathrm{D}}}$ is obtained by solving the equilibrium equations defined as

$$\mathbf{Kd} = \mathbf{p} \tag{3.1}$$

The total strain energy E is computed from

$$E = \frac{1}{2}\mathbf{d}^{\mathrm{T}}\mathbf{K}\mathbf{d} \tag{3.2}$$

In this problem, given a lattice shell with n_x by n_y square grids (Figure 3.1(a)) and diagonal bracing in each grid cell $k \in \{1, \dots, n_x n_y\}$, there can be two possible directions for bracing indicated by $c_{1,k}, c_{2,k} \in \{0,1\}$ which correspond to absence and presence of the brace in each direction (Figure 3.1(b)).

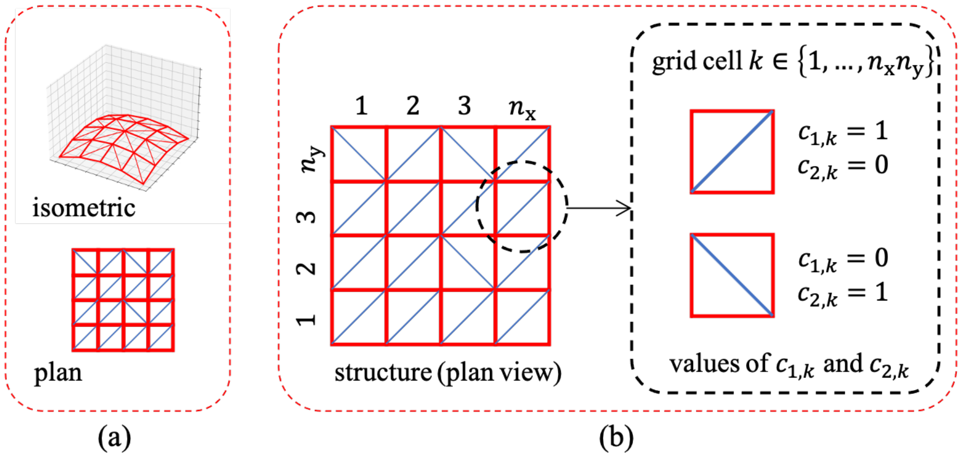


Figure 3.1: Bracing directions in the grid cell k and corresponding values of $c_{1,k}$ and $c_{2,k}$: bracing direction optimization of lattice shells

From Figure 3.1, only one brace should exist in the grid cell k, and the summation $c_{1,k} + c_{2,k}$ is always 1. Let $\mathbf{c}_1 \in \mathbb{R}^{n_{\mathrm{x}}n_{\mathrm{y}}}$ and $\mathbf{c}_2 \in \mathbb{R}^{n_{\mathrm{x}}n_{\mathrm{y}}}$ be vectors consisting of $c_{1,k}$ and $c_{2,k}$ for all bracing elements. The global stiffness matrix of the structure is formulated as a function of $\mathbf{c}_1$ and $\mathbf{c}_2$ denoted as $\mathbf{K}(\mathbf{c}_1, \mathbf{c}_2)$. The topology optimization problem to minimize the strain energy, by defining bracing directions, is formulated as follows:

$$\begin{aligned} &\text{minimize} && E(\mathbf{c}_1, \mathbf{c}_2) = \frac{1}{2}\mathbf{d}^{\mathrm{T}}\mathbf{K}(\mathbf{c}_1, \mathbf{c}_2)\mathbf{d} && \\ &\text{subject to} && c_{1,k}, c_{2,k} \in \{0,1\}, && (k = 1, 2, \dots, n_{\mathrm{x}}n_{\mathrm{y}}) \\ &&& c_{1,k} + c_{2,k} = 1, && (k = 1, 2, \dots, n_{\mathrm{x}}n_{\mathrm{y}}) \end{aligned} \tag{3.3}$$

where $\mathbf{d}$ is an implicit function of $\mathbf{c}_1$ and $\mathbf{c}_2$ obtained by solving Eq. (3.1).

3.2.2 Sizing optimization

In this problem, the structural volume of the lattice shell, subjected to external load under stress constraints, is the objective function to be minimized. Let n be the number of lattice beam elements in the structure. Predetermined indices of cross-sectional properties and lengths of element i are denoted as $c_i \in \{0,1,\dots,n_\mathrm{c}\}$ and L_i, respectively. The structural volume V of a lattice shell is obtained from sectional areas $A(c_i)$ when defining the cross-sectional indices c_i and lengths L_i, as follows

$$V(\mathbf{c}) = \sum_{i=1}^{n} A(c_i) \cdot L_i \tag{3.4}$$

In this sizing optimization problem, all the structural elements are three-dimensional beam elements with 12 DoFs. The internal stress in the structural element is derived from local displacements of each element after solving the stiffness equation similar to Eq. (3.1). The ratio of internal stress to the allowable stress σ_i of an element i is computed similar to Largaros et al. [127], expressed as follows:

$$\sigma_i = \begin{cases} \frac{|\sigma_1|}{\bar{\sigma}_1} + \frac{|\sigma_2|}{\bar{\sigma}_2} + \frac{|\sigma_3|}{\bar{\sigma}_3} & : \text{if } \frac{|\sigma_1|}{\bar{\sigma}_1} \leq 0.15 \\ \frac{|\sigma_1|}{0.6\times\bar{\sigma}_1} + \frac{|\sigma_2|}{\bar{\sigma}_2} + \frac{|\sigma_3|}{\bar{\sigma}_3} & : \text{if } \frac{|\sigma_1|}{\bar{\sigma}_1} > 0.15 \end{cases} \tag{3.5}$$

where σ_1, σ_2, and σ_3 denote the axial stress, maximum stress due to the bending moment in the local y-axis, and in the local z-axis, respectively, and $\bar{\sigma}_1$, $\bar{\sigma}_2$, and $\bar{\sigma}_3$ are their upper bounds.

The sizing optimization problem of the lattice shell to minimize the structural volume under stress constraints is defined using Eq. (3.4) as follows:

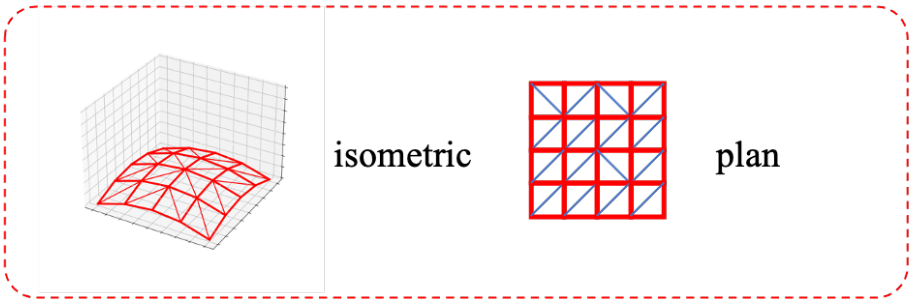
$$\begin{aligned} \text{minimize} \quad & V(\mathbf{c}) \\ \text{subject to} \quad & \sigma_{\max} \leq 1 \\ & c_i \in \{0,1,\dots,n_{\mathrm{c}}\} \end{aligned} \tag{3.6}$$

where $\sigma_{\max}$ denotes the maximum absolute value of the stresses in the structure.

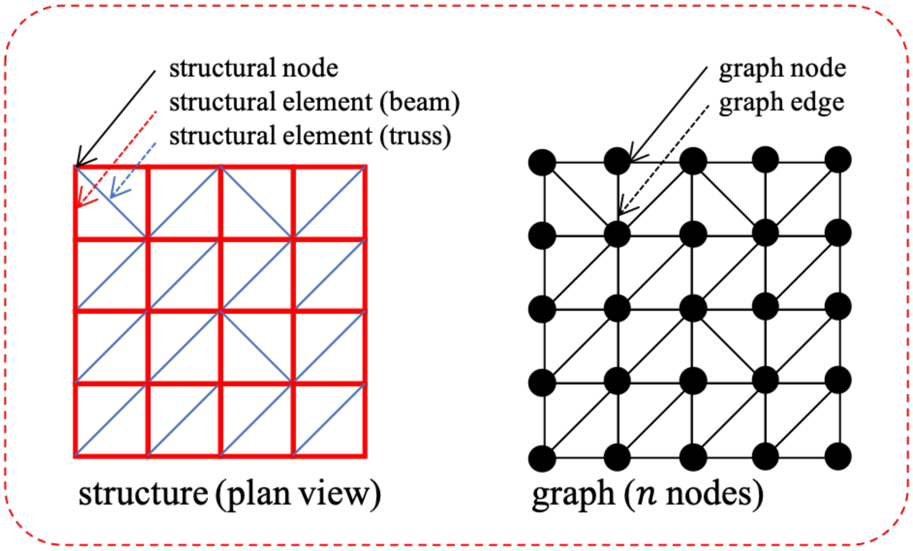
3.3 Topology optimization of lattice shell using DDPG and GCN

This section introduces the RL method for solving problems in Section 3.1.1. The optimization problem is formulated into an MDP where the state is the current structural configuration and response, the action is the modification of bracing directions, and the reward is computed from the improvement of the objective function. The agent is the DDPG agent that takes state data as inputs to compute the output (i.e., action).

In order to provide information on the structural configurations and response to the agent, this research utilizes the graph representation as explained in Section 2.4 in Chapter 2, illustrated in Figure 3.2. In this problem, the lattice shell (Figure 3.2(a)) is represented as a graph where the graph nodes and edges represent structural nodes and elements, respectively (Figure 3.2(b)). Information of the structural nodes including coordinates and boundary conditions are stored in the graph node and represented using a node feature matrix. Information on the structural elements such as the existence/non- existence of the element and internal forces are represented through adjacency and weighted adjacency matrices (Figure 3.2(c)). The RL agent utilizes these matrices as state data to compute the actions in each optimization step.



(a)



(b)

$$\mathbf{N} = \overbrace{\begin{bmatrix} n_{1,1} & \cdots & n_{1,f} \\ \vdots & \ddots & \vdots \\ n_{n,1} & \cdots & n_{n,f} \end{bmatrix}}^{f \text{ features}} \left.\vphantom{\begin{bmatrix} a \\ a \\ a \end{bmatrix}}\right\} n \text{ nodes}$$

Node feature matrix

$$\mathbf{M} = \overbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & \cdots & \vdots \\ 0 & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}}^{n \text{ nodes}} \left.\vphantom{\begin{bmatrix} a \\ a \\ a \\ a \end{bmatrix}}\right\} n \text{ nodes}$$

Adjacency matrix

$$\mathbf{P} = \overbrace{\begin{bmatrix} 0 & p_{1,2} & 0 & 0 \\ p_{2,1} & 0 & \cdots & \vdots \\ 0 & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}}^{n \text{ nodes}} \left.\vphantom{\begin{bmatrix} a \\ a \\ a \\ a \end{bmatrix}}\right\} n \text{ nodes}$$

Weighted adjacency matrix

(c)

Figure 3.2: Graph representation of the lattice shell for bracing direction optimization of lattice shells

Figure 3.3 shows the MDP of the topology optimization of the lattice shell, formulated as explained in Section 2.3 in Chapter 2. At a timestep t, the agent observes the state S_t which is the graph representation of the current structural configuration shown in Figure 3.3(a). The agent does the action A_t which determines the bracing direction in each grid, illustrated in Figure 3.3(b). The structure is modified and the objective function is computed. The agent obtains a reward R_{t+1} based on the modified structure shown in Figure 3.3(c) and the new step of MDP begins. The optimization ends when t reaches a maximum $t_{\max}$.

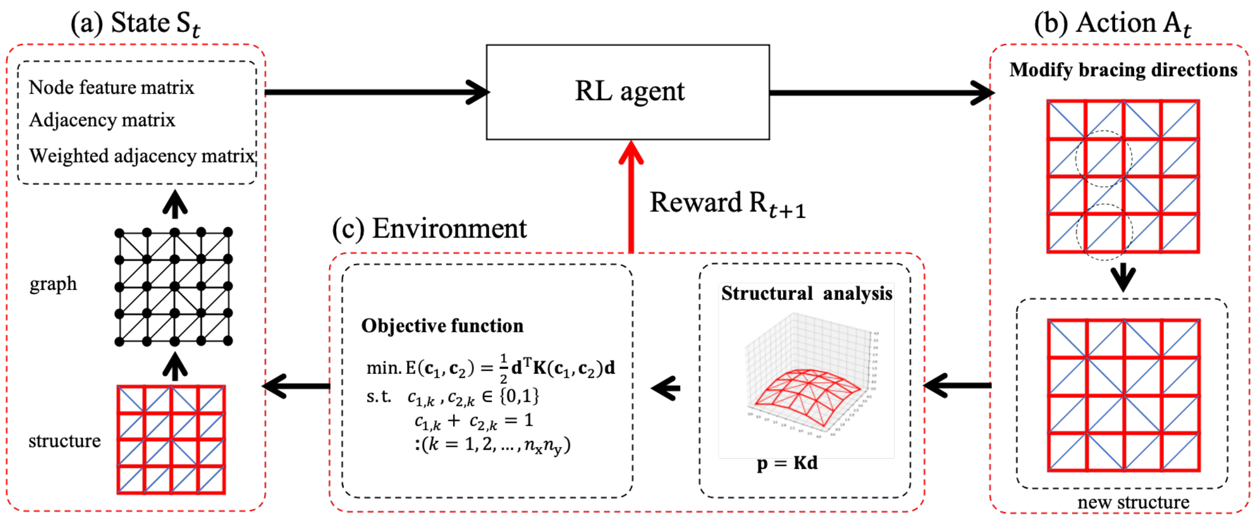

(a) State S_t
Node feature matrix
Adjacency matrix
Weighted adjacency matrix
graph
structure
RL agent
(b) Action A_t
Modify bracing directions
new structure
Reward R_{t+1}
(c) Environment
Objective function
$\min. \mathrm{E}(\mathbf{c}_1, \mathbf{c}_2) = \frac{1}{2}\mathbf{d}^{\mathrm{T}}\mathbf{K}(\mathbf{c}_1, \mathbf{c}_2)\mathbf{d}$
s.t. $c_{1,k}, c_{2,k} \in \{0,1\}$
$c_{1,k} + c_{2,k} = 1$
$:(k = 1, 2, \ldots, n_x n_y)$
Structural analysis
$\mathbf{p} = \mathbf{Kd}$

Figure 3.3: MDP of the bracing direction optimization of lattice shells

3.3.1 State

In this problem, information of the structural nodes is represented through the node feature matrix $\mathbf{N} \in \mathbb{R}^{n\times 5}$. The i th row of $\mathbf{N}$, representing information of node i, is $\boldsymbol{n}_i = \left\{x_i/\max\limits_p x_p \quad y_i/\max\limits_p y_p \quad z_i/z_{\max} \quad k_{\text{free}}^i \quad k_{\text{fix}}^i\right\}$ where x_i, y_i, and z_i are the coordinates of node i, $z_{\max}$ is the pre-determined upper-bound value of z_i. $\max\limits_p x_p$ and $\max\limits_p y_p$ are the maximum x and y coordinate values, respectively, and the minimum coordinate values are 0 in all coordinates. k_{free}^i and k_{fix}^i are defined based on the boundary conditions (i.e., supports) as

$(k^i_{\mathrm{free}}, k^i_{\mathrm{fix}}) = (0, 1)$ if node i is a fixed support, and $(k^i_{\mathrm{free}}, k^i_{\mathrm{fix}}) = (1, 0)$ if node i is not supported.

Information about the existence of lattice beam elements is represented using the adjacency matrix for beam elements $\mathbf{M}_1 \in \mathbb{R}^{n\times n}$. In this matrix, $m_{1_{ij}}$, indicating the (i, j) component of matrix $\mathbf{M}_1$, denotes the existence of a beam element e connecting nodes i and j. Note that $m_{1_{ij}} = m_{1_{ji}} = k^e_{\mathrm{beam}}$, and k^e_{beam} indicates the existence and non-existence of a 12-DoF beam element e that connects nodes i and j by $k^e_{\mathrm{beam}} = 1$ and 0, respectively. Similar to $\mathbf{M}_1$, the adjacency matrix for truss elements $\mathbf{M}_2 \in \mathbb{R}^{n\times n}$ represents the existence of a truss element e connecting nodes i and j as $m_{2_{ij}} = m_{2_{ji}} = k^e_{\mathrm{truss}}$, which indicates the existence and non-existence of a 6-DoF truss element e that connects nodes i and j by $k^e_{\mathrm{truss}} = 1$ and 0, respectively.

To evaluate efficiency of the structural configuration, this research introduces weighted adjacency matrices representing the internal forces. Information of the internal force in beam elements is represented through weighted adjacency matrix $\mathbf{P}_1 \in \mathbb{R}^{n\times n}$ whose entries are the ratio between the bending moment and the axial force in the elements. Let beam element e connect nodes i and j. The entry $p_{1_{ij}}$ in $\mathbf{P}_1$ is expressed as follows:

$$p_{1_{ij}} = k^e_{\mathrm{beam}}\, b'_{ei}/(a'_e + 1) \tag{3.7}$$

$$b'_{ei} = \left(|b_{ei}| - b_{\mathrm{f}}{}^{\mathrm{min}}\right)/\left(b_{\mathrm{f}}{}^{\mathrm{max}} - b_{\mathrm{f}}{}^{\mathrm{min}}\right) \tag{3.8}$$

$$a'_e = (|a_e| - a_{\mathrm{f}}{}^{\mathrm{min}})/(a_{\mathrm{f}}{}^{\mathrm{max}} - a_{\mathrm{f}}{}^{\mathrm{min}}) \tag{3.9}$$

where b_{ei} is the bending moment around the horizontal axis on the section at node i, and a_e is the axial force of beam element e. $b_{\mathrm{f}}{}^{\mathrm{max}}$ and $b_{\mathrm{f}}{}^{\mathrm{min}}$ are the maximum and minimum absolute values of bending moments at the element ends of all beam elements. $a_{\mathrm{f}}{}^{\mathrm{max}}$ and $a_{\mathrm{f}}{}^{\mathrm{min}}$ are the maximum and minimum absolute axial forces of beam elements.

Information of the internal force in the truss elements is represented by $\mathbf{P}_2 \in \mathbb{R}^{n\times n}$ whose entries are normalized forms of the axial forces. The entry $p_{2_{ij}}$ in $\mathbf{P}_2$ corresponding to element e connecting nodes i and j is defined as

$$p_{2_{ij}} = k^e_{\mathrm{truss}} a'_e \tag{3.10}$$

$$a'_e = (|a_e| - a_{\mathrm{q}}{}^{\mathrm{min}})/(a_{\mathrm{q}}{}^{\mathrm{max}} - a_{\mathrm{q}}{}^{\mathrm{min}}) \tag{3.11}$$

where a_e is the axial force in the truss element e. $a_{\mathrm{q}}{}^{\mathrm{max}}$ and $a_{\mathrm{q}}{}^{\mathrm{min}}$ are the maximum and minimum absolute values of axial forces for truss elements, respectively.

Note that all values in the node adjacency matrix and weighted adjacency matrices are in the range of $[0,1]$, which helps avoid numerical instability during training.

Values in adjacency matrices are normalized using the degree matrices. Degrees of connectivity of each structural node to other nodes, through beam and truss elements, are represented using degree matrices for beam, truss, and combined beam and truss elements, denoted by $\mathbf{D}_1 \in \mathbb{R}^{n\times n}$, $\mathbf{D}_2 \in \mathbb{R}^{n\times n}$, and $\mathbf{D}_3 \in \mathbb{R}^{n\times n}$, respectively. Entries in these matrices $\mathbf{D}_u$ $(u = 1,2,3)$ are computed using their associated adjacency matrix $\mathbf{M}_u$ $(u = 1,2,3)$ as $d_{u_{ij}} = \delta_{ij}\sum_{i=1}^{n} m_{u_{ij}}$, where δ_{ij} is the Kronecker delta which is 0 if $i \neq j$ and 1 if $i = j$. The normalized adjacency matrices of beam, truss, and combined beam and truss elements denoted as $\widetilde{\mathbf{M}}_1 \in \mathbb{R}^{n\times n}$, $\widetilde{\mathbf{M}}_2 \in \mathbb{R}^{n\times n}$, and $\widetilde{\mathbf{M}}_3 \in \mathbb{R}^{n\times n}$ are computed as

$$\widetilde{\mathbf{M}} = \mathbf{D}^{-1/2}[\mathbf{M} + \mathbf{I}]\mathbf{D}^{-1/2} \tag{3.12}$$

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

Table 3.1 summarizes entries in the node feature matrix **N**, and Table 3.2 shows entries in the adjacency matrix of beam elements $\mathbf{M}_1$, the adjacency matrix of truss elements $\mathbf{M}_2$, the adjacency matrix of combined beam and truss elements $\mathbf{M}_3$, the weighted adjacency matrix of beam $\mathbf{P}_1$, and the weighted adjacency matrix of truss $\mathbf{P}_2$.

Table 3.1: Entries in node feature matrix (**N**) for bracing direction optimization of lattice shells

$n_{i,1}$	x-coordinate of node i: $x_i/\max\limits_p x_p$
$n_{i,2}$	y-coordinate of node i: $y_i/\max\limits_p y_p$
$n_{i,3}$	z-coordinate of node i: z_i/z_{max}
$n_{i,4}$	1 if the node has fixed support, else 0: k_{fix}^i
$n_{i,5}$	1 if the node has no support, else 0: k_{free}^i

Table 3.2: Entries in adjacency and weighted adjacency matrices ($\mathbf{M}_1$, $\mathbf{M}_2$, $\mathbf{M}_3$, $\mathbf{P}_1$, and $\mathbf{P}_2$) for bracing direction optimization of lattice shells

$m_{1_{i,\mathrm{j}}}$	1 if there is a beam element connecting nodes i and j ($k^e_{\mathrm{beam}} = 1$), else 0
$m_{2_{i,\mathrm{j}}}$	1 if there is a truss element connecting nodes i and j ($k^e_{\mathrm{truss}} = 1$), else 0
$m_{3_{i,\mathrm{j}}}$	$m_{1_{i,\mathrm{j}}} + m_{2_{i,\mathrm{j}}}$
$p_{1_{i,j}}$	a ratio between the bending moment and the axial force of an element if there is a beam element connecting nodes i and j, else 0 where $p_{1_{i,j}} = (k^e_{\mathrm{beam}}\, b'_{ei}/(a'_e + 1)$
$p_{2_{i,j}}$	an axial force of an element if there is a truss element connecting nodes i and j, else 0 where $p_{2_{i,j}} = k^e_{\mathrm{truss}} a'_e$

3.3.2 Agent

In this optimization problem, policy and value functions (i.e., *actor* and *critic* networks of a DDPG agent) are made of GCN layers, explained in Chapter 2. In some GCN layers, weighted adjacency matrices representing internal forces replace normalized adjacency matrices. The output of a GCN layer is transformed by activation functions. In general, the ReLU activation function is applied to all layers of both policy and value functions except the last layer of the policy function, where the Sigmoid activation function is used to scale the output to be within [0,1] for predicting probability-based output (i.e., probability of taking action A_t^i in a state S_t).

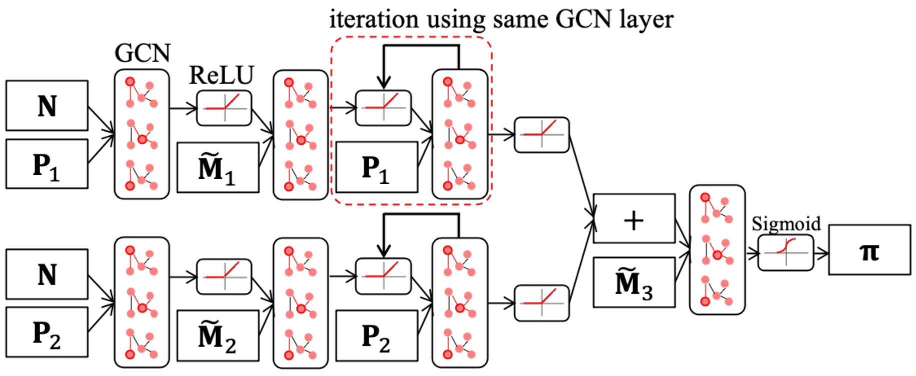
The policy function $\boldsymbol{\pi}$ takes state data ($\mathbf{N}$, $\widetilde{\mathbf{M}}_1$, $\widetilde{\mathbf{M}}_2$, $\widetilde{\mathbf{M}}_3$, $\mathbf{P}_1$, $\mathbf{P}_2$), explained in Section 3.2.1, as input to compute the output $\boldsymbol{\pi} \in \mathbb{R}^{n\times 100}$. This output, having the same number of rows as those of the node feature matrix $\mathbf{N} \in \mathbb{R}^{n\times 5}$, is interpreted as the modifications of bracing directions. Note that the number of columns (i.e., embedded node signal) in $\boldsymbol{\pi}$ should be sufficient to represent the probability of brace direction modification in the lattice shell. In this research, this number is 100.

Value function Q computes an estimation of the accumulated reward (i.e., Q-value) using state data and the output from the policy function ($\boldsymbol{\pi}$) as input. To compute the Q-value, another matrix denoted as $\mathbf{M}_\pi \in \mathbb{R}^{n\times n}$ is computed from the output of the policy function $\boldsymbol{\pi} \in \mathbb{R}^{n\times 100}$, multiplied element-wise with $\mathbf{M}_2$ to exclude non-bracing elements, and normalized within the range [0,1] through dividing by 100 as $\mathbf{M}_\pi = (\boldsymbol{\pi} \cdot \boldsymbol{\pi}^{\mathrm{T}}) \odot \mathbf{M}_2/100$, where $\odot$ is the element-wise product. The value function output should be a scalar representing the estimation of accumulated reward but outputs of GCN layers are matrices. Therefore, the GSP operation and MLP are utilized for transforming the output matrix of the last GCN layer of the value function into a scalar value and estimating the accumulated reward (i.e., Q-value $\in \mathbb{R}^{1\times 1}$), respectively.

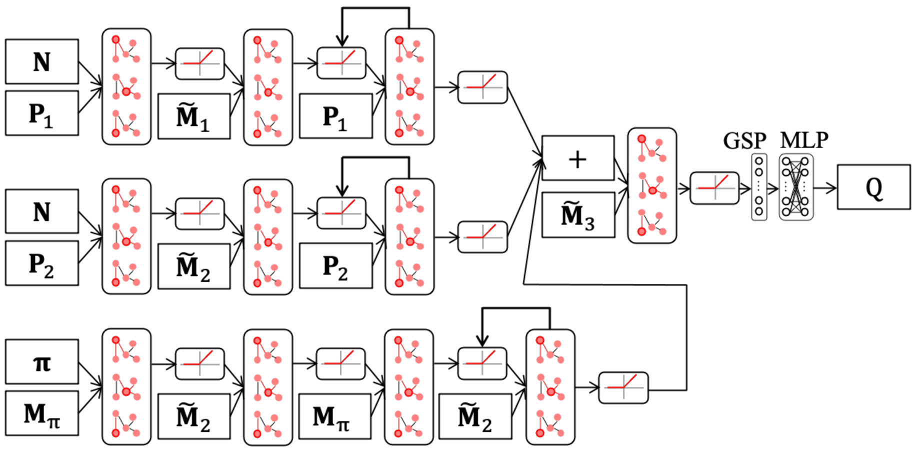
Table 3.3 summarizes the computation processes of the policy and value functions of the GCN-DDPG agent for bracing direction optimization. In each column, the first row indicates whether the computation belongs to the policy or the value function. The second row denotes the input data used for the computation. The third row indicates the computation process using GCN layers, GSP operation, and MLP, as explained in Section 2.5 in Chapter 2. The last row denotes the output of the function. Figure 3.4 illustrates actor and critic networks in this optimization process, utilizing symbols in Figure 2.6 in Chapter 2.

Table 3.3: Policy and value functions of GCN-DDPG for bracing direction optimization of lattice shells

policy function $\boldsymbol{\pi}$	value function Q
input: $\mathbf{N}$, $\widetilde{\mathbf{M}}_1$, $\widetilde{\mathbf{M}}_2$, $\widetilde{\mathbf{M}}_3$, $\mathbf{P}_1$, $\mathbf{P}_2$	input: $\mathbf{N}$, $\widetilde{\mathbf{M}}_1$, $\widetilde{\mathbf{M}}_2$, $\widetilde{\mathbf{M}}_3$, $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{M}_2$, $\boldsymbol{\pi}$
computation: step 1: $\mathbf{N}_{1.1} = \mu\left(\mu(\mathbf{N}, \mathbf{P}_1), \widetilde{\mathbf{M}}_1\right)$ $\mathbf{N}_{2.1} = \mathrm{iter}^2[\mu(\mathbf{N}_{1.1}, \mathbf{P}_1)]$ step 2: $\mathbf{N}_{1.2} = \mu\left(\mu(\mathbf{N}, \mathbf{P}_2), \widetilde{\mathbf{M}}_2\right)$ $\mathbf{N}_{2.2} = \mathrm{iter}^2[\mu(\mathbf{N}_{1.2}, \mathbf{P}_2)]$ step 3: $\mathbf{N}_3 = \mathbf{N}_{2.1} + \mathbf{N}_{2.2}$ $\boldsymbol{\pi} = \sigma\left(\mathbf{N}_3, \widetilde{\mathbf{M}}_3\right)$	computation: step 1: $\mathbf{N}_{1.1} = \mu\left(\mu(\mathbf{N}, \mathbf{P}_1), \widetilde{\mathbf{M}}_1\right)$ $\mathbf{N}_{2.1} = \mathrm{iter}^2[\mu(\mathbf{N}_{1.1}, \mathbf{P}_1)]$ step 2: $\mathbf{N}_{1.2} = \mu\left(\mu(\mathbf{N}, \mathbf{P}_2), \widetilde{\mathbf{M}}_2\right)$ $\mathbf{N}_{2.2} = \mathrm{iter}^2[\mu(\mathbf{N}_{1.2}, \mathbf{P}_2)]$ step 3: $\mathbf{M}_\pi = (\boldsymbol{\pi} \cdot \boldsymbol{\pi}^{\mathrm{T}}) \odot \mathbf{M}_2/100$ step 4: $\mathbf{N}_{1.3} = \mu\left(\mu(\boldsymbol{\pi}, \mathbf{M}_\pi), \widetilde{\mathbf{M}}_2\right)$ $\mathbf{N}_{2.3} = \mathrm{iter}^2\left[\mu\left(\mu(\mathbf{N}_{1.3}, \mathbf{M}_\pi), \widetilde{\mathbf{M}}_2\right)\right]$ step 5: $\mathbf{N}_3 = \mathbf{N}_{2.1} + \mathbf{N}_{2.2} + \mathbf{N}_{2.3}$ $\mathbf{N}_4 = \mu\left(\mathbf{N}_3, \widetilde{\mathbf{M}}_3\right)$ step 6: $\mathbf{n}_5 = \mathrm{GSP}(\mathbf{N}_4)$ step 7: $\mathrm{Q} = f_{\mathrm{NN}}(\mathbf{n}_5)$
output: $\boldsymbol{\pi} \in \mathbb{R}^{n\times 100}$	output: $\mathrm{Q} \in \mathbb{R}^{1\times 1}$


iteration using same GCN layer
GCN
ReLU
N
$\mathbf{P}_1$
$\widetilde{\mathbf{M}}_1$
$\mathbf{P}_1$
N
$\mathbf{P}_2$
$\widetilde{\mathbf{M}}_2$
$\mathbf{P}_2$
+
$\widetilde{\mathbf{M}}_3$
Sigmoid
$\boldsymbol{\pi}$

(a) Actor network


N
$\mathbf{P}_1$
$\widetilde{\mathbf{M}}_1$
$\mathbf{P}_1$
N
$\mathbf{P}_2$
$\widetilde{\mathbf{M}}_2$
$\mathbf{P}_2$
$\boldsymbol{\pi}$
$\mathbf{M}_\pi$
$\widetilde{\mathbf{M}}_2$
$\mathbf{M}_\pi$
$\widetilde{\mathbf{M}}_2$
+
$\widetilde{\mathbf{M}}_3$
GSP
MLP
Q

(b) Critic network

Figure 3.4: Actor and critic networks for bracing direction optimization of lattice shells

3.3.3 Action

Agent optimizes the lattice shell by determining the bracing direction in each grid cell. Similar to the link prediction using GCN for predicting a connection between two nodes in the graph, the direction of the brace is determined from a dot product of each row of the policy function output $\boldsymbol{\pi}$. Since the structural nodes are already represented as graph nodes, the prediction of a link or connection between two structural nodes is equivalent to a truss element (bracing element) that connects those nodes.

Figure 3.5 illustrates a 4-node structure in a grid cell with two possible diagonal braces that connect node i to node j and node n to node m, respectively (Figure 3.5(a)). The values of l_{ij} and

l_{nm} are dot products of the pairs of rows (i, j) and rows (n, m) of the policy function output, respectively. Note that $l_{ij} = l_{ji}$ and $l_{nm} = l_{mn}$ as shown in Figure 3.5(b). Since the optimization problem in Eq. (3.3) allows only one brace in each grid cell, l_{ij} and l_{nm} are compared to determine the bracing direction in the grid cell.

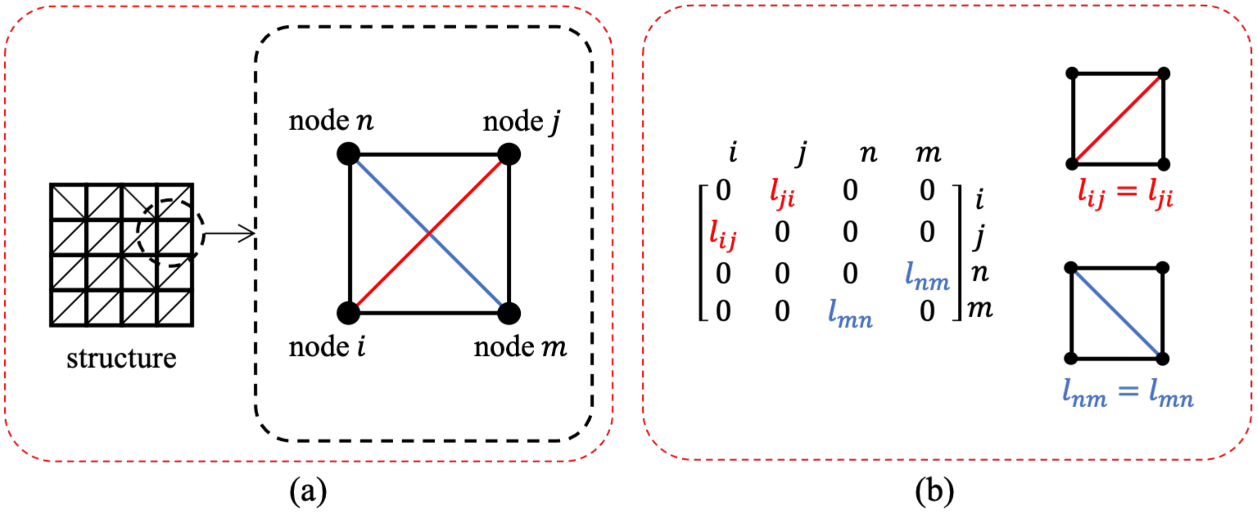

structure
node n
node j
node i
node m
$$\begin{array}{ccccc} i & j & n & m & \\ 0 & l_{ji} & 0 & 0 & i \\ l_{ij} & 0 & 0 & 0 & j \\ 0 & 0 & 0 & l_{nm} & n \\ 0 & 0 & l_{mn} & 0 & m \end{array}$$
$l_{ij} = l_{ji}$
$l_{nm} = l_{mn}$
(a)
(b)

Figure 3.5: Bracing directions and associate link predictions in a grid cell for bracing direction optimization of lattice shells

From the output of the policy function $\boldsymbol{\pi} \in \mathbb{R}^{n\times 100}$, the brace in grid cell k is determined as

$$\mathrm{A}_t^k = \begin{cases} (c_{1,k}, c_{2,k}) = (1, 0) & : \text{if } l_{ij} > l_{mn} \\ (c_{1,k}, c_{2,k}) = (0, 1) & : \text{if } l_{ij} < l_{mn} \end{cases} \tag{3.13}$$

$$l_{ij} = l_{ji} = \sum_{v=1}^{100} \pi_{i,v}\pi_{j,v} \tag{3.14}$$

$$l_{mn} = l_{nm} = \sum_{v=1}^{100} \pi_{n,v}\pi_{m,v} \tag{3.15}$$

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

3.3.4 Reward

The reward is calculated from the difference of the strain energies measured before and after the agent does the action in each optimization step to motivate the agent to do actions that improve the objective function (i.e., reducing the strain energy). At step t, the reward signal R_{t+1}, which the agent receives after doing action A_t in a state S_t, is computed from the change of the total strain energy as

$$\mathrm{R}_{t+1} = (E_t - E_{t+1})/E_0 \tag{3.16}$$

where E_t and E_0 are the value of total strain energy of the structure at step t and the initial structure, respectively.

3.3.5 Numerical examples

The agent is first trained to optimize the structure in the *training phase* to assess the ability to improve its performance, utilizing small structures. In the *test phase*, the agent performance is evaluated on larger structural configurations than those used in training. In the test phase, results obtained by the proposed method are compared with those obtained by the enumeration method (EM) and GA; EM is utilized for benchmarking problems when computing all possible solutions is feasible, and GA is utilized for the benchmark when it is not feasible to compute all possible solutions because of the large search space. The number of feature maps h in all GCN layers, explained in Section 2.5 in Chapter 2, is 200. MLP has two hidden layers, each consisting of 200 neurons.

The 12-DoF frame element has a hollow cylindrical section with an external and internal diameter of 100 mm and 90 mm, respectively. The 6-DoF truss element has a solid circular section with a diameter of 43.6 mm. Both elements have Young's modulus of 205 kN/mm^2 and a same weight of 12 kg/m. All structural nodes are subjected to a vertical downward point load of 10 kN. The program is implemented using Python 3.6 environment. A PC with CPU Intel Core i5-6600 (3.3 GHz, 4 cores) and GPU AMD Radeon R9 M395 2 GB is utilized for computation.

Figure 3.6 illustrates the algorithm flowcharts for the training and test phases. During the optimization process, each MDP is denoted as a step. The loop of MDPs, defined as a *game* or an *episode*, is terminated when the specified number of steps is reached.

The algorithm in the training phase (Figure 3.6(a)) is as follows; (0) The finite element model of the lattice shell is initialized with random bracing direction in each grid. (1) Information of the structure is represented using graph representation. (2) Graph representation becomes state data. (3) The agent takes graph representation as input and computes the action. (4) Bracing directions in the lattice shell are modified according to the action. The finite element analysis is performed to obtain the objective function value and structural responses. The reward is computed from the change of the objective function value. (5) The agent is trained by collected state-action-reward data and the parameters in the agent are updated. (6) If the final step is not reached go to step (1), if the final step is reached end and save data.

Once the agent has undergone training for a certain number of episodes, it is then applied to optimize a lattice shell using the algorithm during the test phase, as shown in Figure 3.6(b). The process can be described as follows; (0) After initialization, the finite element model of the lattice shell is set up with random bracing directions at each grid. (1) The structural information is represented as a graph. (2) This graph representation is then converted into the state data. (3) The agent, using this state data as input, proceeds to calculate an action. (4) Subsequently, based on the action, adjustments are made to the bracing directions in the grid cells of the lattice shell. The finite element analysis is performed. If the objective function is improved, then the structure is modified; otherwise, the lattice shell is reverted to the configuration before the action. (5) If the final step is reached, the process concludes, and the data is saved; If not, it loops back to (1).

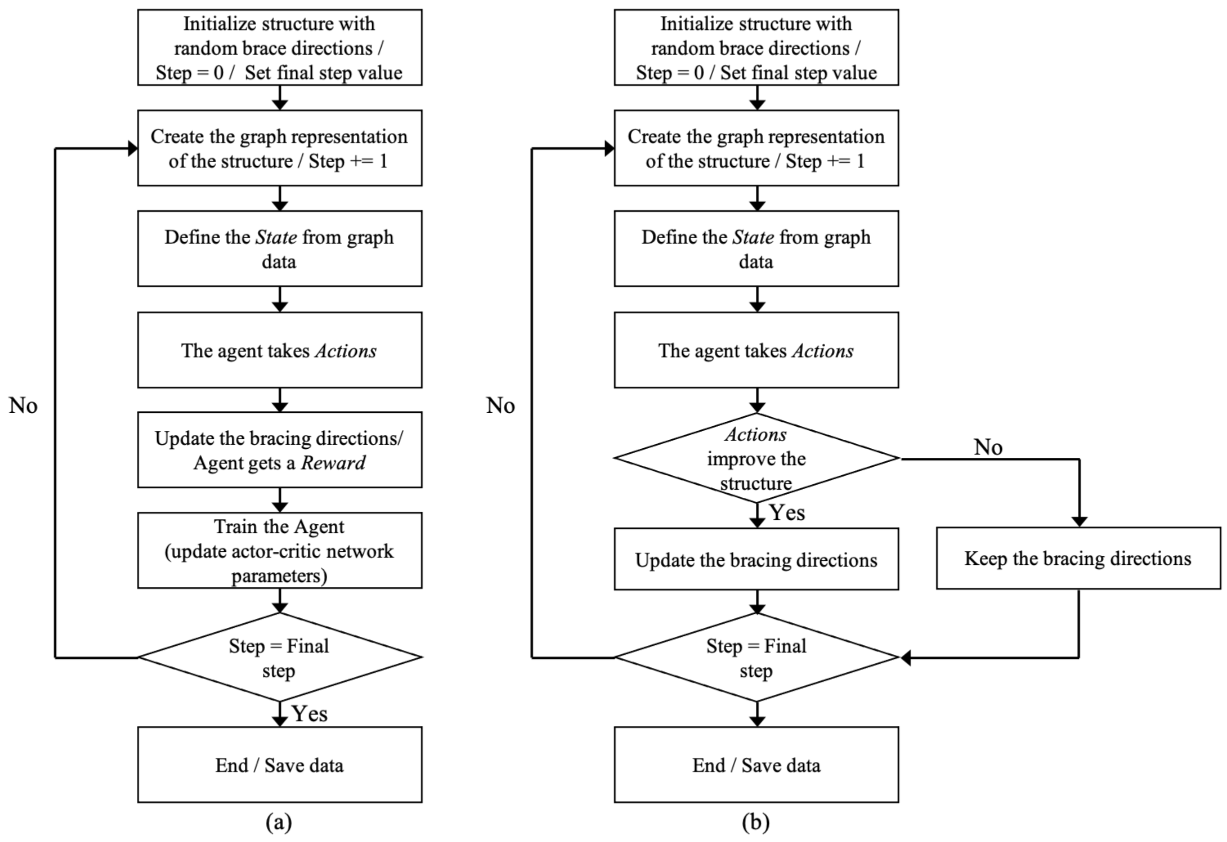

Initialize structure with random brace directions / Step = 0 / Set final step value
Create the graph representation of the structure / Step += 1
Define the *State* from graph data
The agent takes *Actions*
Update the bracing directions/ Agent gets a *Reward*
Train the Agent (update actor-critic network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

Figure 3.6: Flowcharts of the algorithm for bracing direction optimization of lattice shells; (a) Training phase, (b) Test phase

3.3.5.1 Training phase

Training is carried out on lattice shell structures with 4×4 grids and diagonal truss braces. Each grid cell has dimensions of 1.0 m by 1.0 m. In each game of training, a model is initialized for each game with supports assigned to either of the two pre-determined dome-shaped structures indicated in Figure 3.7. The maximum nodal height is 1 m and the minimum is 0. The brace directions are randomly initialized at every game.

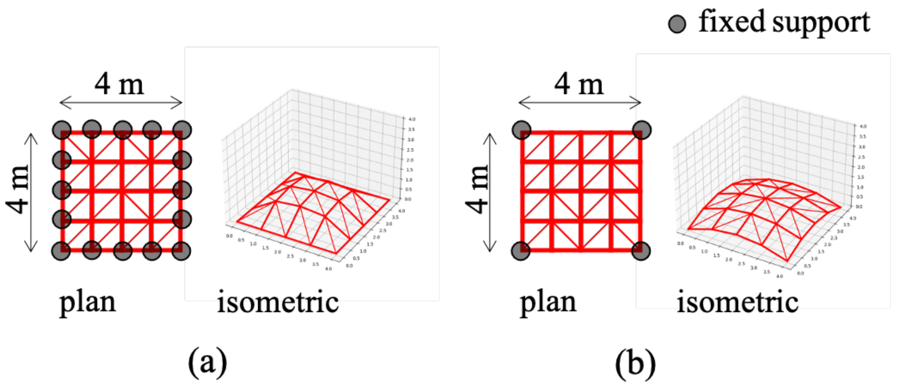


Figure 3.7: Structural models for bracing direction optimization during training phase; (a) Support condition 1, (b) Support condition 2.

The number of steps in each game is 200. The agent surrogate functions, as explained in Section 2.3.3 in Chapter 2, are adjusted using the Adam optimizer, as explained in Section 2.5.3 in Chapter 2. The mini-batch size is set to 32 and the learning rates are 10^{-7} and 10^{-6} for policy and value functions, respectively. Note that learning rates are reduced by a factor of β=0.1 every 200 games (20000 steps) to stabilize the training process. Note that the details of the training process are explained in Section 2.5.2 in Chapter 2. Weights and biases of the surrogate functions are synchronized with those of the online functions every 100 steps using τ=0.05, as explained in Section 2.3.3 in Chapter 2. The agent is trained for 1000 games. In the value function, the MSE is the loss function.

Figure 3.8 shows the cumulative reward obtained during the training phase, which indicates the ability of the agent to improve its performance. In this figure, the vertical and horizontal axes represent the cumulative reward obtained during training, and the game number, respectively. Since support and bracing directions are changed at every training game, there are fluctuations in the obtained rewards. However, the moving average with a window size set to 50 of these rewards, indicated as the thick red line, increases during the first 200 games and then remains stable around a certain value. This indicates that the agent has the learning capability to optimize structural configurations of braces in different topology and support conditions. The number of structural analyses carried out with GCN-DDPG in the training phase is equal to the number of training games multiplied by the number of steps in each training game, which is 100000.

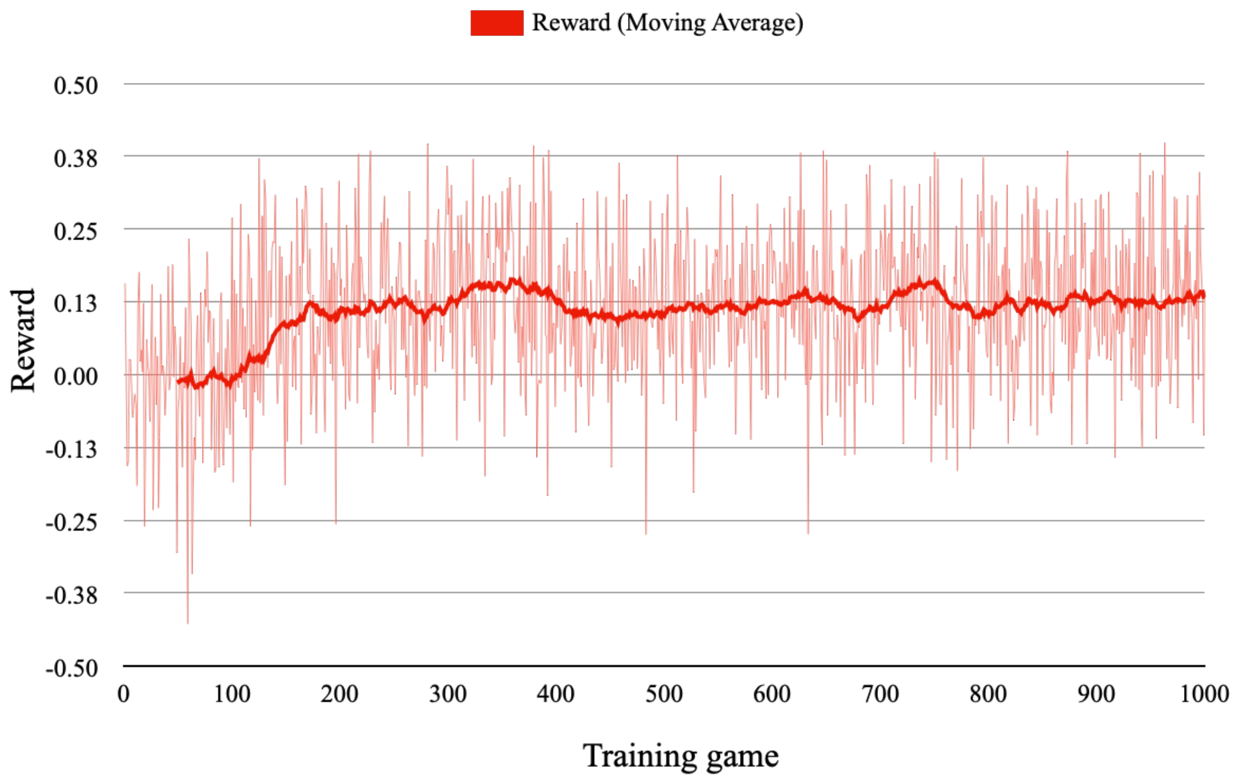


Figure 3.8: Variation of reward and its moving average in training phase for bracing direction optimization of lattice shells

3.3.5.2 Test phase

To investigate the capability of the agent on structures that have not been included in the training phase, the trained agent is applied to topology optimization using four lattice shells with pre-determined geometries and support conditions denoted as 1, 2, 3, and 4 (Figures 3.9-3.12). Each of these support conditions also has eight sizes of lattice shells including 4×4, 4×6, 6×6, 4×10, 10×6, 10×10, and 20×20-grids, and the bracing directions are initialized randomly. The agent optimizes each structural model 10 times and the number of optimization steps for the test phase is 100.

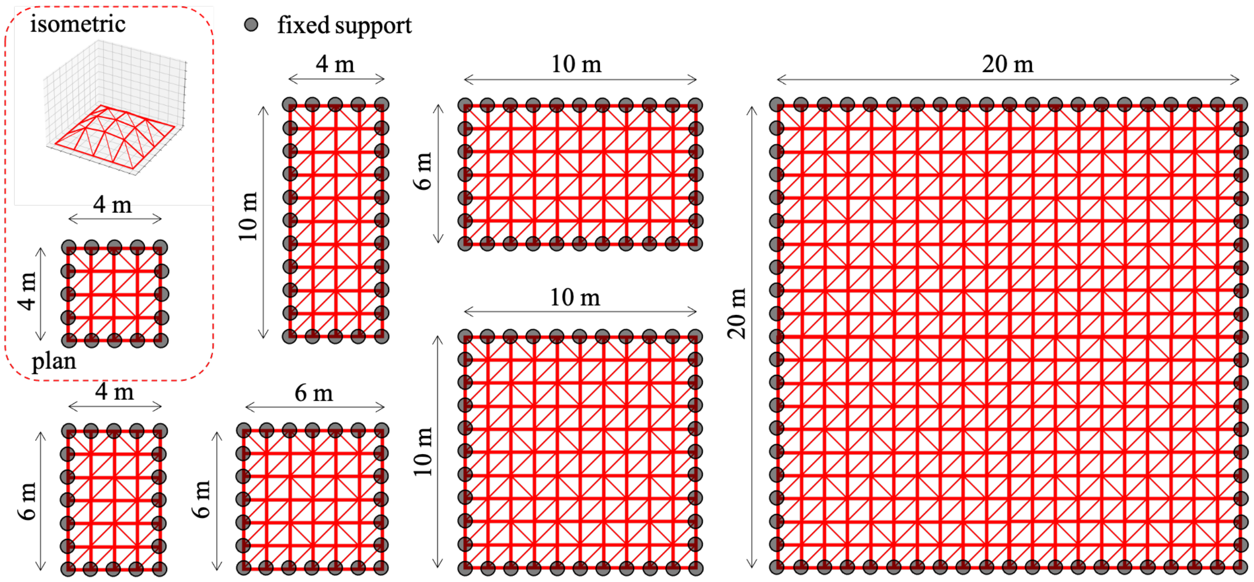


Figure 3.9: Structural models for test phase for bracing direction optimization of lattice shells – support condition 1

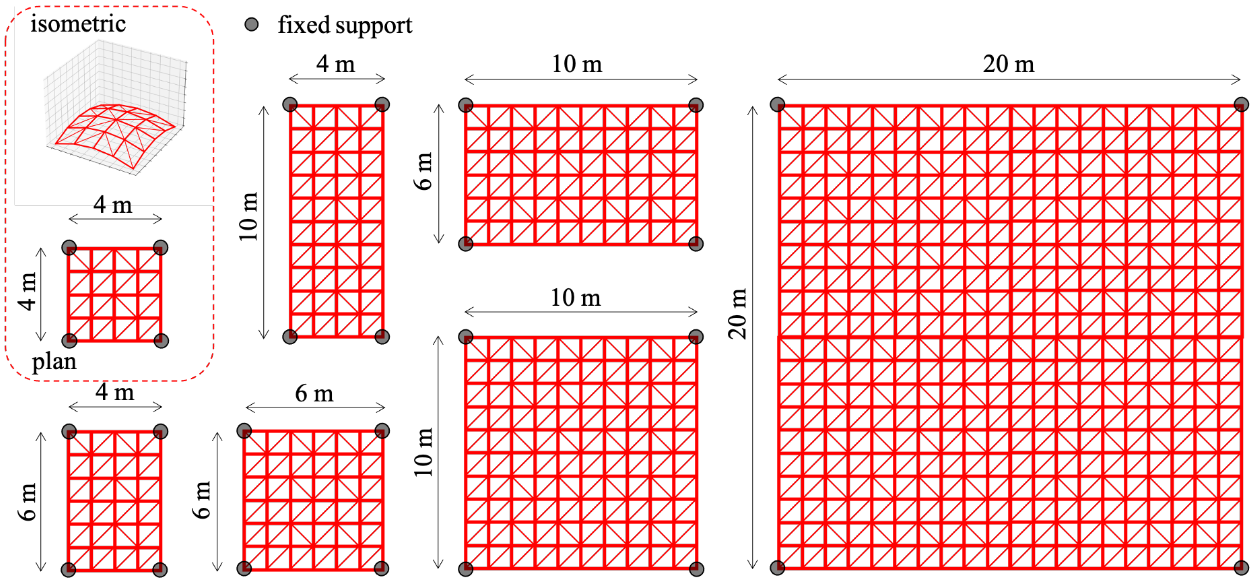


Figure 3.10: Structural models for test phase for bracing direction optimization of lattice shells – support condition 2

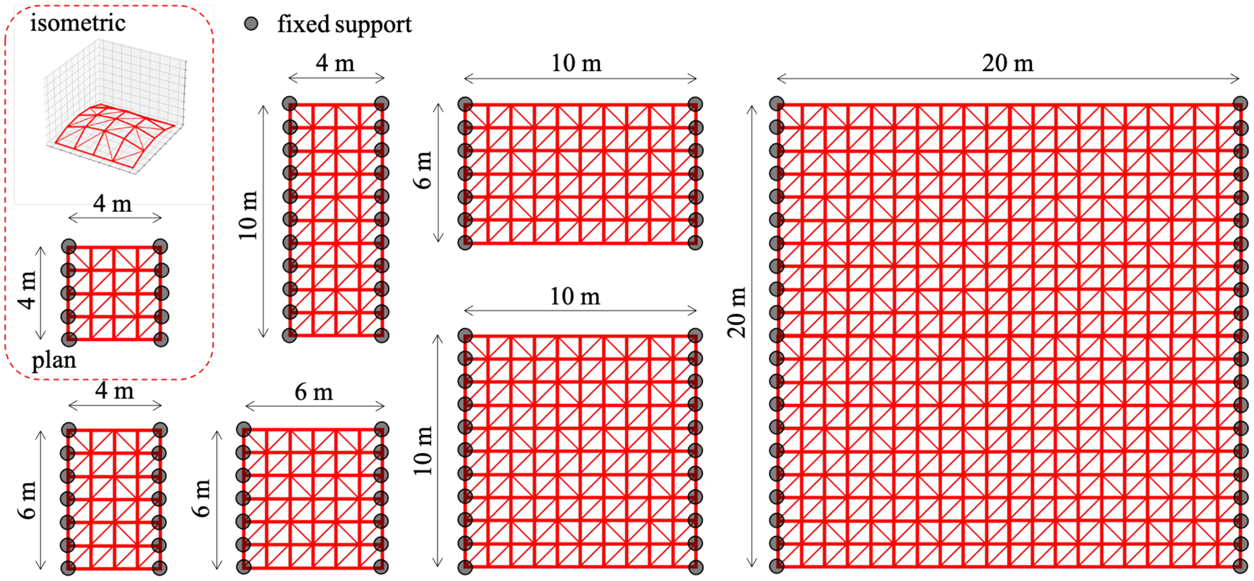


Figure 3.11: Structural models for test phase for bracing direction optimization of lattice shells – support condition 3

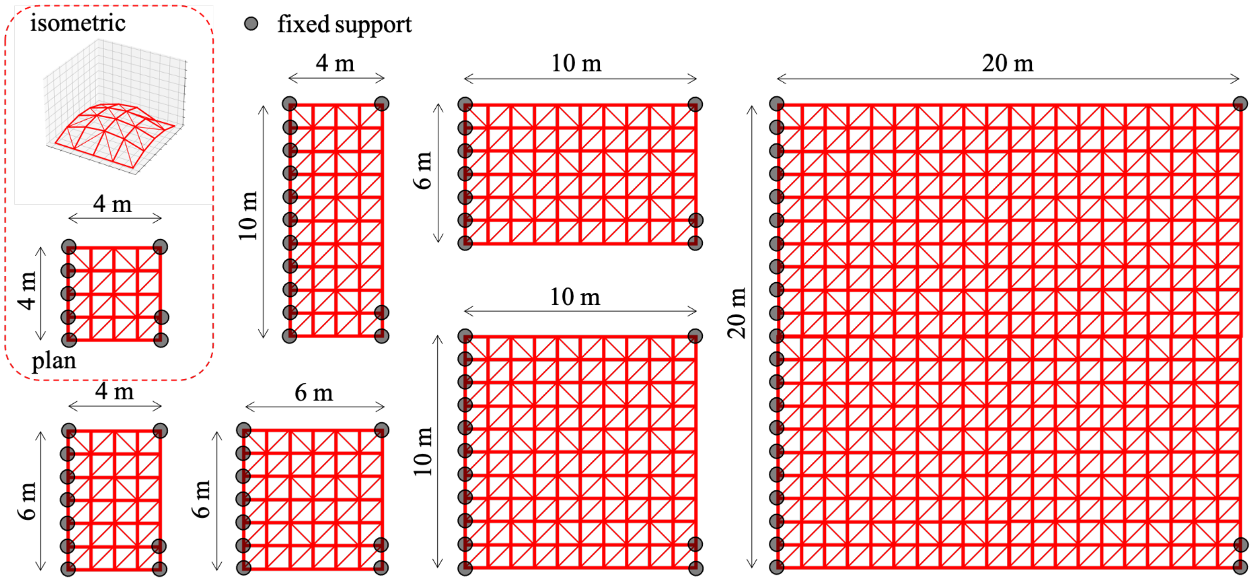


Figure 3.12: Structural models for test phase for bracing direction optimization of lattice shells – support condition 4

The minimum (min.), mean, and standard deviation (std.) of the strain energy, and mean energy reduction rate (reduction) of each structural model are shown in Table 3.4. Note that the reduction rate is $(E_0 - E_*)/E_0$ where E_* is the final strain energy of the lattice shells. From this table, the trained agent improves lattice shells by reducing the strain energy by 5-25%, depending on the structure size and support conditions. In all ten trials of test structures, the standard deviations are low compared with the mean, indicating a good convergence property.

Table 3.4: Test results for bracing direction optimization of lattice shells

support conditions	grid size	min. (N·m)	mean (N·m)	std. (N·m)	reduction (%)
1	4×4	4.38	4.53	0.19	16.83
	4×6	15.71	16.01	0.38	13.21
	6×6	63.15	67.40	2.56	18.10
	4×8	72.49	73.72	0.69	5.55
	10×6	450.47	459.92	7.89	12.89
	10×10	2242.26	2288.05	40.79	19.80
	20×20	213715.82	217120.90	1742.84	10.08
2	4×4	77.88	89.35	10.02	19.94
	4×6	284.81	290.09	3.98	20.21
	6×6	818.14	896.06	46.93	19.71
	4×10	2124.56	2142.64	17.10	20.76
	10×6	4642.28	4713.70	37.63	19.79
	10×10	17683.39	17966.80	181.50	24.36
	20×20	1275569.06	1312975.69	42317.86	19.66
3	4×4	32.11	32.78	0.53	6.43
	4×6	70.09	72.40	1.44	5.29
	6×6	345.73	356.54	6.92	12.59
	4×10	193.91	198.18	2.84	6.69
	10×6	2841.17	2869.52	18.50	10.95
	10×10	8219.87	8333.38	101.09	12.28
	20×20	589997.91	596066.09	4460.72	9.38
4	4×4	40.81	41.34	0.46	17.71
	4×6	156.09	161.09	3.61	22.68
	6×6	527.56	535.43	5.95	17.30
	4×10	1192.81	1213.16	11.94	21.76
	10×6	3466.03	3542.59	47.95	11.63
	10×10	12629.96	12762.32	109.98	16.49
	20×20	920936.67	931040.36	8426.17	16.35

3.3.5.3 Comparison of computation cost and performance with EM and approximate GA

Results obtained by the RL agent are compared to benchmarks. For 4×4-grid structures, global optimal solutions are obtained using EM by generating 2^k combinations of bracing directions for k grid cells. For structures with a greater number of grids, EM is not feasible and GA is utilized to obtain approximate optimal solutions. GA is a population-based method mimicking the process of natural evolution which can be utilized for solving combinatorial optimization problems. The important operations in GA are *selection* to transfer superior solutions from one generation to the next generation, *crossover* to generate new diverse solutions, and *mutation* to modify solutions with a certain probability in order to avoid convergence to local minima. In this topology optimization problem, bracing directions in grid cells are represented as binary strings. The GA algorithm in the Python library named DEAP (Distributed Evolutionary Algorithms in Python) [128] is used. The number of population and generation of GA are determined based on the feasibility of computational cost. In the following examples, the numbers of population and generations are 50 and 100, respectively. Note that a comparison is made to benchmark both the performance and computational efficiency of GCN-DDPG and GA for the preliminary design process of lattice shells where several configurations are to be evaluated. Table 3.5 shows the computational cost of GCN-DDPG (test phase), EM, and GA for each problem and the size of the global stiffness matrix.

In Table 3.6, the ratio of the difference between the minimum strain energy solution obtained by RL and that obtained by GA (or EM) is shown in the column labeled as 'diff', computed as

$$\begin{aligned} \text{diff} &= (\min(\text{Result}_{\text{RL}}) - \text{Result}_{\text{GA}})/\text{Result}_{\text{GA}} \\ \text{diff} &= (\min(\text{Result}_{\text{RL}}) - \text{Result}_{\text{EM}})/\text{Result}_{\text{EM}} \end{aligned} \tag{3.17}$$

From Tables 3.5 and 3.6, the solution quality and the efficiency of the GCN-DDPG agent can be assessed. In most cases, results obtained by the GCN-DDPG agent are comparable to those obtained by EM and GA with less than 10% difference using less computational cost. In this problem, the proposed method may not be ideal for finding the most optimal structure, however, once the trained agent is obtained, the method is useful in the preliminary design process, which typically requires testing several structures, and efficiency in computational cost is desired. Note that the structure with a large size of global stiffness matrix requires a long computing time for solving the linear analysis problem. Therefore, the GCN-DDPG is significantly more efficient than GA when applied to the bracing direction optimization of large structures.

Figure 3.13 illustrates the initial brace topology, final brace topology, and the change of strain energy of GCN-DDPG best results for 6×6-grid structural models. Although the structural configurations differ considerably from those used in the training phase in terms of the support conditions and the size, the agent is capable of minimizing the strain energy by adjusting the bracing directions. From Figures 3.13(a) and 3.13(c), where the support locations are symmetric, the agent obtains solutions with symmetric layouts, despite the fact that the symmetry condition feature is not explicitly imposed.

Table 3.5: Total computational cost of structural analysis of each method for bracing direction optimization of lattice shells

support conditions	grid size	size of the global stiffness matrix $n_\mathrm{D} \times n_\mathrm{D}$	number of structural analyses (times)		
			GCN-DDPG (test phase)	benchmarks	
				EM	GA
1	4×4	54×54	1000	65536	5000
	4×6	90×90	1000	-	5000
	6×6	150×150	1000	-	5000
	4×10	198×198	1000	-	5000
	10×6	270×270	1000	-	5000
	10×10	486×486	1000	-	5000
	20×20	2166×2166	1000	-	5000
2	4×4	126×126	1000	65536	5000
	4×6	186×186	1000	-	5000
	6×6	270×270	1000	-	5000
	4×10	306×306	1000	-	5000
	10×6	438×438	1000		5000
	10×10	702×702	1000	-	5000
	20×20	2622×2622	1000	-	5000
3	4×4	90×90	1000	65536	5000
	4×6	138×138	1000	-	5000
	6×6	210×210	1000	-	5000
	4×10	234×234	1000	-	5000
	10×6	378×378	1000	-	5000
	10×10	594×594	1000	-	5000
	20×20	2394×2394	1000	-	5000
4	4×4	102×102	1000	65536	5000
	4×6	150×150	1000	-	5000
	6×6	234×234	1000	-	5000
	4×10	246×246	1000	-	5000
	10×6	402×402	1000	-	5000
	10×10	642×642	1000	-	5000
	20×20	2502×2502	1000	-	5000

Table 3.6: Comparison of results obtained by GCN-DDPG (test phase) and benchmarks for bracing direction optimization of lattice shells

support conditions	grid size	min (N·m)	method	benchmarks (N·m)	diff
1	4×4	4.38	EM / GA	4.38 / 4.38	0.00
	4×6	15.71	GA	15.66	0.00
	6×6	63.15		61.55	0.03
	4×10	72.49		71.88	0.01
	6×10	450.47		422.35	0.07
	10×10	2242.26		2135.28	0.05
	20×20	213715.82		202566.99	0.06
2	4×4	77.88	EM / GA	77.88 / 77.88	0.00
	4×6	284.81	GA	279.67	0.02
	6×6	818.14		777.99	0.05
	4×10	2124.56		2046.13	0.04
	6×10	4642.28		4342.47	0.07
	10×10	17683.39		16294.39	0.09
	20×20	1275569.06		1170860.54	0.09
3	4×4	32.11	EM / GA	31.59 / 31.59	0.02
	4×6	70.09	GA	66.69	0.05
	6×6	345.73		342.38	0.01
	4×10	193.91		182.56	0.06
	6×10	2841.17		2725.89	0.04
	10×10	8219.87		7697.87	0.07
	20×20	589997.91		553614.84	0.07
4	4×4	40.81	EM / GA	40.77 / 40.77	0.00
	4×6	156.09	GA	150.95	0.03
	6×6	527.56		485.53	0.09
	4×10	1192.81		1140.11	0.05
	6×10	3466.03		3150.67	0.10
	10×10	12629.96		11462.31	0.10
	20×20	920936.67		850282.94	0.08

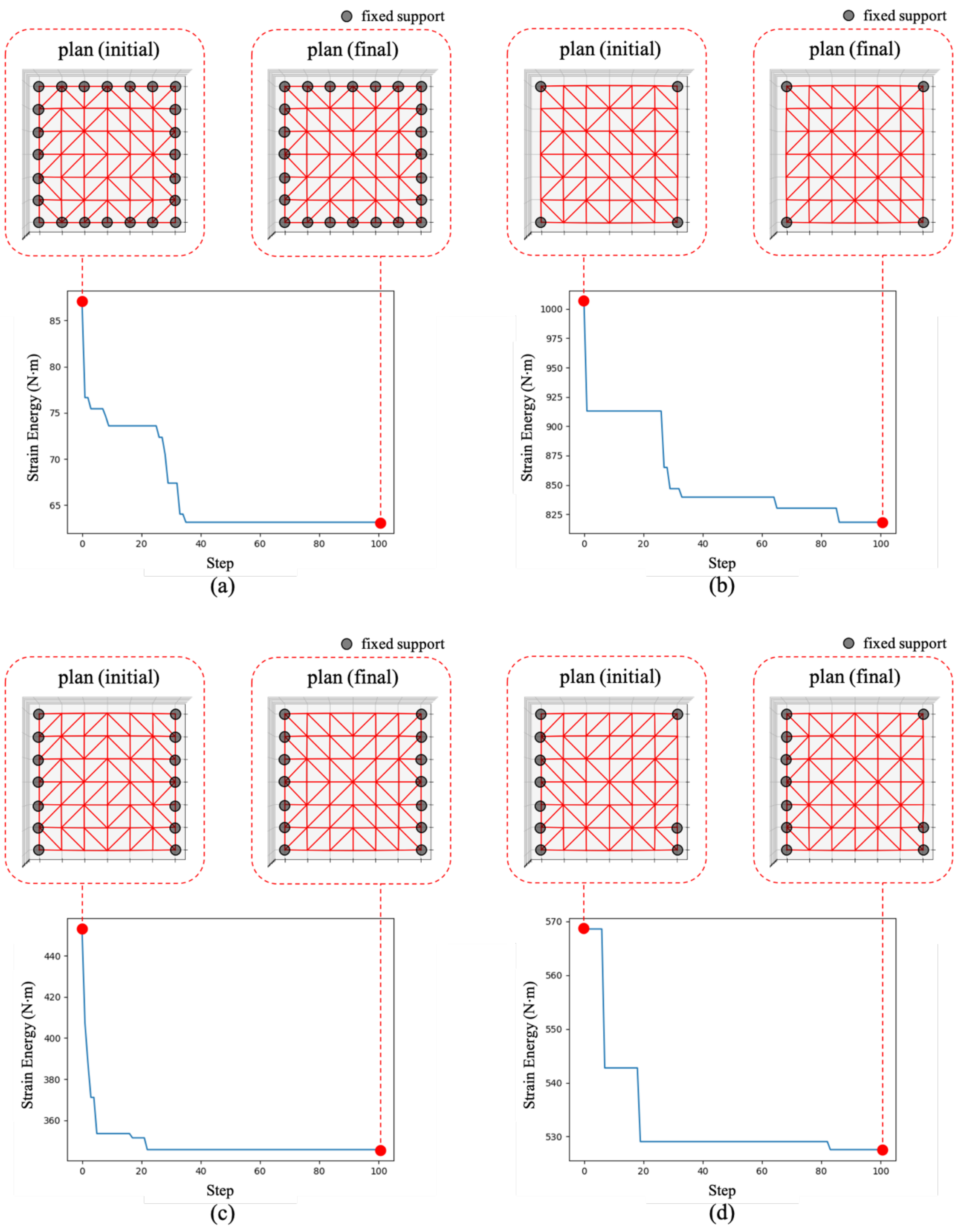
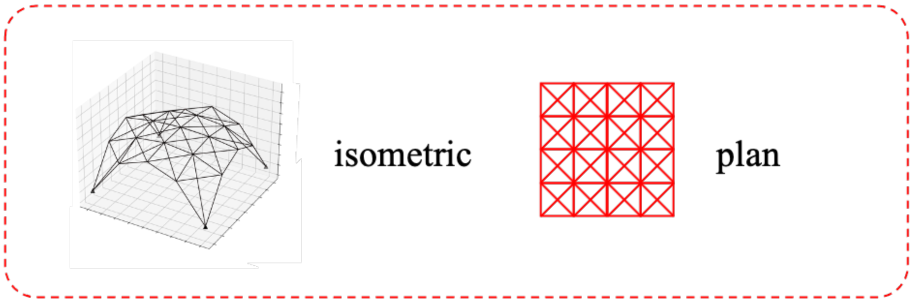


Figure 3.13: GCN-DDPG result of the 6×6-grid structure in the test phase for bracing direction optimization of lattice shells; (a) Support condition 1, (b) Support condition 2, (c) Support condition 3, (d) Support condition 4.

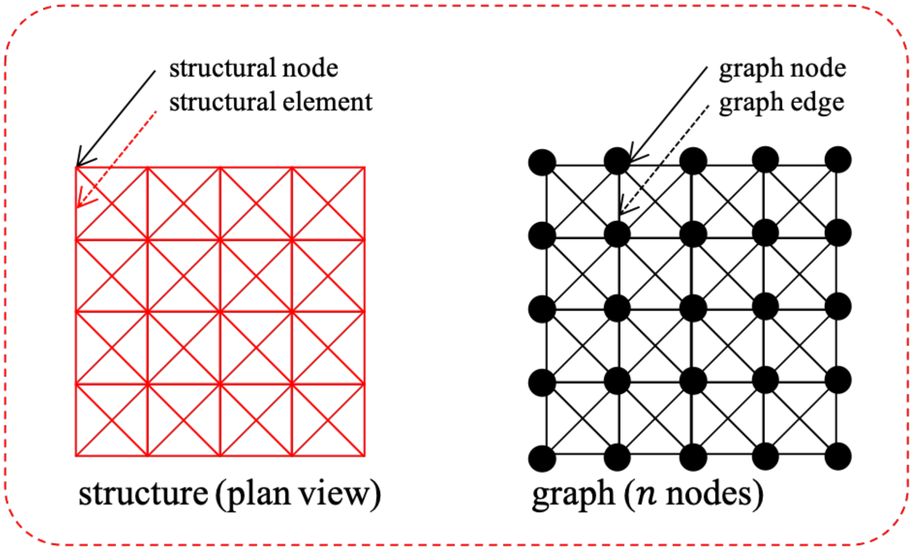
3.4 Sizing optimization of lattice shell using DDPG and GCN

This section introduces the RL method for solving problems in Section 3.1.2. The method is modified from those in Section 3.2. The optimization problem is also formulated into MDPs where the reward is computed from the improvement of the objective function. The structure is represented using graph representation, with slight modification from Section 3.2, as illustrated in Figure 3.14.

These graph representations serve as state data for the RL agent to compute the actions in each optimization step. Figure 3.15 illustrates the MDP in this example. In a discrete timestep t, the agent observes the state S_t which is the graph representation of the structural configuration (Figure 3.15(a)). The agent determines the action A_t which is reducing the cross-sectional indices of elements (Figure 3.15(b)). The structure is modified and structural responses are computed. The agent obtains a reward R_{t+1} based on the changed structure (Figure 3.15(c)) and the new step of MDP begins.



(a)



(b)

$$\mathbf{N} = \left.\overbrace{\begin{bmatrix} n_{1,1} & \cdots & n_{1,f} \\ \vdots & \ddots & \vdots \\ n_{n,1} & \cdots & n_{n,f} \end{bmatrix}}^{f \text{ features}}\right\} n \text{ nodes}$$

Node feature matrix

$$\mathbf{M} = \left.\overbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & \cdots & \vdots \\ 0 & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}}^{n \text{ nodes}}\right\} n \text{ nodes}$$

Adjacency matrix

$$\mathbf{P} = \left.\overbrace{\begin{bmatrix} 0 & p_{1,2} & 0 & 0 \\ p_{2,1} & 0 & \cdots & \vdots \\ 0 & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}}^{n \text{ nodes}}\right\} n \text{ nodes}$$

Weighted adjacency matrix

(c)

Figure 3.14: Graph representation of the lattice shell for sizing optimization of lattice shells

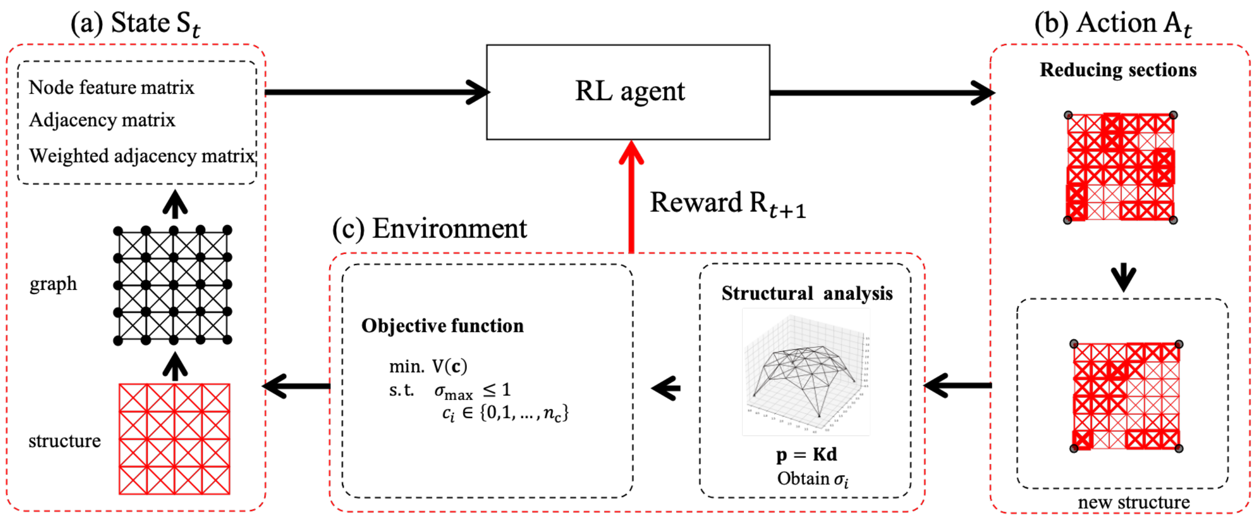


Figure 3.15: Proposed MDP for sizing optimization of lattice shells

3.4.1 State

In this example, the state is the graph representation of the structure consisting of node feature matrix $\mathbf{N}$, adjacency matrix $\mathbf{M}$, weighted adjacency matrix of sectional area $\mathbf{P}_\mathrm{a}$, weighted adjacency matrix of stress ratio $\mathbf{P}_\sigma$, and weighted adjacency matrix of internal forces $\mathbf{P}_\mathrm{f}$. Tables 3.7 and 3.8 show entries in the node feature matrix, and entries in adjacency and weighted adjacency matrices, respectively. Note that all values in these Tables are normalized using the min-max normalization method.

Table 3.7: Entries in node feature matrix (**N**) for sizing optimization of lattice shells

$n_{i,1}$	x-coordinate of node i
$n_{i,2}$	y-coordinate of node i
$n_{i,3}$	z-coordinate of node i
$n_{i,4}$	1 if the node has fixed support; else 0
$n_{i,5}$	1 if the node has no support; else 0

Table 3.8: Entries in adjacency and weighted adjacency matrices ($\mathbf{M}$, $\mathbf{P}_\mathrm{a}$, $\mathbf{P}_\sigma$, and $\mathbf{P}_\mathrm{f}$) for sizing optimization of lattice shells

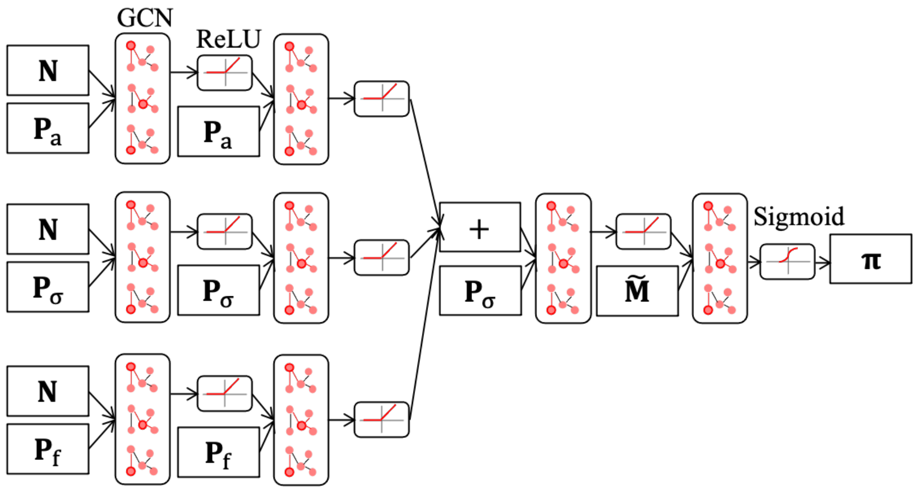
$m_{i,\mathrm{j}}$	1 if there is an element connecting nodes i and j; else 0
$p_{\mathrm{a}_{i,j}}$	sectional area of an element if there is an element connecting nodes i and j; else 0
$p_{\sigma_{i,j}}$	a ratio of internal stress to the yield stress of an element if there is an element connecting nodes i and j; else 0
$p_{\mathrm{f}_{i,j}}$	a ratio of the summation of internal bending forces in the local y and z axes ($\mathrm{f}_2, \mathrm{f}_3$) to the local axial force (f_1) of an element if there is an element connecting nodes i and j; else 0 where $p_{\mathrm{f}_{i,j}} = (\lvert\mathrm{f}_2\rvert + \lvert\mathrm{f}_3\rvert)/(1 + \lvert\mathrm{f}_1\rvert)$

3.4.2 Agent

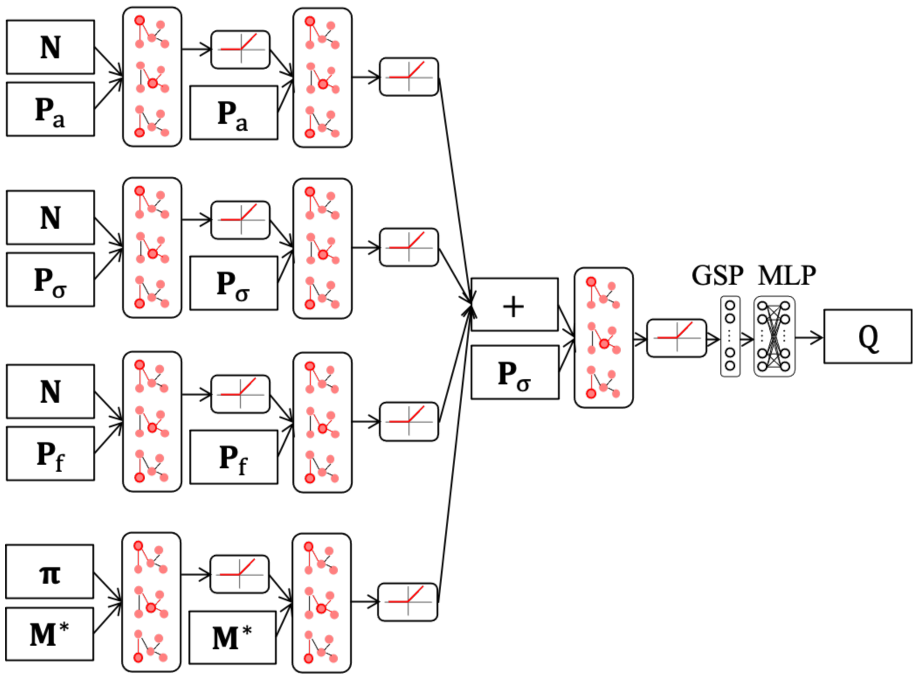
In this optimization problem, actor and critic networks of a DDPG agent are made of GCN layers. The policy function $\boldsymbol{\pi}$ takes state data ($\mathbf{N}$, $\widetilde{\mathbf{M}}$, $\mathbf{P}_\mathrm{a}$, $\mathbf{P}_\sigma$, $\mathbf{P}_\mathrm{f}$), explained in Section 3.3.1, as input to compute the output $\boldsymbol{\pi} \in \mathbb{R}^{n\times 100}$. This output is interpreted to determine which element to be reduced in size. In order to sufficiently represent the probability of modifying the sectional area in each element of the lattice shell, the number of columns (i.e., embedded node signal) in $\boldsymbol{\pi}$ is set to 100. Value function Q computes Q-value utilizing state data and $\boldsymbol{\pi}$ as input. Table 3.9 shows the inputs, computations, and outputs of the policy function (left column) and the value function (right column). Figure 3.16 shows actor and critic networks in this optimization process.

Table 3.9: Policy and value functions of GCN-DDPG for sizing optimization of lattice shells

policy function	value function
input: $\mathbf{N}$, $\widetilde{\mathbf{M}}$, $\mathbf{P}_\mathrm{a}$, $\mathbf{P}_\sigma$, $\mathbf{P}_\mathrm{f}$	input: $\mathbf{N}$, $\widetilde{\mathbf{M}}$, $\mathbf{P}_\mathrm{a}$, $\mathbf{P}_\sigma$, $\mathbf{P}_\mathrm{f}$, $\boldsymbol{\pi}$
computation: step 1: $\mathbf{N}_{1.1} = \mathrm{ReLU}(\mathrm{ReLU}(\mathbf{N}, \mathbf{P}_\mathrm{a}), \mathbf{P}_\mathrm{a})$ $\mathbf{N}_{1.2} = \mathrm{ReLU}(\mathrm{ReLU}(\mathbf{N}, \mathbf{P}_\sigma), \mathbf{P}_\sigma)$ $\mathbf{N}_{1.3} = \mathrm{ReLU}(\mathrm{ReLU}(\mathbf{N}, \mathbf{P}_\mathrm{f}), \mathbf{P}_\mathrm{f})$ step 2: $\mathbf{N}_2 = \mathbf{N}_{1.1} + \mathbf{N} + \mathbf{N}_{1.3}$ step 3: $\boldsymbol{\pi} = \mathrm{Sigmoid}\big(\mathrm{ReLU}(\mathbf{N}_2, \mathbf{P}_\sigma), \widetilde{\mathbf{M}}\big)$	computation: step 1: $\mathbf{N}_{1.1} = \mathrm{ReLU}(\mathrm{ReLU}(\mathbf{N}, \mathbf{P}_\mathrm{a}), \mathbf{P}_\mathrm{a})$ $\mathbf{N}_{1.2} = \mathrm{ReLU}(\mathrm{ReLU}(\mathbf{N}, \mathbf{P}_\sigma), \mathbf{P}_\sigma)$ $\mathbf{N}_{1.3} = \mathrm{ReLU}(\mathrm{ReLU}(\mathbf{N}, \mathbf{P}_\mathrm{f}), \mathbf{P}_\mathrm{f})$ step 2: $\mathbf{M}^* = (\boldsymbol{\pi} \cdot \boldsymbol{\pi}^\mathrm{T}) \odot \widetilde{\mathbf{M}}/100$ $\mathbf{N}_2 = \mathrm{ReLU}(\mathrm{ReLU}(\boldsymbol{\pi}, \mathbf{M}^*), \mathbf{M}^*)$ step 3: $\mathbf{N}_3 = \mathbf{N}_{1.1} + \mathbf{N}_{1.2} + \mathbf{N}_{1.3} + \mathbf{N}_2$ step 4: $\mathbf{N}_4 = \mathrm{ReLU}\big(\mathbf{N}_3, \widetilde{\mathbf{M}}\big)$ step 5: $\mathrm{Q} = \mathrm{f}_{\mathrm{NN}}\big(\mathrm{GSP}(\mathbf{N}_4)\big)$
output: $\boldsymbol{\pi} \in \mathbb{R}^{n\times 100}$	output: $\mathrm{Q} \in \mathbb{R}^{1\times 1}$



(a) Actor network



(b) Critic network

Figure 3.16: Actor and critic networks for sizing optimization of lattice shells

3.4.3 Action

The agent can make decision to reduce the sectional size of multiple elements simultaneously by interpreting the normalized output of the policy function $\boldsymbol{\pi} \rightarrow \boldsymbol{\pi}/100$. In this interpretation, each sectional size c_i of an element i that connects nodes p and q is determined as

$$c_i = \begin{cases} \min(0, c_i - 1) & : \text{if } \sum_{u=1}^{100} \pi_{p,u}\, \pi_{q,u} > threshold_1 \\ c_i & : \text{if } \sum_{u=1}^{100} \pi_{p,u}\, \pi_{q,u} \leq threshold_1 \end{cases} \tag{3.18}$$

where $threshold_1 = 0.3$.

3.4.4 Reward

The agent obtains a reward R_{t+1} after modifying the structure by the action A_t in a state S_t in an optimization step t. R_{t+1} is obtained from the change of the structural volume as follows:

$$\mathrm{R}_{t+1} = \begin{cases} (V_t - V_{t+1})/V_0 & : \text{if} \quad \sigma_{i\,\max} \leq 1 \\ -1 & : \text{if} \quad \sigma_{i\,\max} > 1 \text{ or } t = t_{\max} \end{cases} \tag{3.19}$$

where V_t and V_0 are the structural volume at step t and the initial step, respectively.

3.4.5 Numerical examples

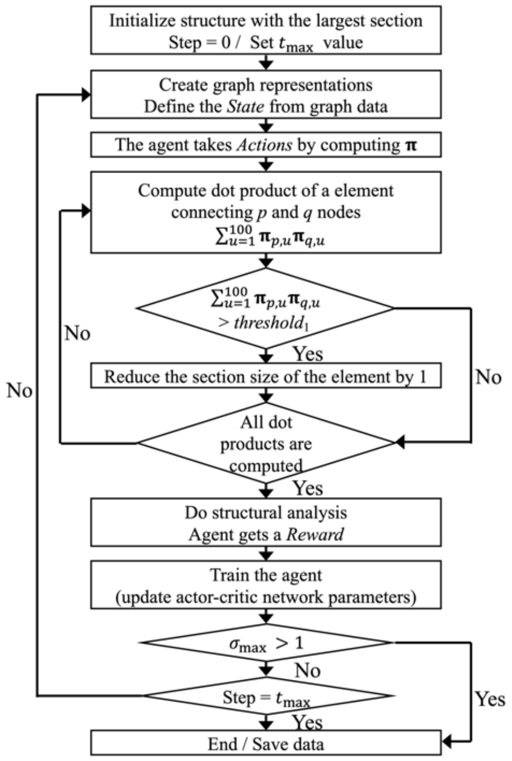
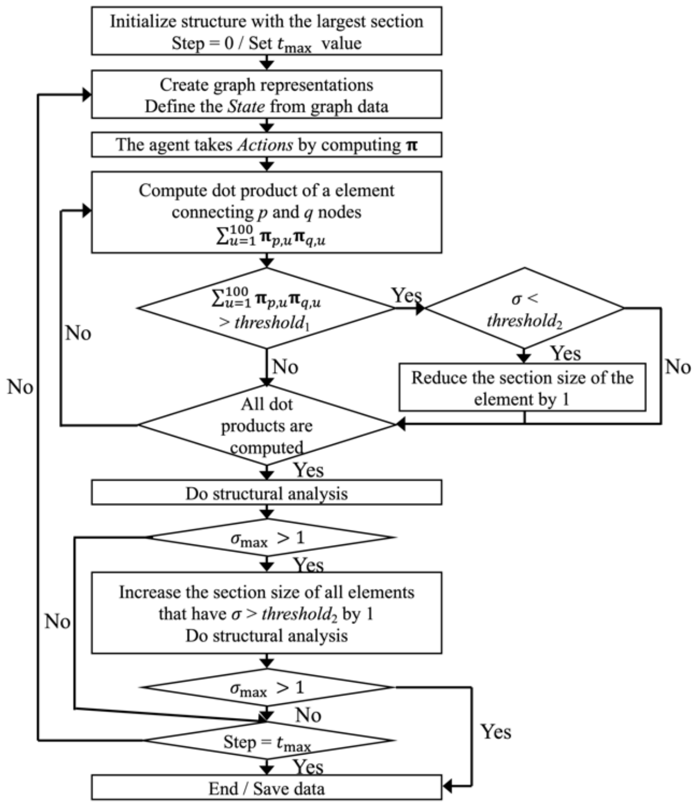
In each optimization step, the DDPG agent observes the graph representation of the structure, chooses elements to reduce their section sizes, and obtains the reward. The optimization ends when at least one of the structural elements violates the stress constraint or the number of optimization steps reaches a predetermined value. Similar to the optimization process in Section 3.2, the agent is trained to optimize the structure in the training phase to measure its ability to improve performance, utilizing small structures, and the agent performance is evaluated on large structures in the test phase.

In this problem, the lattice shells consist of 12-DoF frame elements having circular hollow sections chosen from Table 3.10. Young's modulus is 205 kN/mm^2. $\bar{\sigma}_1$, $\bar{\sigma}_2$, and $\bar{\sigma}_3$ are 235/1.5 N/mm^2, 0.66×235/1.5 N/mm^2, and 0.66×235/1.5 N/mm^2, respectively. The number of feature maps h in all GCN layers, explained in Section 2.5 in Chapter 2, is set to 300 and each layer in MLP has 300 cells.

Table 3.10: List of sections for sizing optimization of lattice shells

section No.	diameter (mm)	thickness (mm)	area (cm^2)	moment of inertia (cm^4)
0	21.7	2.0	1.238	0.607
1	34.0	2.3	2.291	2.890
2	42.7	3.2	3.971	7.800
3	48.6	3.2	4.564	11.80
4	60.5	4.0	7.100	28.50
5	76.3	4.0	9.085	59.50

This research trains the agent to optimize the structure to obtain high rewards in the training phase and apply it to other structures in the test phase. In the test phase, the agent cannot modify the element that has a ratio of internal stress to the yield stress larger than $threshold_2$. To stabilize the algorithm, if the structure violates the stress constraint, all elements with an internal stress ratio larger than $threshold_2$ will have the section number increased by one. Figure 3.17 illustrates flowcharts of proposed algorithms in the training (a) and test (b) phases.


Initialize structure with the largest section
Step = 0 / Set $t_{\max}$ value
Create graph representations
Define the *State* from graph data
The agent takes *Actions* by computing $\boldsymbol{\pi}$
Compute dot product of a element connecting p and q nodes
$\sum_{u=1}^{100} \boldsymbol{\pi}_{p,u}\boldsymbol{\pi}_{q,u}$
$\sum_{u=1}^{100} \boldsymbol{\pi}_{p,u}\boldsymbol{\pi}_{q,u} > threshold_1$
Yes
No
Reduce the section size of the element by 1
All dot products are computed
Yes
No
Do structural analysis
Agent gets a *Reward*
Train the agent
(update actor-critic network parameters)
$\sigma_{\max} > 1$
No
Yes
Step = $t_{\max}$
Yes
No
End / Save data

Initialize structure with the largest section
Step = 0 / Set $t_{\max}$ value
Create graph representations
Define the *State* from graph data
The agent takes *Actions* by computing $\boldsymbol{\pi}$
Compute dot product of a element connecting p and q nodes
$\sum_{u=1}^{100} \boldsymbol{\pi}_{p,u}\boldsymbol{\pi}_{q,u}$
$\sum_{u=1}^{100} \boldsymbol{\pi}_{p,u}\boldsymbol{\pi}_{q,u} > threshold_1$
Yes
$\sigma < threshold_2$
Yes
Reduce the section size of the element by 1
No
No
All dot products are computed
Yes
No
Do structural analysis
$\sigma_{\max} > 1$
Yes
No
Increase the section size of all elements that have $\sigma > threshold_2$ by 1
Do structural analysis
$\sigma_{\max} > 1$
No
Yes
Step = $t_{\max}$
Yes
End / Save data

(a) (b)

Figure 3.17: Flowcharts of the algorithm for sizing optimization of lattice shells; (a) Training phase, (b) Test phase

3.4.5.1 Training phase

In this phase, the agent is trained to optimize six different lattice shells illustrated in Figure 3.18(a)-(f). Table 3.11 shows details of these structures subjected to different nodal loads applied to all nodes. Note that these loads are selected to guarantee that the structures with the largest element sections do not violate the constraint while those with the smallest ones violate the constraint. Since it is difficult to measure the improvement of the agent using different structures, the ability of the agent to improve its policy is evaluated every 10 times (*episodes*) of training, using the same 6×6-grid lattice shells subjected to a downward nodal load of 1 kN applied to all nodes, as shown in Figure 3.19. $t_{\max}$ of the episode for training and episode for evaluation are set to 50 and 100, respectively. Adam optimizer using the mini-batch size of 32, as explained in Section 2.5.3 in Chapter 2, adjusts the trainable parameters in surrogate policy and value functions, as explained in Section 2.3.3 in Chapter 2, with learning rates of 10^{-6} and 10^{-5}, respectively. τ is set to 0.05 to update weights in surrogate functions to the online function every 100 steps, as explained in Section 2.3.3 in Chapter 2.

Table 3.11: Structures for training for sizing optimization of lattice shells

	S 1	S 2	S 3	S 4	S 5	S 6
grid size	4×4	4×4	4×4	4×4	4×4	4×4
grid span (m)	1	1	1	1	1	1
downward load (kN)	25	5	10	0.5	0.2	15

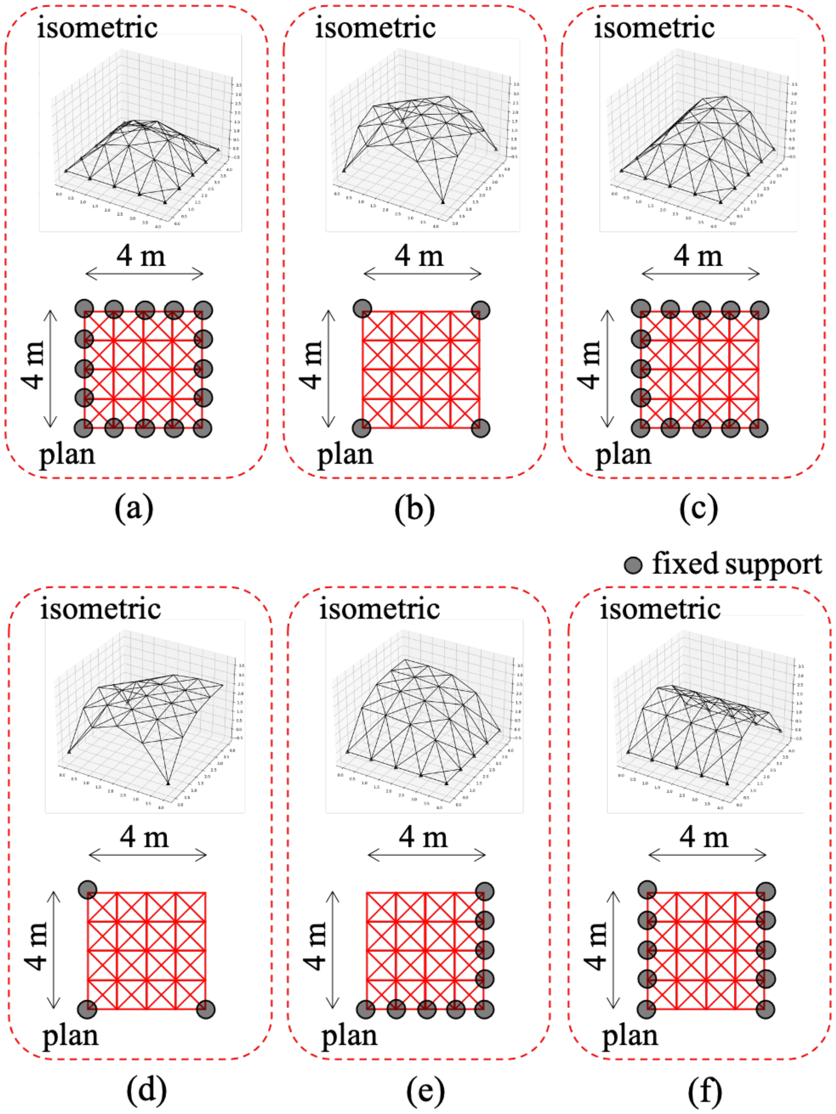


Figure 3.18: Structures used for training the agent for sizing optimization of lattice shells; (a) S 1, (b) S 2, (c) S 3, (d) S 4, (e) S 5, (f) S 6

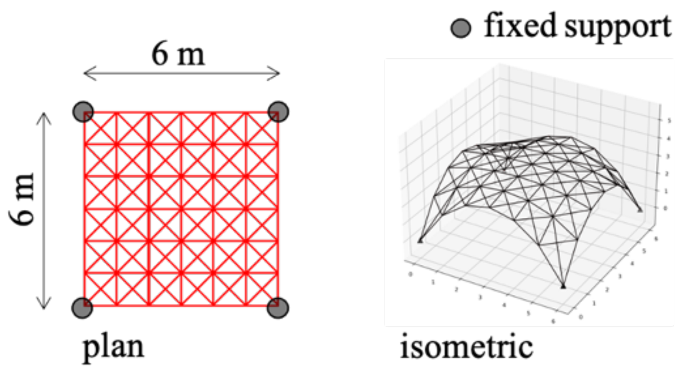


Figure 3.19: Structure used for evaluating the improvement of the trained agent for sizing optimization of lattice shells

Figure 3.20 shows variation of the cumulative reward (vertical axis) obtained during the training phase, which indicates the ability of the agent to improve its performance in each evaluation episode (horizontal axis). The thick red line, indicating the moving average of the reward every 20 episodes of evaluation, increases during the first 2000 episodes and becomes stable, which indicates that the agent can improve its policy function to obtain a high reward by minimizing the structural volume. The total number of structural analyses in this phase is 25849.

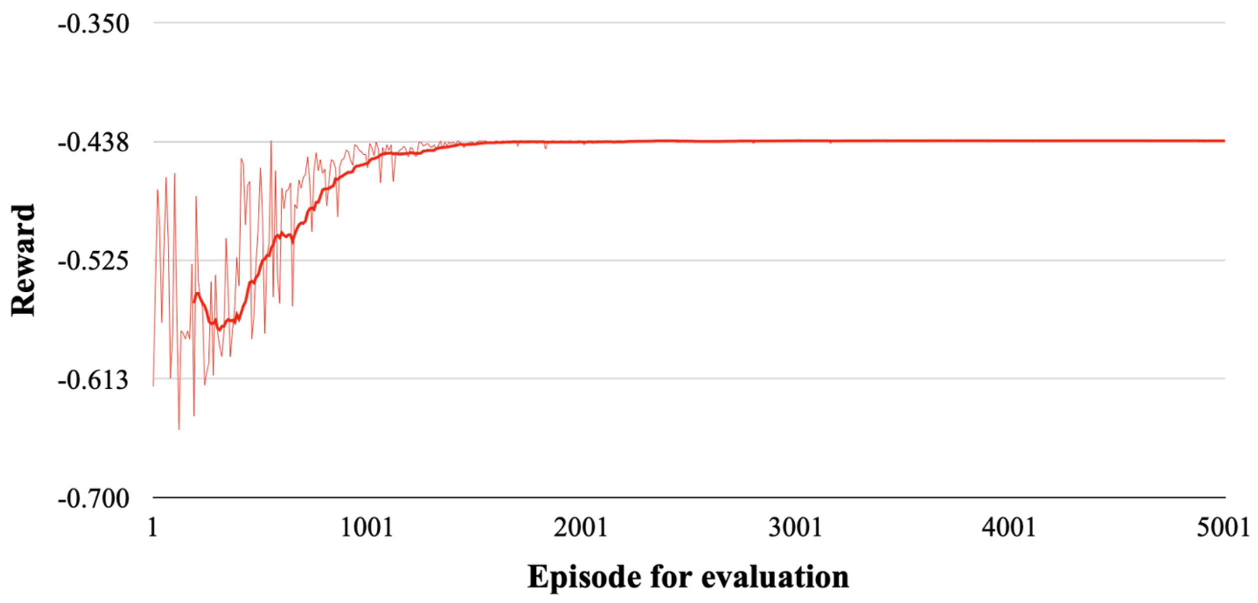


Figure 3.20: Variation of reward and its moving average in training phase for sizing optimization of lattice shells

3.4.5.2 Test phase

The trained agent with the weights saved at episode 5000 of the training phase is applied to larger lattice shell problems as shown in Table 3.12. In this phase, $threshold_2$ is set to 0.6 and the agent is applied to each structure ten times. Figures 3.21-3.23 illustrate the structures in the test phase: Structure A has fixed supports at the corners, Structure B has fixed supports at the corners with an irregular shape, and Structure C has fixed supports at the perimeter nodes.

Table 3.12: Structures for test for sizing optimization of lattice shells

	Structure A	Structure B	Structure C
grid size	10×10	20×20	20×20
grid span (m)	1	1	1
downward load (N)	200	5	300
number of elements	420	1640	1184
$t_{\max}$	100	200	200

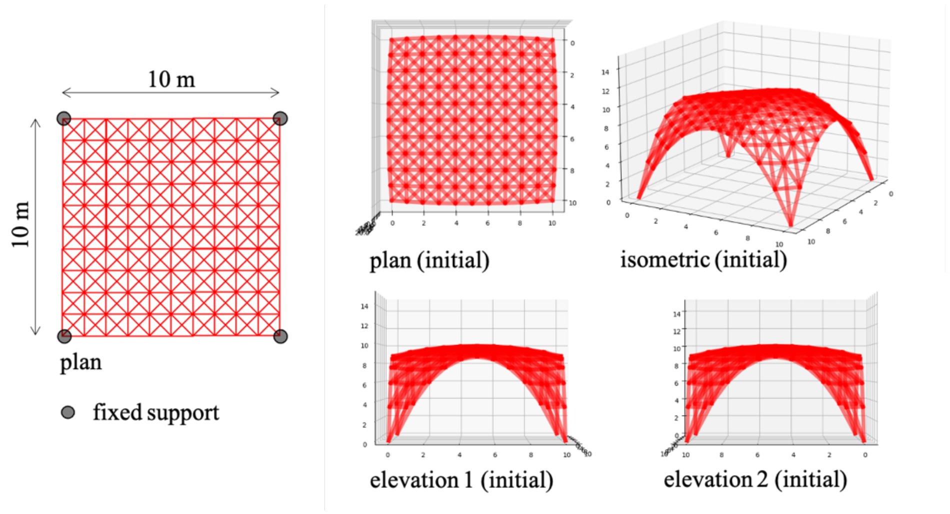


Figure 3.21: Structural models for test phase for sizing optimization of lattice shells – Structure A

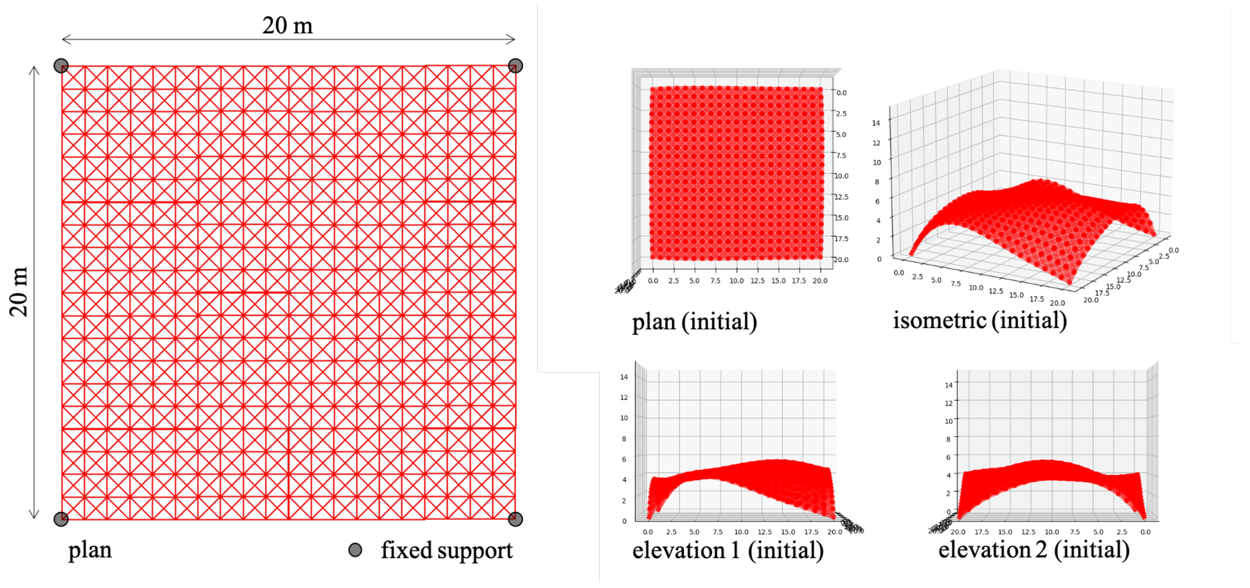


Figure 3.22: Structural models for test phase for sizing optimization of lattice shells – Structure B

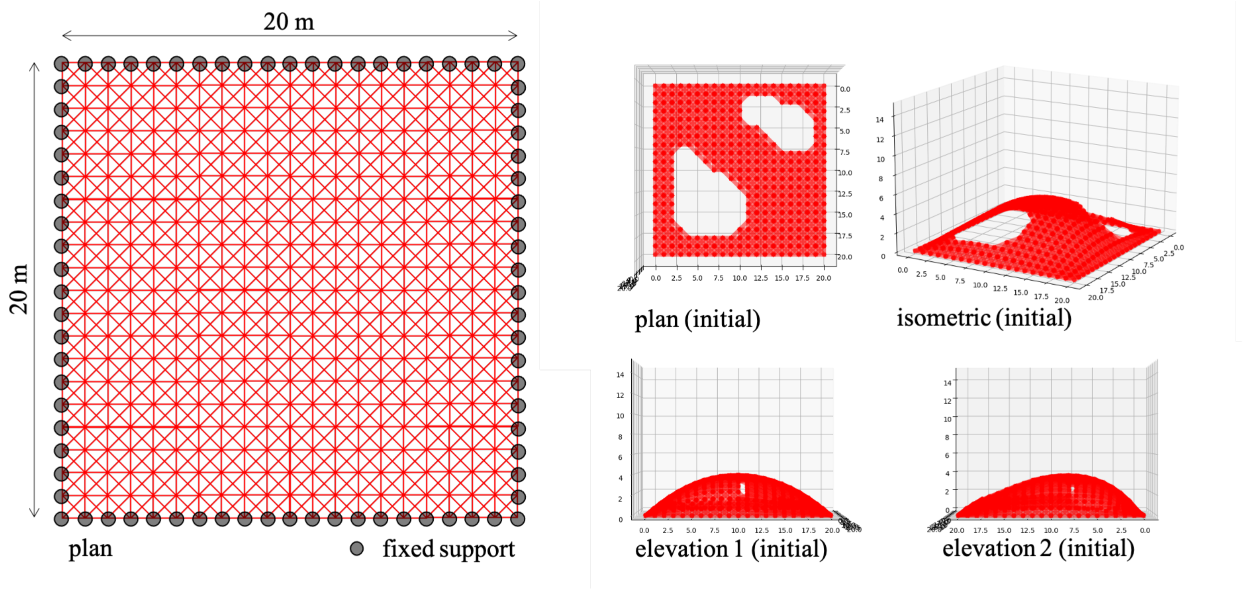


Figure 3.23: Structural models for test phase for sizing optimization of lattice shells – Structure C

Solutions obtained from the RL agent are compared to the results obtained by GA and SA, using Python libraries DEAP [128] and SciPy [129], respectively. In the GA case, each gene is represented as an integer in the range $\{0,1,2,3,4,5\}$ to indicate the sectional index of each element.

The numbers of the population and generation of GA are set as 100 and 10, respectively. In SA, the design parameters are the sectional indices of elements indicated as integer values in $\{0,1,2,3,4,5\}$. The maximum number of structural analyses of SA is set as 1000 times. Similar to RL, both SA and GA, performing ten trials using different random seeds, aim to minimize the cost function $\phi(\mathbf{c})$ computed as

$$\phi(\mathbf{c}) = \frac{V(\mathbf{c})}{V_0} + \sum_{i=1}^{n} w_\mathrm{c}(\max[0, |\sigma_i| - 1])^2 \tag{3.20}$$

where $V(\mathbf{c})$ and V_0 are the total structural volume of the design $\mathbf{c}$ and the initial volume, respectively. n is the number of elements in the structure. w_c is the weight for the constraint set as 50.

The performance of RL, GA, and SA are indicated using the volume reduction computed as

$$\text{volume reduction} = (V_0 - V_*)/V_0 \tag{3.21}$$

where V_* is the final volume of the lattice shells.

Table 3.13 shows the minimum (min.), the mean, the maximum (max.), and the standard deviation (std.) of the volume reductions obtained by the RL, GA, and SA, together with the number of structural analyses. The RL agent can obtain superior solutions (i.e., better volume reductions) than those of GA and SA in every structure. Note that the standard deviations of results of RL are also lower compared to those of GA and SA indicating stabile performance of the proposed RL method.

Table 3.13: Test result (volume reduction (%)) for sizing optimization of lattice shells

	structure	min.	mean	max.	std.	No. of analyses
RL	A	82.06	82.36	82.53	0.15	2000
	B	86.00	86.07	86.12	0.03	4000
	C	84.57	84.69	84.75	0.05	4000
GA	A	0.00	22.86	39.35	19.69	10000
	B	82.25	82.74	83.20	0.37	10000
	C	73.29	73.97	74.67	0.49	10000
SA	A	0.00	34.46	70.34	36.33	10000
	B	68.08	68.34	68.60	0.19	10000
	C	0.00	60.97	68.08	21.42	10000

Figures 3.24-3.26 illustrate the best results of each structure obtained by RL in the test phase. In these figures, the graph on the left indicates the structural volume (vertical axis) in each optimization step (horizontal axis). The isometric images of the initial and final structures are also illustrated over the graph. In this image, the thickness of the line is proportional to the value of sectional indices. Four images on the right illustrate the plan and elevations of the final structure together with the initial stress ratio.

From these figures, obtained solutions by RL generally have elements with large sections in the region where the initial stress ratio is large. In Figure 3.24, where the structure is symmetry, the agent obtained a symmetric configuration of the element sizes despite the symmetrical characteristic of the structure is not explicitly assigned for the agent. The RL agent also obtained reasonable element sizes of large structures in Figures 3.25 and 3.26 even though it is trained using much smaller lattice shells in the training phase.

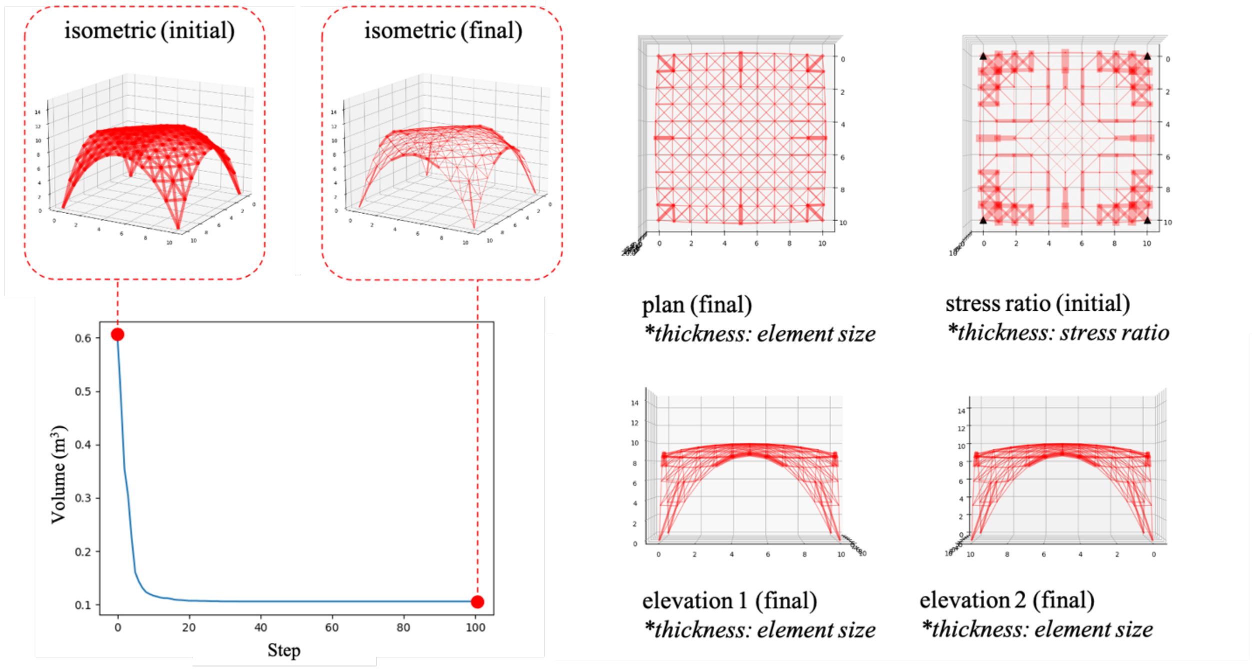


Figure 3.24: Optimized test structure for sizing optimization of lattice shells - Structure A

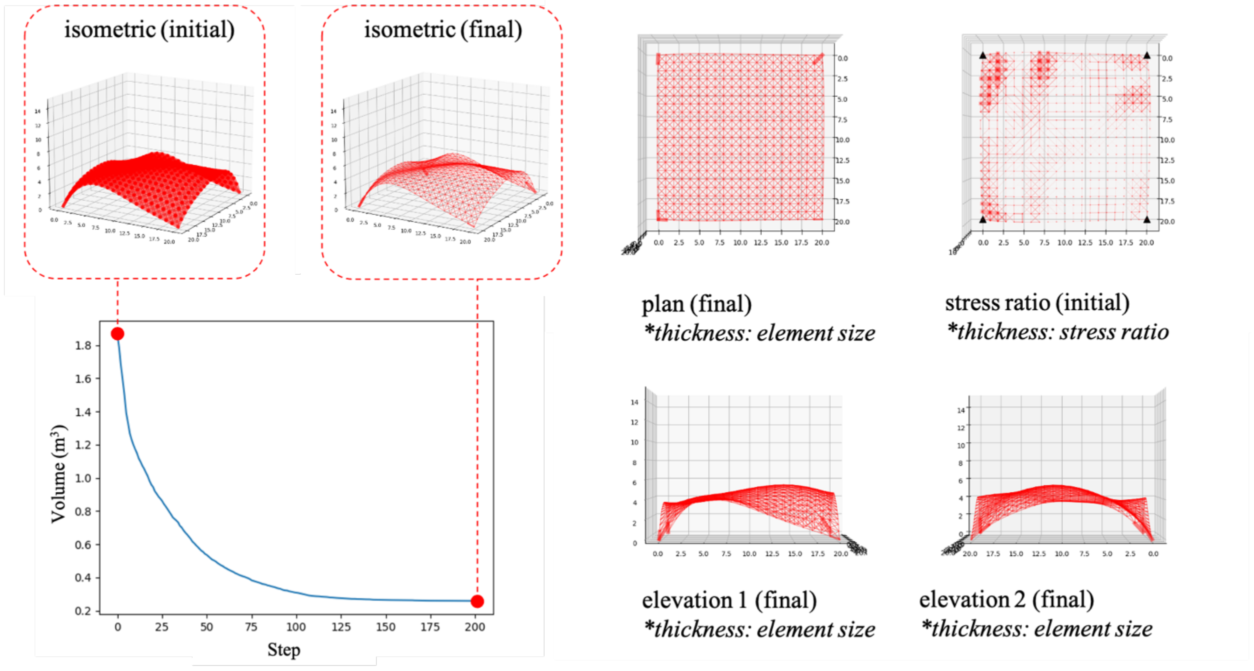


Figure 3.25: Optimized test structure for sizing optimization of lattice shells - Structure B

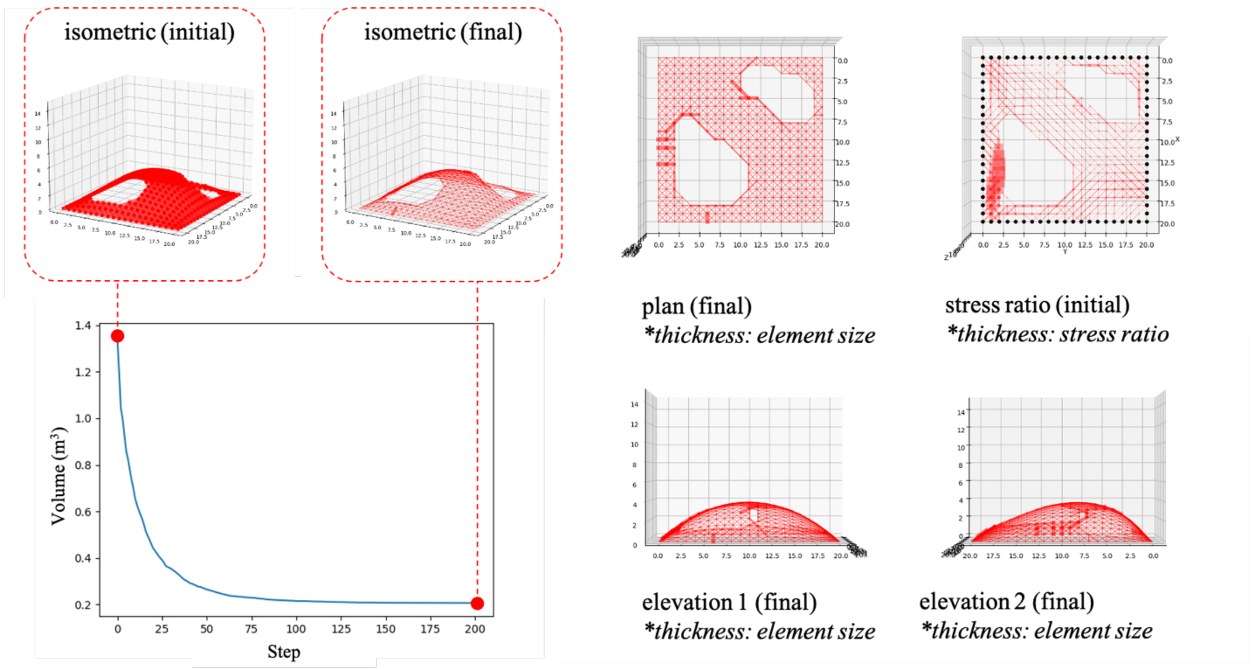


Figure 3.26: Optimized test structure for sizing optimization of lattice shells - Structure C

3.5 Conclusion

In this chapter, we introduced a GCN-DDPG agent and explored its application in the context of topology and sizing optimization for lattice shell structures. The primary objective of the proposed method is to train an RL agent to iteratively enhance the structural performance by modifying the structural configuration, which is represented as a graph. In this representation, structural nodes correspond to nodes within the graph, while structural elements are mapped to graph edges. The salient findings of this chapter are summarized as follows:

(1) The GCN-DDPG agent exhibits a capability to learn and improve the topology of lattice shell structures. It achieves this by employing a reward function derived from the reduction in the strain energy, showcasing its ability for structural optimization.

(2) The trained GCN-DDPG agent demonstrates its effectiveness in optimizing the topology of various lattice shell structures with the objective of minimizing the total strain energy. Furthermore, its performance is shown to be competitive when compared to the GA, a widely used population-based optimization technique.

(3) One of the distinctive advantages of the RL agent lies in its capacity to be trained using smaller lattice shell structures, which entail lower computational costs. This trained agent can subsequently be applied to larger structural configurations, illustrating its scalability.

(4) The GCN-DDPG agent can also be adapted to optimize the sizes of structural elements within lattice shell structures. This is achieved through the formulation of a reward function that captures variations in structural volumes.

(5) In addressing challenges in sizing optimization for different lattice shell structures, the trained RL agent exhibits superior performance in comparison to other optimization algorithms including GA and SA.

(6) Notably, the proposed GCN-DDPG agent showcases a more stable and consistent performance when compared with stochastic- and population-based optimization methods. This enhanced stability translates into a reduced number of trials required to converge to a good approximate optimal solution.

(7) Finally, the methodologies presented in this chapter hold promise for practical applications in the architectural and engineering domains, especially in the initial design phases where computational time for each design decision is a critical factor. Architects and engineers can leverage these techniques to efficiently explore and refine lattice shell structures, contributing to more efficient and resource-conscious design processes. It should be noted that loads in the optimizations in this chapter are only vertical loads, which directly affect the optimal topology of the lattice shells and sizes of the lattice members.

In conclusion, this chapter proposed RL-based optimization methods for topology and sizing optimization problems of lattice shells utilizing the strain energy and the structural volume as objective functions in the first and second problems, respectively. The results show that the proposed methods are applicable to the problem of topology and sizing optimizations for structural engineering. It should be noted that the problems in this chapter consider only vertical loads to simplify the problem formulations. Future research should aim to verify the proposed method in other loading and collapse scenarios of lattice shells, such as horizontal load and shell buckling.

Chapter 4
Simultaneous topology and sizing optimization of braced building frames using RL

4.1 Introduction

In regions prone to seismic activity, such as Japan, ensuring the seismic resistance of building frames is important for structural integrity and public safety. Achieving seismic resistance involves a careful selection of structural member sizes and bracing configurations that can effectively withstand external loads while minimizing structural costs. This complex optimization task, encompassing both topology (arrangement of structural elements) and sizing (dimension of structural members) considerations, presents a formidable task in structural engineering. It is fundamentally a discrete combinatorial optimization problem, where design variables include the size of structural elements and the configuration of bracing systems. The intricacies and non-linearity of the problem render it unsuitable for conventional gradient-based optimization methods. To address these challenges, conventional practice often resorts to population-based algorithms such as GA, which, while effective, come with significant computational demands.

The primary focus of this chapter is on advancing the methodology for topology and sizing optimization of building frames subjected to seismic loads by harnessing RL. This novel approach seeks to reduce computational costs represented by structural material volume and enhance optimization efficiency while delivering structures that meet seismic resistance requirements. Section 4.1 sets the stage by introducing the seismic analysis and the optimization problem intrinsic to building frames. The chapter proceeds to unveil a novel methodology in Section 4.2, wherein a DDPG agent, complemented by a unique graph representation of the structure, undertakes the task of optimizing building frame configurations. In this graph representation, structural elements are represented as graph nodes, fostering a more intuitive and effective way to encode the structural information.

The DDPG agent, constructed using GCNs, is trained to iteratively refine an initial structural configuration until it attains seismic-resistant performance while minimizing structural volume. This is achieved by formulating the training process in a way that induces the agent to seek structural improvements while taking into account the associated structural volume. Consequently, the training objectives encompass the dual aims of minimizing structural volume and ensuring stiffness to satisfy seismic constraints. Numerical results show that the RL-based optimization offers the promise of efficiently solving simultaneous topology and sizing optimization of braced building frames subjected to seismic loads.

4.2 Formulation of optimization problem

4.2.1 Estimated seismic load using A*ᵢ*-bunpu method

In the Japanese A_i distribution (*ai-bunpu*) method, the horizontal seismic force on each floor is estimated by distributing all shear forces applied to the building. The horizontal load P_i on the ith floor of the building frame is computed and utilized as the design load of the building frame. Let Q_i, C_i, W_i, R, and $\mathcal{A}_i$ be the shear force applying on the ith story between the $(i$-1)th and ith floors, shear coefficient of the ith story, the weight of the ith floor, a value representing soil category and vibration characteristics of the building, and the shear distribution coefficient of the ith floor, respectively. The computation of P_i is as follows:

$$P_i = \begin{cases} Q_i & : \text{if } i = N_\mathrm{f} \\ Q_{i+1-}Q_i & : \text{if } i = 1, \dots, N_\mathrm{f} - 1 \end{cases} \tag{4.1}$$

$$Q_i = C_i \sum_{j=i}^{N_\mathrm{f}} W_j \tag{4.2}$$

$$C_i = Z_0 R C_0 \mathcal{A}_i \tag{4.3}$$

$$R = \begin{cases} 1 & : \text{if } T < T_\mathrm{c} \\ 1 - 0.2\left(\frac{T}{T_\mathrm{c}} - 1\right)^2 & : \text{if } T_c \leq T < 2T_c \\ 1.6\frac{T}{T_c} & : \text{if } 2T_\mathrm{c} \leq T \end{cases} \tag{4.4}$$

$$T = 0.03h \tag{4.5}$$

$$\mathcal{A}_i = 1 + \left(\frac{1}{\sqrt{\alpha_i}} - \alpha_i\right)\frac{2T}{1+3T} \tag{4.6}$$

$$\alpha_i = \frac{\sum_{j=i}^{N_\mathrm{f}} W_j}{\sum_{k=1}^{N_\mathrm{f}} W_k} \tag{4.7}$$

$$W_i = w_i \alpha_i \tag{4.8}$$

where C_0, T, w_i, and α_i are the base shear coefficient, the fundamental natural period of the building, the unit load of the ith floor, and the area of the ith floor, respectively. h, T_c, Z_0, and N_f are the total height of the building, the specific period assigned according to the soil type under the building, the regional seismic coefficient, and the number of floors excluding the base, respectively.

These estimated horizontal forces are denoted as *short-term loading* and are applied in both left and right directions to evaluate the largest response of the building frame. The structural responses, consisting of internal forces and displacements, are computed using linear elastic analysis. Note that the responses obtained under dead and live loads denoted as *long-term loading* are also computed. The building frames consist of beams, columns, and bracing elements. The beams and columns are modeled using 2-dimensional beam elements with 6-DoFs, whereas the bracing elements are 2-dimensional truss elements with 4-DoFs. The stiffness matrices of the beam or column element and the brace element are denoted, in the local coordinate system, as $\mathbf{k}_\mathrm{f} \in \mathbb{R}^{6\times6}$ and $\mathbf{k}_\mathrm{e} \in \mathbb{R}^{4\times4}$, respectively. These matrices are transformed into the global coordinate system and assembled into the global stiffness matrix $\mathbf{K} \in \mathbb{R}^{n_\mathrm{D}\times n_\mathrm{D}}$, where n_D is the number of DoFs of the building frame after assigning the boundary conditions.

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

$$\mathbf{Kd} = \mathbf{p} \tag{4.9}$$

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

Figure 4.1 illustrates the internal forces $\mathbf{f}_i = \left(f_p^1, f_p^2, f_p^\theta, f_q^1, f_q^2, f_q^\theta\right)$ of a beam or column element i obtained from p and q end displacements of the element i in the local coordinate system and its corresponding stiffness matrix. Internal forces $\mathbf{f}_i = \left(f_p^1, f_p^2, f_q^1, f_q^2\right)$ of a brace element can be obtained similarly. Let A_i and Z_i denote the cross-sectional area and the section modulus of element i, respectively. Internal stress σ_i of a structural element i can be computed as follows:

$$\sigma_i = \frac{\left|f_p^1\right|}{A_i} + \frac{\max\left(\left|f_p^\theta\right|, \left|f_q^\theta\right|\right)}{Z_i} \tag{4.10}$$

where the second term does not exist for a truss element. Note that this research assumes the floor to be rigid. The in-plane stiffness of the slab is incorporated by multiplying the axial stiffness of the beam element by 10.

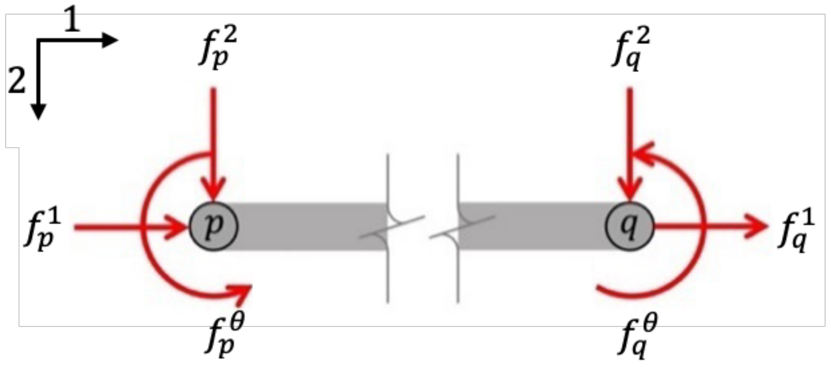


Figure 4.1: Internal forces in local coordinates for simultaneous topology and sizing optimization of building frames

4.2.2 Simultaneous topology and sizing optimization of the braced building frame

In the topology and sizing optimization of the building frame, the building frame without brace is initialized with the smallest section index in the list. During the optimization, braces of various types, including diagonal braces, K-type, and V-type as shown in Figure 4.2, are placed into the building frame together with adjustment of sizes of beams, columns, and braces until the building frame can satisfy all constraints on the structural responses.

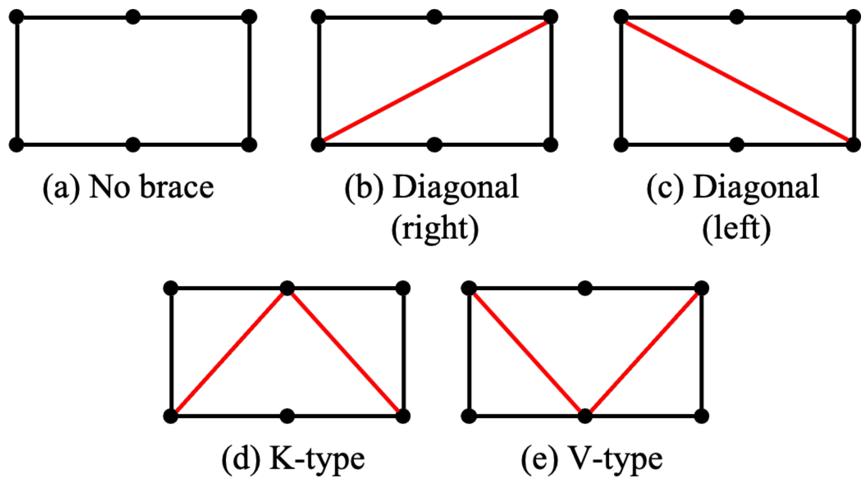


Figure 4.2: Five brace types including no brace for simultaneous topology and sizing optimization of building frames

The optimal section sizes of beams, columns, and braces as well as brace placements of steel frames are to be obtained for minimizing the objective function, which represents the material and construction costs of steel building frames to be optimized to obtain optimal section sizes of beams, columns, and braces as well as brace placements. Let $c_i \in \{0,1,\dots,n_\mathrm{c}\}$, $b_i \in \{0,1,\dots,n_\mathrm{b}\}$, and $br_i \in \{0,1,\dots,n_\mathrm{br}\}$ be the indices of the cross-sectional properties of columns, beams, and braces, respectively. A vector representing the element properties of the building frame with N_s spans and N_f stories, which has c_f columns and b_f beams is defined as

$$\mathbf{A} = \left(c_1,\dots,c_{c_\mathrm{f}}, b_1,\dots,b_{b_\mathrm{f}}, br_1,\dots,br_{N_s N_f}\right) \tag{4.11}$$

Also, a vector, representing the type of brace in each domain of a building frame is defined as

$$\mathbf{B} = \left(r_1,\dots,r_{N_s N_f}\right), \quad r_i \in \{0,1,2,3,4\} \tag{4.12}$$

where 0, 1, …, 4 correspond to the brace types (a), (b), …, (e) in Figure 4.2, respectively.

From Eqs. (4.11) and (4.12), a design variable vector for a building frame can be represented as a combination of **A** and **B**, defined as

$$\mathbf{X} = \{\mathbf{A}, \mathbf{B}\} \tag{4.13}$$

The cost $V(\mathbf{X})$ is represented by the total structural volume, which is the objective function to be minimized. Note that the cost of braces is multiplied by the cost coefficient B to avoid placing too many braces. Let $\sigma^{\mathrm{L}}_{\max}$, $\sigma^{\mathrm{S}}_{\max}$, and $\theta_{\max}$ denote the maximum absolute values of stresses at the element ends among all members due to *long-term loading*, the largest maximum absolute value of stresses at the element ends among all members due to *short-term loading* in both directions, and the maximum absolute value of inter-story drift angles among all stories, respectively. The optimization problem of brace placements (i.e., topology) and member sizes (i.e., sizing) of the steel frame is formulated as

$$\begin{aligned} \text{minimize} \quad & V(\mathbf{X}) \\ \text{subject to} \quad & \sigma^{\mathrm{L}}_{\max} \le \hat{\sigma}^{\mathrm{L}} \\ & \sigma^{\mathrm{S}}_{\max} \le \hat{\sigma}^{\mathrm{S}} \\ & \theta_{\max} \le \hat{\theta} \\ & \delta_{\max} \le 1 \\ & \lambda_{\max} \le 1 \end{aligned} \tag{4.14}$$

where $\hat{\sigma}^{\mathrm{L}}$, $\hat{\sigma}^{\mathrm{S}}$, and $\hat{\theta}$ denote upper bounds of $\sigma^{\mathrm{L}}_{\max}$, $\sigma^{\mathrm{S}}_{\max}$, and $\theta_{\max}$, respectively. Note that constraints on base beams are also measured because they are necessary for adding V-type braces. $\delta_{\max}$ is the maximum value of deflection ratio δ^{b}_i at the centers of all beams defined as follows:

$$\delta^{\mathrm{b}}_i = \max\left(\frac{300\left|d^{\mathrm{b}}_i\right|}{L^{\mathrm{b}}_i}, 1\right) \tag{4.15}$$

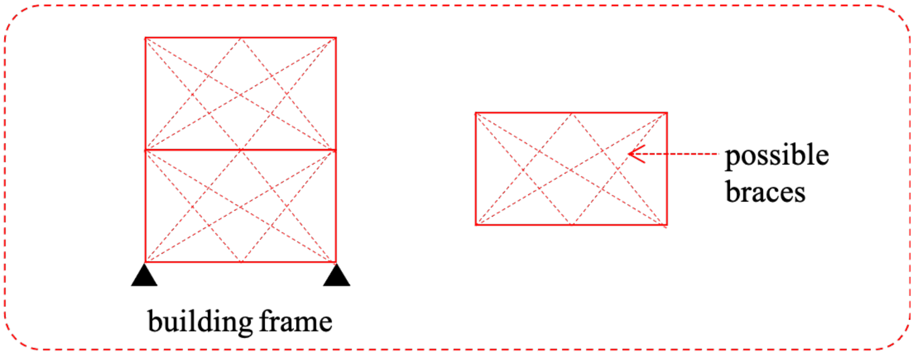
where d^{b}_i and L^{b}_i are the deflection at the center of beam i in the local y coordinate, and the length of beam i, respectively. $\lambda_{\max}$ is the maximum value of buckling stress ratios λ_i of all braces, defined as

$$\lambda_i = \begin{cases} \dfrac{\sigma^{\mathrm{S}}_i}{\pi^2 E_i I_i / (A_i {L_i}^2)} & : \text{if } f^x_p > 0 \\ 0 & : \text{if } f^x_p \le 0 \end{cases} \tag{4.16}$$

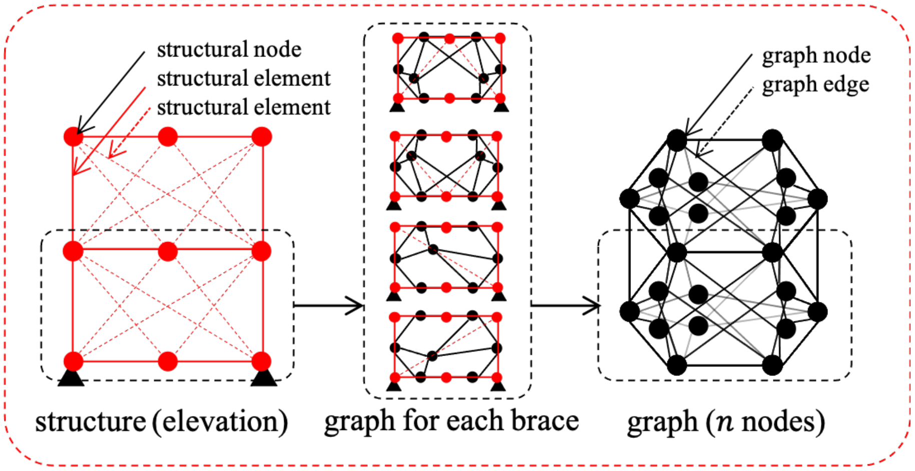
where σ_i^S is the maximum absolute value of axial stress in the brace i due to *short-term loading* in both directions.

4.3 Simultaneous topology and sizing optimization of braced building frame using DDPG and GCN

This section introduces the RL and graph representation method for solving the problem in Section 4.1.2. The optimization problem is formulated into an MDP and reward is computed using the cost of improving the structure. The DDPG agent observes the state from the graph representation of the structure. Figure 4.3 shows the graph representation in this problem where graph nodes and edges represent structural elements and connectivity of each element, respectively. The building frames have main elements (i.e., columns and beams) represented as solid red lines and possible bracing represented as dotted red lines (Figure 4.3(a)). Note that in each structural gird, there can be only one type of bracing. The main elements and all possible bracing, illustrated in red dots, are represented as graph nodes, illustrated in black dots (Figure 4.3(b)). Information of the structural elements is stored in the graph node, represented using a node feature matrix while the connectivity of the structural elements is represented through an adjacency matrix (Figure 4.3(c)). Note that there is no need for weighted adjacency matrices to represent internal forces because these forces are already incorporated into the node feature matrix. The GCN-DDPG agent utilizes these matrices as state data to compute the actions at each optimization step.



(a)



(b)

$$\mathbf{N} = \overbrace{\begin{bmatrix} n_{1,1} & \cdots & n_{1,f} \\ \vdots & \ddots & \vdots \\ n_{n,1} & \cdots & n_{n,f} \end{bmatrix}}^{f \text{ features}} \Big\} n \text{ nodes} \qquad \textbf{Node feature matrix}$$

$$\mathbf{M} = \overbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & \cdots & \vdots \\ 0 & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}}^{n \text{ nodes}} \Big\} n \text{ nodes} \qquad \textbf{Adjacency matrix}$$

(c)

Figure 4.3: Graph representation of the braced building frame for simultaneous topology and sizing optimization of braced building frames

Figure 4.4 illustrates the MDP of the topology and sizing optimization of the braced building frame. At a step t, the agent observes the state S_t which is the graph representation of the current structural configuration shown in Figure 4.4(a). The agent does the action A_t which is either increasing the sectional area of an existing structural element or adding bracing in an unoccupied structural grid as shown in Figure 4.4(b). The structure is modified and structural analysis for three load cases is computed to obtain the objective function value. The agent obtains a reward R_{t+1} based on the cost of improving the building frame Figure 4.4(c) and the new step of MDP begins. This MDP loop keeps on until all constraints are satisfied.

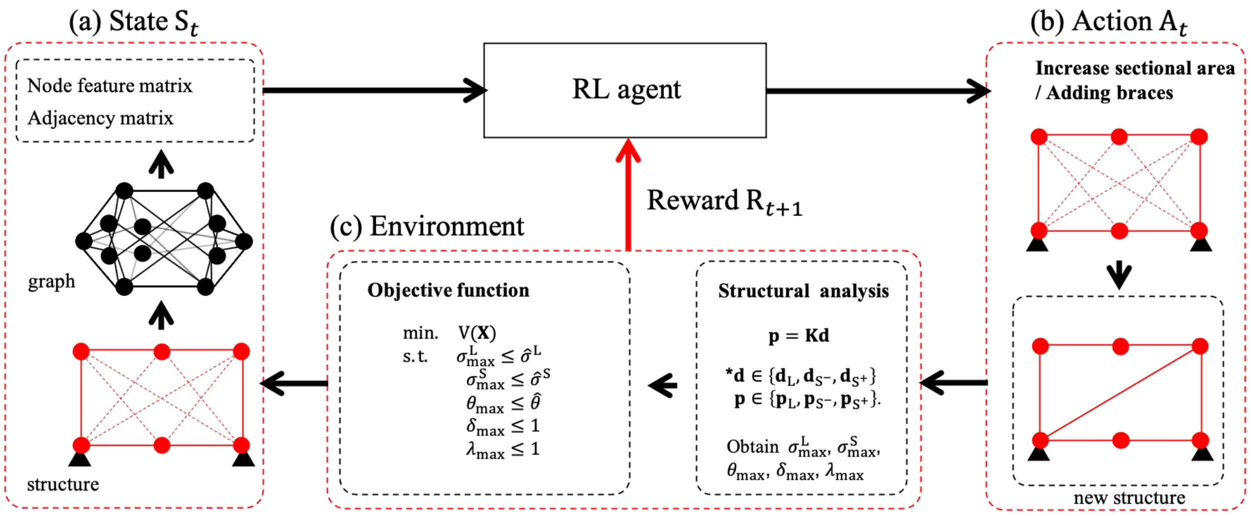

(a) State S_t
Node feature matrix
Adjacency matrix
graph
structure
RL agent
(b) Action A_t
Increase sectional area / Adding braces
new structure
Reward R_{t+1}
(c) Environment
Objective function
min. $V(\mathbf{X})$
s.t. $\sigma^{L}_{\max} \le \hat{\sigma}^{L}$
$\sigma^{S}_{\max} \le \hat{\sigma}^{S}$
$\theta_{\max} \le \hat{\theta}$
$\delta_{\max} \le 1$
$\lambda_{\max} \le 1$
Structural analysis
$\mathbf{p} = \mathbf{K}\mathbf{d}$
$^{*}\mathbf{d} \in \{\mathbf{d}_L, \mathbf{d}_{S^-}, \mathbf{d}_{S^+}\}$
$\mathbf{p} \in \{\mathbf{p}_L, \mathbf{p}_{S^-}, \mathbf{p}_{S^+}\}$.
Obtain $\sigma^{L}_{\max}, \sigma^{S}_{\max}, \theta_{\max}, \delta_{\max}, \lambda_{\max}$

Figure 4.4: MDP of the topology optimization of braced building frame for simultaneous topology and sizing optimization of braced building frames

4.3.1 State

In this example, the state is a graph representation of the structure consisting of node feature matrix **N** and adjacency matrix **M**. Tables 4.1 and 4.2 show entries in the node feature matrix and the adjacency matrix, respectively. All values in these Tables are normalized using the min-max normalization method.

Table 4.1: Entries in node feature matrix (**N**) for simultaneous topology and sizing optimization of braced building frames

$n_{i,1}$	1 if i is in the structure; else 0
$n_{i,2}$	0 if sectional area of element i is equal to the largest area for i; else 1
$n_{i,3}$	1 if element i violates stress constraint under $\mathrm{p}_{\mathrm{S}-}$; else 0
$n_{i,4}$	1 if element i violates stress constraint under $\mathrm{p}_{\mathrm{S}+}$; else 0
$n_{i,5}$	1 if element i violates stress constraint under p_{L}; else 0
$n_{i,6}$	1 if the p end of element i is a fixed support; else 0
$n_{i,7}$	0 if the p end of element i is a fixed support; else 1
$n_{i,8}$	magnitude of load acting on the p end of element i in x direction
$n_{i,9}$	magnitude of load acting on the p end of element i in y direction
$n_{i,10}$	x-directional displacement of the p end of element i under $\mathrm{p}_{\mathrm{S}-}$
$n_{i,11}$	x-directional displacement of the p end of element i under $\mathrm{p}_{\mathrm{S}+}$
$n_{i,12}$	y-directional displacement of the p end of element i under $\mathrm{p}_{\mathrm{S}-}$
$n_{i,13}$	y-directional displacement of the p end of element i under $\mathrm{p}_{\mathrm{S}+}$
$n_{i,14}$	p end's x coordinate of element i
$n_{i,15}$	p end's y coordinate of element i
$n_{i,16}$	1 if the q end of element i is a fixed support; else 0
$n_{i,17}$	0 if the q end of element i is a fixed support; else 1
$n_{i,18}$	magnitude of load acting on the q end of element i in x direction
$n_{i,19}$	magnitude of load acting on the q end of element i in y direction
$n_{i,20}$	x-directional displacement of the q end of element i under $\mathrm{p}_{\mathrm{S}-}$
$n_{i,21}$	x-directional displacement of the q end of element i under $\mathrm{p}_{\mathrm{S}+}$
$n_{i,22}$	y-directional displacement of the q end of element i under $\mathrm{p}_{\mathrm{S}-}$
$n_{i,23}$	y-directional displacement of the q end of element i under $\mathrm{p}_{\mathrm{S}+}$
$n_{i,24}$	q end's in x coordinate of element i
$n_{i,25}$	q end's in y coordinate of element i
$n_{i,26}$	length of element i
$n_{i,27}$	cross-sectional area of element i
$n_{i,28}$	second moment area of element i
$n_{i,29}$	internal stress in element i under $\mathrm{p}_{\mathrm{S}-}$
$n_{i,30}$	internal stress in element i under $\mathrm{p}_{\mathrm{S}+}$
$n_{i,31}$	internal stress in element i under p_{L}

Table 4.2: Entries in adjacency matrix (**M**) for simultaneous topology and sizing optimization of braced building frames

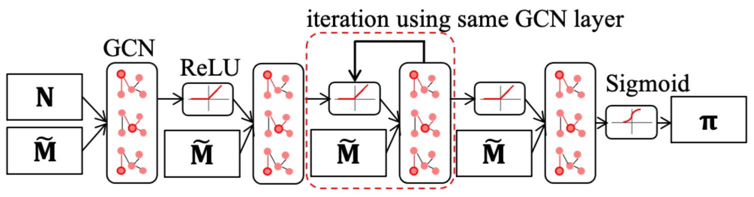
$m_{i,\mathrm{j}}$	1 if there is element i connecting to element j; else 0

4.3.2 Agent

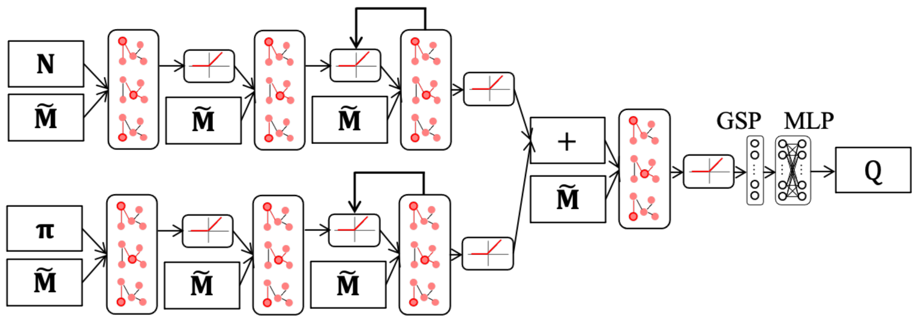
This research utilizes GCN for making the policy and value functions (i.e., actor and critic networks) of a DDPG agent. The policy function $\boldsymbol{\pi}$ takes state data ($\mathbf{N}$, $\widetilde{\mathbf{M}}$) as input to compute the output $\boldsymbol{\pi} \in \mathbb{R}^{n\times 1}$. This output is interpreted to determine the element to be increased in size or added (for a brace). Value function Q computes Q-value utilizing state data and $\boldsymbol{\pi}$ as inputs. Table 4.3 shows the inputs, computations, and outputs of the policy function (left column) and the value function (right column). Figure 4.5 shows the neural network architectures of actor and critic networks in this chapter.

Table 4.3: Policy and value functions of GCN-DDPG for simultaneous topology and sizing optimization of braced building frames

policy function $\boldsymbol{\pi}$	value function Q
input: $\mathbf{N}$, $\widetilde{\mathbf{M}}$	input: $\mathbf{N}$, $\widetilde{\mathbf{M}}$, $\boldsymbol{\pi}$
computation: step 1: $\mathbf{N}_1 = \mu(\mathbf{N}, \widetilde{\mathbf{M}})$ step 2: $\mathbf{N}_2 = \mathrm{iter}^2[\mu(\mathbf{N}_1, \widetilde{\mathbf{M}})]$ step 3: $\boldsymbol{\pi} = \sigma(\mathbf{N}_2, \widetilde{\mathbf{M}})$	computation: step 1: $\mathbf{N}_{1.1} = \mu(\mathbf{N}, \widetilde{\mathbf{M}})$ $\mathbf{N}_{1.2} = \mathrm{iter}^2[\mu(\mathbf{N}_{1.1}, \widetilde{\mathbf{M}})]$ step 2: $\mathbf{N}_{2.1} = \mu(\boldsymbol{\pi}, \widetilde{\mathbf{M}})$ $\mathbf{N}_{2.2} = \mathrm{iter}^2[\mu(\mathbf{N}_{2.1}, \widetilde{\mathbf{M}})]$ step 3: $\mathbf{N}_2 = \mathbf{N}_{1.2} + \mathbf{N}_{2.2}$ step 4: $\mathbf{N}_3 = \mu(\mathbf{N}_2, \widetilde{\mathbf{M}})$ step 5: $\mathbf{n}_4 = \mathrm{GSP}(\mathbf{N}_3)$ step 6: $\mathrm{Q} = \mathrm{f}_{\mathrm{NN}}(\mathbf{n}_4)$
output: $\boldsymbol{\pi} \in \mathbb{R}^{n\times 1}$	output: $\mathrm{Q} \in \mathbb{R}^{1\times 1}$



(a) Actor network



(b) Critic network

Figure 4.5: Actor and critic networks for simultaneous topology and sizing optimization of braced building frames

4.3.3 Action

In this research, the agent does one of the actions including (1) increasing the index (i.e., sectional area) of the existing element and (2) adding a brace into an unoccupied structural grid. To unify the representation, indices of all braces are initialized as 0, representing no braces in the structure. Since the output of the agent $\boldsymbol{\pi} \in \mathbb{R}^{n\times 1}$ represents a value for each structural element, the element to be modified (i.e., enlarged or added) can be determined from the one with the largest value among the entries of $\boldsymbol{\pi}$. To enhance the optimization process by restricting the agent to modify irrelevant elements including the elements with the largest index and the brace elements that cannot be added because the gird is already occupied, the vector that represents feasible action $\mathbf{g} \in \mathbb{R}^{n\times 1}$ is introduced. In this vector, the entry g_i is 1 if it is possible to increase the index (i.e., increasing a section of an existing element or adding a brace) of the element i and is 0 otherwise, making the probability to select an infeasible action becomes 0. The element i to be modified is determined from the index i that has the largest value of the product of an element of the policy function $\boldsymbol{\pi}_i$ and $\mathbf{g}_i$ as follows:

$$i = \underset{j}{\operatorname{argmax}}\, \mathbf{g}_j \boldsymbol{\pi}_j \tag{4.17}$$

Note that no brace exists in the initial frame and the number of grids with braces on each story can be only half of the total number of domains on that story. Once the braces are added, their domain or type cannot be modified. Note that if one of the braces that make up K-Type or V-Type is selected, the other member will also be changed to the same index.

From Eq. (4.17), the section index of element *i* is increased by 1, and its section properties are changed accordingly. The following rules are also applied to simplify the structural design problem:

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

4.3.4 Reward

Let ΔV be an increment of the structural volume due to the agent action. To guide the agent to adjust the structure to satisfy all the constraints in Eq. (4.14), the reward signal at step t is formulated based on the cost of improving the existing structure, computed as

$$\mathrm{R}_{t+1} = -\Delta V[4 - (\Delta\sigma_{\max}^{\mathrm{L}} + \Delta\sigma_{\max}^{\mathrm{S}} + \Delta\theta_{\max} + \Delta\delta_{\max})] \tag{4.18}$$

where $\Delta\sigma_{\max}^{\mathrm{L}}$, $\Delta\sigma_{\max}^{\mathrm{S}}$, $\Delta\theta_{\max}$, and $\Delta\delta_{\max}$ are reward components associated with internal stresses from *long-term loading*, internal stresses from *long and short-term loadings*, inter-story drift angle, and beam center deflection, respectively. In the brace element, the ratio to buckling stress is employed as the constraint value if the element is in compressional state.

Maximum stress usually depends on the section sizes of multiple members and reducing the maximum stress may lead to an increase of the second maximum one. To incorporate the effect of section sizes of several members with large stresses, the p-norm $\left(\|\cdot\|_p\right)$, formulated as $\|\mathbf{x}\|_p = \sqrt[p]{\sum_{i=1}^{n}|x_i|^p}$, is utilized for computing reward components associated with stress constraints. However, the p-norm is not utilized for computing reward components associated with the maximum values of inter-story drift angle and beam center deflection because these constraints mainly depend on the sizes of the specific members, and the constraints can be satisfied by simply reducing the maximum values. Reward components associated with each constraint are formulated as

$$\Delta\sigma_{\max}^{\mathrm{L}} = \begin{cases} \dfrac{\|\boldsymbol{\sigma}^{\mathrm{L}}\|_p^t - \|\boldsymbol{\sigma}^{\mathrm{L}}\|_p^{t+1}}{\|\boldsymbol{\sigma}^{\mathrm{L}}\|_p^{t+1}} & : \text{if } \sigma_{\max}^{\mathrm{L},t} > \hat{\sigma}^{\mathrm{L}} \\ 0 & : \text{else} \end{cases} \tag{4.19}$$

$$\Delta\sigma_{\max}^{\mathrm{S}} = \begin{cases} \dfrac{\left\|\boldsymbol{\sigma}^{\mathrm{S}}\right\|_p^t - \left\|\boldsymbol{\sigma}^{\mathrm{S}}\right\|_p^{t+1}}{\|\boldsymbol{\sigma}^{\mathrm{S}}\|_p^{t+1}} & : \text{if } \sigma_{\max}^{\mathrm{S},t} > \hat{\sigma}^{\mathrm{S}} \text{ or } \lambda_{\max}^t > 1 \\ 0 & : \text{else} \end{cases} \tag{4.20}$$

$$\Delta\theta_{\max} = \begin{cases} \dfrac{\theta_{\max}^t - \theta_{\max}^{t+1}}{\theta_{\max}^{t+1}} & : \text{if } \theta_{\max}^t > \hat{\theta} \\ 0 & : \text{else} \end{cases} \tag{4.21}$$

$$\Delta\delta_{\max} = \begin{cases} \dfrac{\delta_{\max}^t - \delta_{\max}^{t+1}}{\delta_{\max}^{t+1}} & : \text{if } \delta_{\max}^t > 1 \\ 0 & : \text{else} \end{cases} \tag{4.22}$$

4.3.5 Numerical examples

As an RL algorithm, the agent is trained in the training phase to measure the improvement of the reward, and the applicability of the agent is measured in the test phase. To verify that the trained agent can be applied to diverse structures, in this research, the training and test phases utilize small structures and larger ones, respectively. Also, results obtained by RL in the test phase are compared to those obtained by the GA which is a benchmark algorithm in this chapter.

Young's modulus, *short-term*, and *long-term* stress limits of the structural elements are 200 kN/mm^2, 235 N/mm^2, and 235/1.5 = 156.7 N/mm^2, respectively. $\hat{\theta}$ is set to 1/200. Tables 4.4, 4.5, and 4.6 are lists of section properties for columns, beams, and braces, respectively. In these Tables, I and Z denote the moment of inertia and section modulus, respectively. MLP has two hidden layers, each consisting of 200 neurons and the dimensions of h in all GCN layers is 200.

The *long-term loading* consists of the floor weight of 6700 N/m^2 applied on each beam using the floor depth of 6 m. The *short-term loadings* in both directions are computed as described in Section 4.1.1 using a floor weight of 5700 N/m^2. The base shear coefficient C_0, the specific period T_c, and the seismic area coefficient Z_0 are 0.2, 0.6, and 1.0, respectively. The cost coefficient B for braces is set as 1.5 since they increase the complexity of construction. The parameter p for the p-norm is 10. The program is made using Python 3.6 environment. A PC with a CPU of Intel Core i9-11900 (2.5 GHz, 8 cores) and a GPU of Nvidia GeForce RTX3060 12GB is used for computation.

Table 4.4: List of column sections for simultaneous topology and sizing optimization of braced building frames

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

Table 4.5: List of beam sections for simultaneous topology and sizing optimization of braced building frames

index	H-B-t1-t2: (t1<t2)	area (cm^2)	I (cm^4)	Z (cm^3)
0	300×200×8×12	71	11100	756
1	400×300×10×16	133	37900	1940
2	500×300×11×18	159	68900	2820
3	600×300×12×20	187	114000	3890
4	700×300×13×24	232	197000	5640
5	800×300×14×22	234	248000	6270
6	900×300×18×34	360	491000	10800

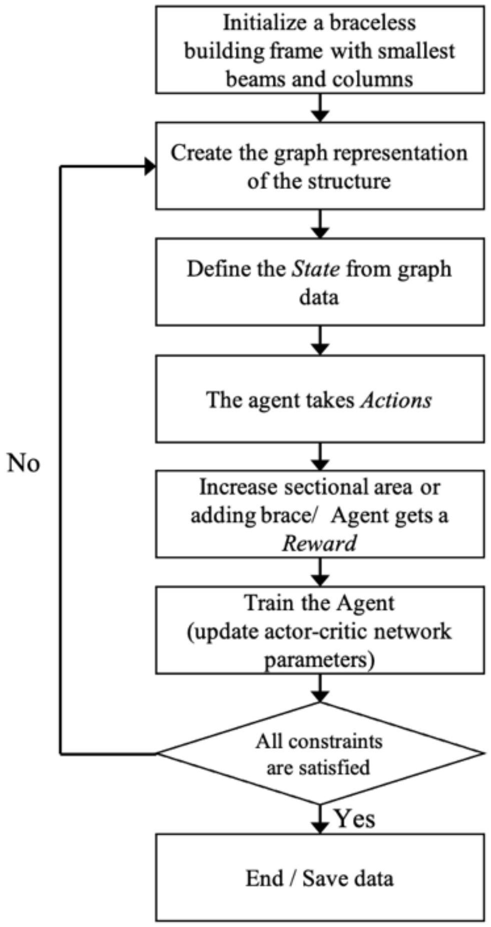
Table 4.6: List of brace sections for simultaneous topology and sizing optimization of braced building frames

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

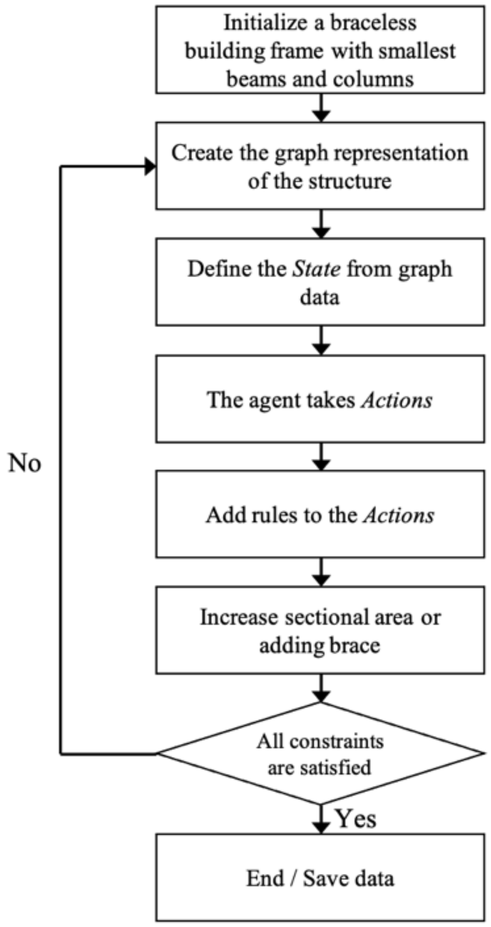
Figure 4.6 illustrates algorithm flowcharts for the training and test phases. During the optimization, each step of MDP is denoted as a step. The loop of MDPs, defined as a game or an episode, is terminated when all constraints are satisfied.

The algorithm in the training phase (Figure 4.6(a)) can be summarized as follows; (0) The finite element model of the unbraced building frame with smallest column and beam indices is initialized. (1) Information of structure is represented using graph representation. (2) Graph representation becomes state data. (3) The agent takes graph representation as input and computes the action. (4) The output from the agent is interpreted into the modification of the building frame. Finite element analysis is performed to evaluate the objective functions, constraints, and structural responses. The reward is computed. (5) The agent is trained by collected state-action-reward, data and the parameters in the agent are updated. (6) All the constraints are checked. If at least one of the constraints is not satisfied, go to step (1); otherwise, terminate the process and save data of the training results.

After the agent has undergone training for a certain number of episodes, it is then applied to optimize other building frames using the algorithm during the testing phase, as shown in Figure 4.6(b). The process can be described as follows; the finite element model of the unbraced building frame with the smallest column and beam indices is initialized. (1) Structural information is encoded through a graph representation. (2) This graph representation is translated into state data. (3) The agent takes this graph-based representation as input and computes an action. (4) To enhance the performance of the agent, a rule is applied to modify the action. (5) The output from the agent is translated into modifications made to the building frame. Subsequently, finite element analysis is conducted to evaluate objective functions, constraints, and structural responses. The reward is then computed. (6) The optimization process includes a validation step wherein all constraints are assessed. If at least one constraint remains violated, the final step is not reached, and the process returns to step (1). However, if all constraints are met, the optimization process concludes, and the data, including the training results and the structural model, is saved.


Initialize a braceless building frame with smallest beams and columns
Create the graph representation of the structure
Define the State from graph data
The agent takes Actions
Increase sectional area or adding brace/ Agent gets a Reward
Train the Agent (update actor-critic network parameters)
All constraints are satisfied
No
Yes
End / Save data

(a)


Initialize a braceless building frame with smallest beams and columns
Create the graph representation of the structure
Define the State from graph data
The agent takes Actions
Add rules to the Actions
Increase sectional area or adding brace
All constraints are satisfied
No
Yes
End / Save data

(b)

Figure 4.6: Flowcharts of the algorithm for simultaneous topology and sizing optimization of braced building frames; (a) Training phase, (b) Test phase

<u>**Training phase**</u>

In each episode of the training phase, the agent is trained to optimize 3-span 3-story steel frames having three predetermined span lengths and story heights, indicated as S1, S2, and S3 in Table 4.7 and Figure 4.7. To measure the performance of the agent, a 4-span 4-story frame with span lengths of {4.0, 6.0, 6.0, 4.0} (m) and story heights of {4.0, 3.0, 3.0, 3.0} (m) (Figure 4.8), is used for measuring improvement of the reward every 10 episodes during the training. The surrogate, as explained in Section 2.3.3 in Chapter 2, policy and value functions are adjusted by the Adam optimizer using the mini-batch size of 32 and the learning rates of 10^{-3} and 10^{-4} for the policy and value functions, respectively, as explained in Section 2.5.3 in Chapter 2. Parameters of the surrogate functions are updated to the real policy and value functions every 100 steps with $\tau = 0.05$, as explained in Section 2.3.3 in Chapter 2. The agent is trained for 3000 episodes. The total number of structural analyses in this phase is 134995.

Table 4.7: Structures for training for simultaneous topology and sizing optimization of braced building frames

structural name	S1	S2	S3
number of spans	3	3	3
span length	{4,4,4} (m)	{8,8,8} (m)	{6,6,6} (m)
number of floors	3	3	3
floor height	{4,4,4} (m)	{3,3,3} (m)	{4,4,4} (m)

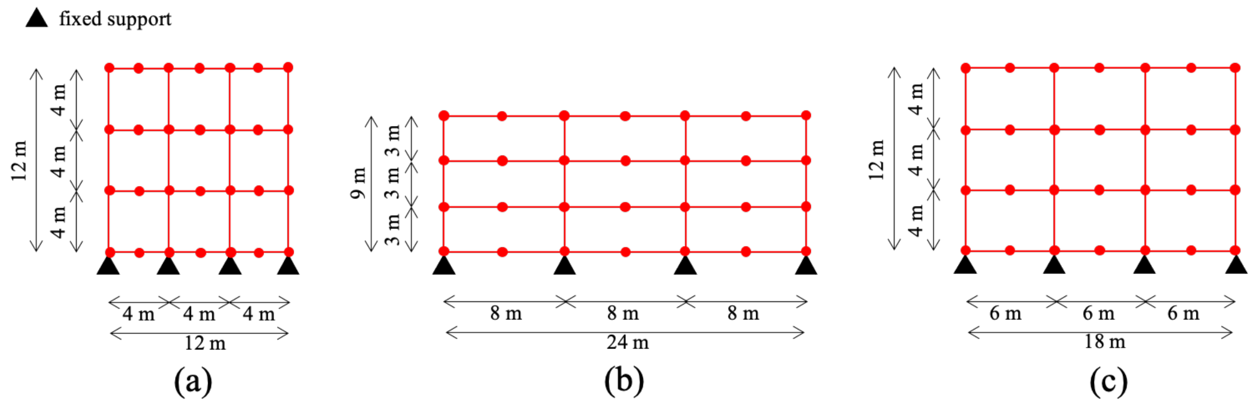


Figure 4.7: Initial structural models during training phase for simultaneous topology and sizing optimization of braced building frames; (a) S1, (b) S2, (c) S3

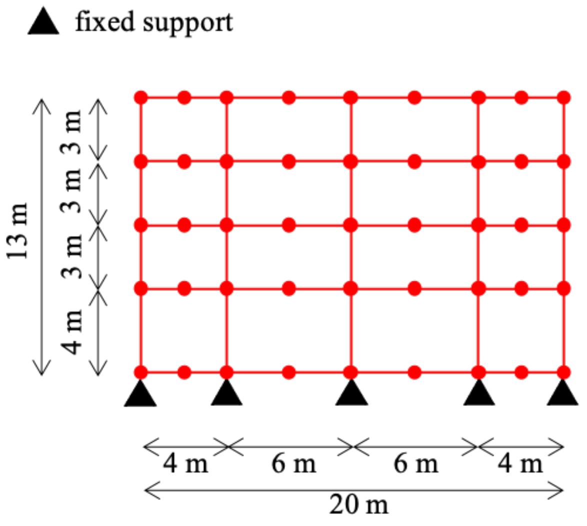


Figure 4.8: Initial structural model for measuring improvement of the agent during training phase for simultaneous topology and sizing optimization of braced building frames

Figures 4.9 and 4.10 show the reward and the final cost of the structure obtained by the agent, respectively, where the horizontal axis is the number of episodes for measuring the agent's improvement and the thick red line in each figure is the moving average of 20 episodes. According to these figures, the agent can increase the obtained reward, reduce the cost of the structure, and maintain its performance throughout the training, indicating the learning ability of the agent. Note that since the structural types S1, S2, and S3 are used for training and the performance is measured using a 4-span 4-story frame and this RL utilizes a policy gradient approach, the agent could not keep on improving the obtained reward using this structure resulting in the fluctuation of the reward and cost.

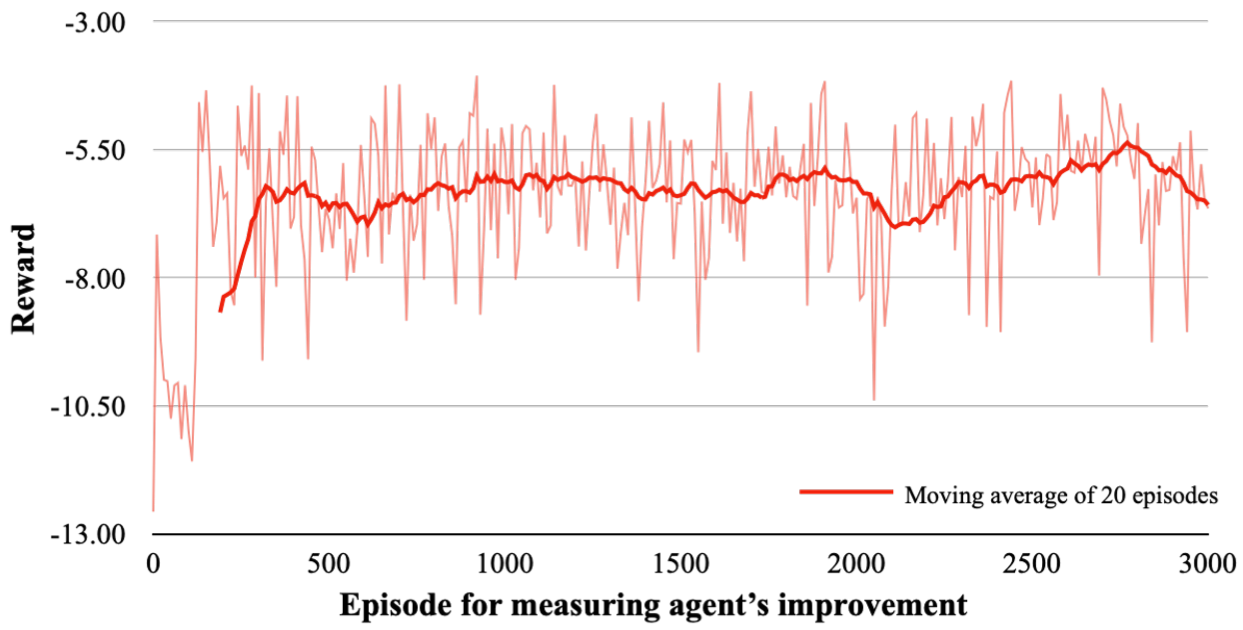


Figure 4.9: Variation of reward and its moving average in training phase for simultaneous topology and sizing optimization of braced building frames

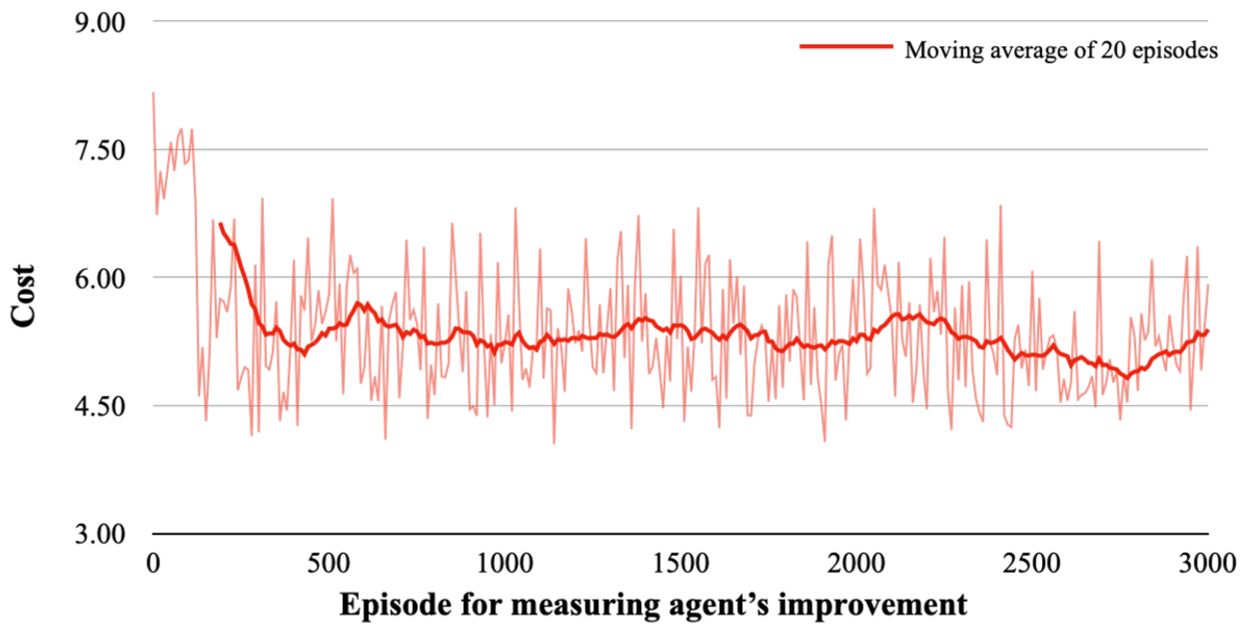


Figure 4.10: Variation of cost and its moving average in training phase for simultaneous topology and sizing optimization of braced building frames

Test phase

In the test phase, an output modification vector $\mathbf{h} \in \mathbb{R}^{n\times 1}$ is introduced for the agent to prevent from choosing unnecessary actions. This vector represents the beam or column that satisfies all constraints, including *long-term loading* stress, *long-term* and *short-term loading* stress, inter-story drift, and deflection constraints. The entry h_i is 0 if the element i is a column or beam that already satisfies every constraint, or a base beam, to prevent the agent from choosing the element that is stiff enough. The element i to be modified is determined from the index i that has the largest value of the Hadamard product of policy function, $\mathbf{g}$, and $\mathbf{h}$ as follows:

$$i = \underset{j}{\operatorname{argmax}}\, \mathbf{g}_j \mathbf{h}_j \boldsymbol{\pi}_j \tag{4.23}$$

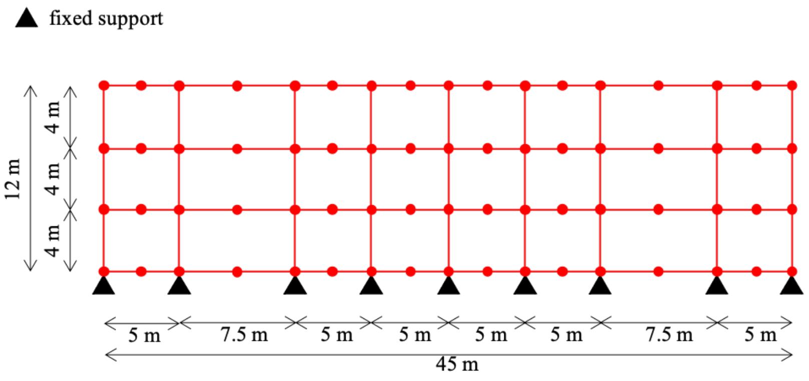
To verify the versatility of the agent, the agent saved from the training phase is applied to the frames in Table 4.8 and Figure 4.11 50 times for each structure, where the section lists and material properties are the same as those in the training phase. Table 4.9 shows the minimum (min.), mean, and standard deviation (std.) of the objective function and the total number of structural analyses of each structure.

Table 4.8: Structures for test for simultaneous topology and sizing optimization of braced building frames

name	number of spans	span length (m)	number of floors	floor height (m)
Structure A	8	{5.0, 7.5, 5.0, 5.0, 5.0, 5.0, 7.5, 5.0}	3	{4.0, 4.0, 4.0}
Structure B	3	{5.0, 5.0, 5.0}	7	{4.0, 4.0, 3.5, 3.5, 3.5, 3.5, 3.5}
Structure C	7	{5.0, 5.0, 7.5, 5.0, 7.5, 5.0, 5.0}	3	{4.0, 3.0, 3.0}
Structure D	3	{5.0, 5.0, 5.0}	4	{4.0, 3.0, 3.0, 3.0}
Structure E	6	{5.0, 7.5, 5.0, 5.0, 7.5, 5.0}	3	{4.0, 3.0, 3.0}
Structure F	3	{5.0, 5.0, 5.0}	5	{4.0, 4.0, 3.0, 3.0, 3.0}
Structure G	5	{5.0, 5.0, 7.5, 5.0, 5.0}	3	{3.0, 3.0, 3.0}
Structure H	4	{5.0, 5.0, 5.0, 5.0}	6	{4.0, 3.0, 3.0, 3.0, 3.0, 3.0}

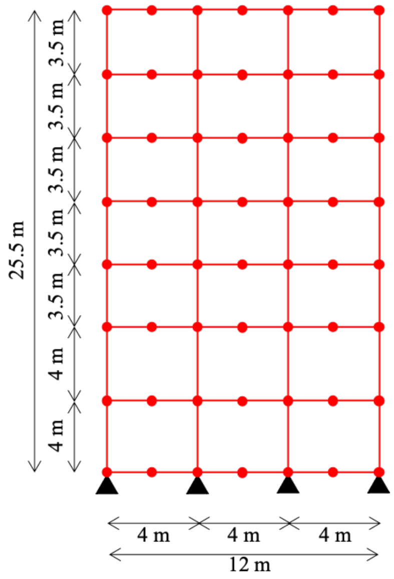
Table 4.9: RL results in test phase (50 trials) for simultaneous topology and sizing optimization of braced building frames

name	min.	mean	std.	number of structural analyses
Structure A	5.61	6.22	0.27	3191
Structure B	5.35	5.98	0.26	3812
Structure C	5.00	5.50	0.20	3237
Structure D	2.51	2.70	0.09	2000
Structure E	4.25	4.67	0.18	2684
Structure F	3.32	3.63	0.14	2515
Structure G	3.35	3.58	0.13	2368
Structure H	5.49	5.85	0.18	3625



fixed support
12 m
4 m
4 m
4 m
5 m
7.5 m
5 m
5 m
5 m
5 m
7.5 m
5 m
45 m

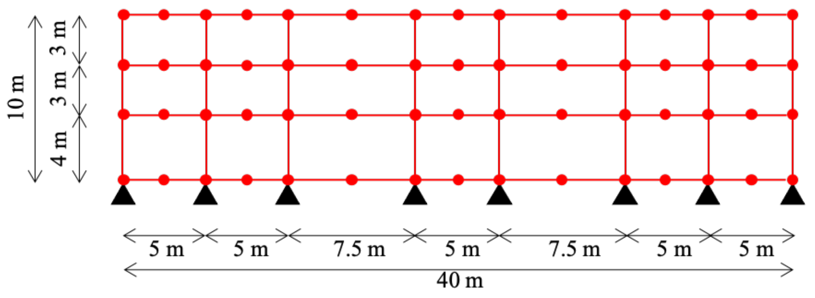
(a)



25.5 m
3.5 m
3.5 m
3.5 m
3.5 m
3.5 m
4 m
4 m
4 m
4 m
4 m
12 m

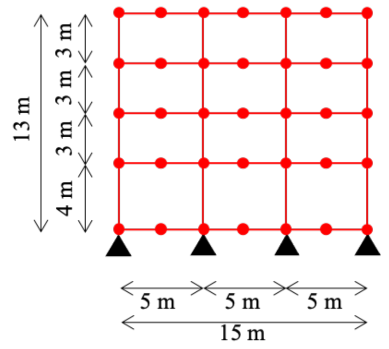
(b)

(b)



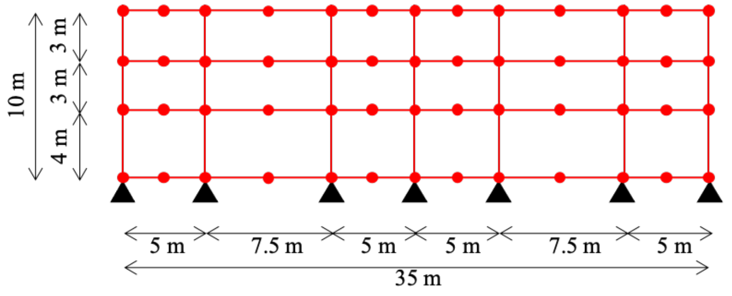
10 m
3 m
3 m
4 m
5 m
5 m
7.5 m
5 m
7.5 m
5 m
5 m
40 m

(c)



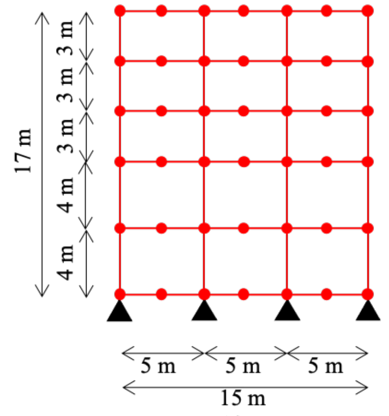
13 m
3 m
3 m
3 m
4 m
5 m
5 m
5 m
15 m

(d)



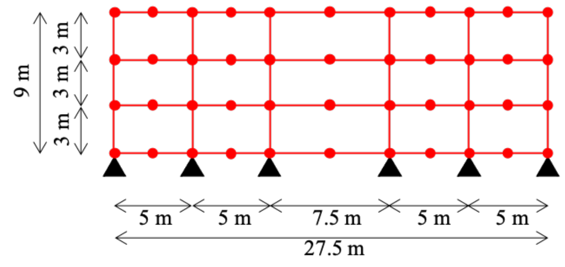
10 m
3 m
3 m
4 m
5 m
7.5 m
5 m
5 m
7.5 m
5 m
35 m

(e)

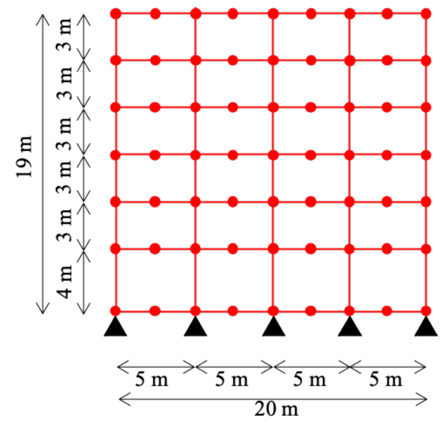


17 m
3 m
3 m
3 m
3 m
4 m
4 m
5 m
5 m
5 m
15 m

(f)



(g)



(h)

Figure 4.11: Initial Structural models during test phase for simultaneous topology and sizing optimization of braced building frames; (a) Structure A, (b) Structure B, (c) Structure C, (d) Structure D, (e) Structure E, (f) Structure F, (g) Structure G, (h) Structure H

<u>Comparison of computation cost and performance with GA</u>

In this research, all genes representing columns, beams, and braces can be crossover. Therefore, all genes should have the same value range. Each gene of GA has an integer value of {0,..,9} so that they can be randomly mixed and mutated. To represent columns, the indices $\{J_1, J_2, \dots, J_{N_\mathrm{f}}\}$ of columns on each floor on the same vertical line are defined by N_f genes $\{G_1, G_2, \dots, G_{N_\mathrm{f}}\}$. G_1 represents the index of the first-floor column. If G_1 is larger than the largest column index (6 in Table 4.4), J_1 is replaced with the largest index. Since the upper column cannot have a larger index than the lower ones, $J_{i\in\{2,\dots,N_\mathrm{f}\}}$ is equal to J_{i-1} if $G_{i-1} < G_i$ and $G_i \in \{0,..,4\}$, whereas $J_{i-1} = J_i - 1$ if $G_{i-1} < G_i$ and $G_i \in \{5,\dots,9\}$.

Since beams on the same floor have the same sections, beam sections on each floor are represented by a single gene which represents the index of the beam section in Table 4.5. Note that if the gene has a larger value than the largest beam index, the largest beam index is assigned to the corresponding beam.

The brace type and brace section in each brace domain are represented by two genes. The first gene has the value of {0,1}, {2,3}, {4,5}, {6,7}, and {8,9} representing no brace, right diagonal brace, left diagonal brace, K-type, and V-type, respectively. The second gene has the value of {0,1}, {2,3}, {4,5}, {6,7}, and {8,9} representing indices 1, 2, 3, 4, 5, of the brace section in Table 4.6, respectively. Similar to the proposed RL method, the number of domains with braces in each story can be only half of the number of brace domains in that story, which is equal to the number of spans for a regular frame in GA.

To compute the fitness function of GA, binary value indicators $\mathrm{c_S}$, $\mathrm{c_L}$, $\mathrm{c_\theta}$, and $\mathrm{c_\delta}$ are introduced for the constraints on maximum short-term, long-term stress, beam deflection, and inter-story drift, respectively. These indicators are equal to 1 if the corresponding constraints are violated, and 0 if satisfied. GA maximizes the objective function defined as follows:

$$G(\mathbf{X}) = -V(\mathbf{X}) - w_0\left(\mathrm{c_S}\frac{w_1\sigma_{\mathrm{max}}^{\mathrm{S}}}{\hat{\sigma}^{\mathrm{S}}} + \mathrm{c_L}\frac{\sigma_{\mathrm{max}}^{\mathrm{L}}}{\hat{\sigma}^{\mathrm{L}}} + \mathrm{c_\theta}\frac{\theta_{\mathrm{max}}}{\hat{\theta}} + \mathrm{c_\delta}\delta_{\mathrm{max}}\right) \quad (4.24)$$

where $w_0 = 5$ and $w_1 = 3$ are the weights of the penalty, which prevent the GA from generating solutions that violate the constraints. These values are decided after some trials and errors.

The GA program is implemented utilizing a Python library DEAP [94]. The total number of analyses of GA is assigned similar to the total number of analyses used in RL. Note that the solution of GA depends on the random seed. Therefore, five GA experiments with different random seeds are conducted on structures in Table 4.8. Table 4.10 summarizes the minimum (min.), mean, and standard deviation (std.) of the objective function, population, generation, and the number of structural analyses of each structure.

Table 4.10: GA results (5 random seeds) for simultaneous topology and sizing optimization of braced building frames

name	min.	mean	std.	number of populations	number of generations	number of structural analyses
Structure A	6.90	7.28	0.46	50	80	4000×5
Structure B	7.34	8.05	0.58	100	50	5000×5
Structure C	7.18	7.36	0.21	50	80	4000×5
Structure D	2.88	3.33	0.42	40	50	2000×5
Structure E	6.39	6.64	0.18	40	80	3200×5
Structure F	4.35	5.09	0.51	60	50	3000×5
Structure G	3.83	4.62	0.48	40	80	3200×5
Structure H	6.41	6.93	0.37	80	50	4000×5

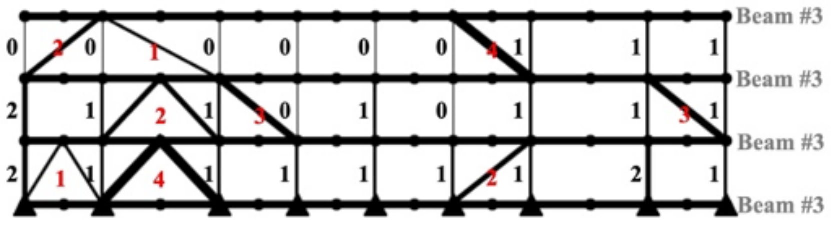
Table 4.11 compares results obtained by RL and GA. The results in Table 4.11 show that the RL agent outperforms GA in obtaining more optimal results for all test structures, as evidenced by the minimum values of the objective function. The results, obtained by the RL agent, from different trials exhibit a consistent value of the objective function, as depicted by the standard deviation value. From a computational standpoint, the trained RL agent requires fewer structural analyses compared to GA, highlighting its efficiency and effectiveness in solving the given problem.

Table 4.11: Comparing results obtained by RL and GA for simultaneous topology and sizing optimization of braced building frames

name	RL			GA		
	min.	mean	std.	min.	mean	std.
Structure A	5.61	6.22	0.27	6.90	7.28	0.46
Structure B	5.35	5.98	0.26	7.34	8.05	0.58
Structure C	5.00	5.50	0.20	7.18	7.36	0.21
Structure D	2.51	2.70	0.09	2.88	3.33	0.42
Structure E	4.25	4.67	0.18	6.39	6.64	0.18
Structure F	3.32	3.63	0.14	4.35	5.09	0.51
Structure G	3.35	3.58	0.13	3.83	4.62	0.48
Structure H	5.49	5.85	0.18	6.41	6.93	0.37

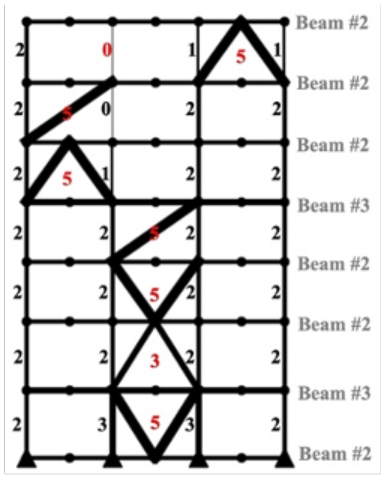
Figures 4.12 and 4.13 respectively illustrate the best solutions of Structures A and B obtained in the test phase by RL agent. The thickness of line is proportional to the index of the section, where numbers on the lines are section size of column (black), beam (grey), and brace (red), and ▲ indicates rigid support. The best solutions of Structures A and B from GA are shown in Figures 4.14 and 4.15, respectively. These figures illustrate the differences between results obtained by RL and GA. In the results obtained by the RL agent, beams and columns exhibit smaller cross-sectional sizes compared to those by GA, whereas both results from RL include more braces than those of GA. Because the cross-sectional areas of braces are smaller than those of beams and columns. Hence, increasing the sections of beams or columns may result in the enlargement of other beams on the same floor or columns on the lower floor, finally leading to a higher value of the objective function. From the perspective of the agent, placing braces could yield a higher reward (lower cost), computed from the objective function value, in each optimization step (i.e., placing braces, increasing sections), compared to merely increasing the sections of beams or columns. Consequently, the RL agent demonstrates a strategy of placing braces rather than enlarging the sections of beams and columns.

In both structures, the most critical constraints in this experiment are the short-term and long-term stress constraints. Figures 4.12-4.15 reveal that the results obtained by the RL agent exhibit higher constraint ratios for the maximum short-term stress, maximum long-term stress constraint, maximum deformation, and maximum inter-story drift constraints, compared to GA, except for the maximum long-term stress constraint in Structure A. These findings indicate that the trained RL agent produces more efficient solutions located near the boundary of the feasible region.



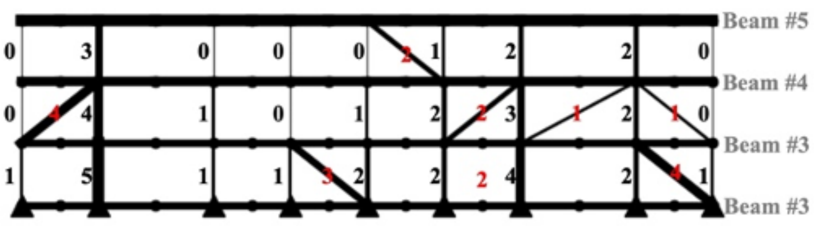
Final cost:	5.61
Maximum short-term stress constraint:	1.00
Maximum long-term stress constraint:	0.85
Maximum deformation constraint:	0.10
Maximum inter-story drift constraint:	0.13

Figure 4.12: RL best result for simultaneous topology and sizing optimization of braced building frames – Structure A



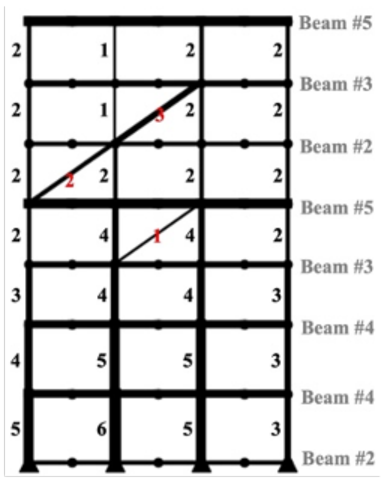
Final cost:	5.35
Maximum short-term stress constraint:	0.97
Maximum long-term stress constraint:	0.82
Maximum deformation constraint:	0.15
Maximum inter-story drift constraint:	0.19

Figure 4.13: RL best result for simultaneous topology and sizing optimization of braced building frames – Structure B



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

Figure 4.14: GA best result for simultaneous topology and sizing optimization of braced building frames – Structure A



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

Figure 4.15: GA best result for simultaneous topology and sizing optimization of braced building frames – Structure B

4.4 Conclusion

This chapter contributes to the field of structural engineering, particularly in the context of seismic-resistant building frame design. The central objective revolves around the introduction and application of a GCN-DDPG agent to tackle the intricate problem of topology and sizing optimization for braced building frames exposed to seismic loads while concurrently minimizing structural costs. The optimization problem is to achieve an optimal balance between structural efficiency and economy by minimizing the structural cost while considering constraints inherent to seismic design. The findings of this chapter are encapsulated as follows:

(1) A novel graph representation of structural elements is introduced which is different from the method proposed by Hayashi et al. [50], representing the structural nodes as the graph node. The proposed representation in this chapter allows for direct manipulation of structural topology and size, with the GCN-DDPG agent serving as the engineer of these modifications. This modification enables a more intuitive and efficient representation of the structure in this optimization problem.

(2) The RL agent demonstrates its ability to learn and optimize the topology and size of braced building frames. Operating within the framework of minimizing structural costs, represented by the structural volume and the cost coefficient for the braces, under a complex set of constraints, the agent relies on a reward function that quantifies changes in structural cost and constraint satisfaction.

(3) One of the standout advantages of the proposed method is its scalability. The GCN-DDPG agent can be effectively trained using small-scale building frames and subsequently applied to larger structures. This scalability feature not only enhances the applicability of the method but also minimizes the computational overhead during the training phase, rendering it an efficient and cost-effective solution. Another noteworthy benefit of the trained RL agent is its ability to achieve optimization results with a reduced computational cost. This feature is particularly advantageous in scenarios where multiple schematic designs of structures are required, allowing for rapid and cost-effective exploration of design alternatives.

(4) The trained RL agent consistently demonstrates its superior performance when compared to conventional optimization techniques such as GA. Another compelling aspect of the proposed RL method is its stability in performance. Numerical results in this chapter show that the agent consistently delivers reliable and efficient optimization outcomes, reducing the need for extensive trial and error iterations that are often encountered in traditional optimization methods.

In conclusion, this chapter proposed optimization methods utilizing RL, for simultaneous topology and sizing optimization problems of building frames utilizing structural cost as the objective function. Different from Chapter 3, this chapter simplifies the graph representation by denoting structural elements as graph nodes and incorporating the internal forces of the element directly in the node feature matrix which also simplifies the DDPG agents. Numerical results indicate the superiority of the proposed RL methods compared to GA.

Chapter 5
Geometry optimization of lattice shells using DDPG and GAT with Bézier surface

5.1 Introduction

The previous chapters of this dissertation have presented a comprehensive exploration of RL as a potent approach for addressing topology and sizing optimization in structural engineering. To complete the overarching optimization framework for truss and frame structures, including topology, sizing, and geometry, this chapter focuses on geometry optimization for lattice shells. In the context of structural design, geometry optimization refers to the process of determining the optimal shape or form of a structure. Specifically, this chapter focuses on lattice shell structures, where the geometry or shape of the shell is determined through the manipulation of the heights of Bézier control points. The primary objective in this context is to minimize strain energy. It is noteworthy that geometry optimization is often characterized as a continuous optimization problem, typically solvable through gradient-based optimization methods albeit at a substantial computational cost. However, the importance of this research lies in the integration of structural information and Bézier control net data into the learning process of the RL agent.

In this geometry optimization problem, the range of permissible heights for Bézier control points is discretized to mitigate computational expenses. Benchmarking against stochastic and population-based methods, namely SA and PSO, the RL method is evaluated in terms of its capacity to generate optimal solutions. The results show the ability of the RL method to outperform SA and PSO, offering a more efficient and effective approach to geometry optimization. Importantly, the RL method maintains the versatility to address lattice shells of varying sizes, configurations, and Bézier control nets. This chapter of the RL-based geometry optimization approach is organized as follows: Section 5.1 provides the foundational understanding by introducing Bézier control nets and outlining the geometry optimization problem employed in this chapter. Section 5.2 explains the RL method formulated for geometry optimization in conjunction with Bézier surfaces, and presents the numerical results that validate its superiority over benchmark methods. Finally, Section 5.3 summarizes the key findings of this research, affirming the potential of the proposed method to solve geometry optimization problems in lattice shell structures.

5.2 Formulation of optimization problem

5.2.1 Bézier surface

Bézier control net and Bernstein basis functions are a type of parametric surface that defines the geometry of the lattice shell, as shown in Figure 5.1. Let b_n be the order of the Bernstein basis function which is defined as

$$B_i^{b_n}(t) = \binom{b_n}{i} u^i (1-u)^{b_n - i}, (\mathrm{i} = 0,1,\dots,b_n) \tag{5.1}$$

$$\text{where } \binom{b_n}{i} = \begin{cases} \frac{b_n!}{i!(b_n - i)!} & \text{for } 0 \leq i \leq b_n \\ 0 & \text{for } i < 0 \text{ or } i > b_n \end{cases} \tag{5.2}$$

where $u \in [0,1]$ is the parameter, and $0^0 = 0! = 1$.

From Eq. (5.1), the tensor product of the Bernstein basis functions $B_i^{c_n}(u)$ and $B_j^{c_m}(v)$ with respect to the parameters $u, v \in [0,1]$ is the Bézier surface computed as

$$\mathbf{B}_{\mathrm{b}}^{c_n,c_m}(u,v) = \sum_{i=0}^{c_n} \sum_{j=0}^{c_m} \mathbf{C}_{i,j}\, B_i^{c_n}(u) B_j^{c_m}(v) \tag{5.3}$$

where $\mathbf{C}_{i,j} (i = 1,..,c_n; j = 1,..,c_m)$ are the coordinates of the vertices of the control net (i.e., the coordinates of control points). c_n and c_m are the orders of basis functions in the u and v directions, respectively. Note that the values of u and v are scaled nodal coordinates on the horizontal axes in the range [0,1] and the nodal heights are defined by the control points as $\mathbf{B}_{\mathrm{b}}^{c_n,c_m}(u_i, v_i)$ using Eq. (5.3) where (u_i, v_i) are the parameter values of node i.

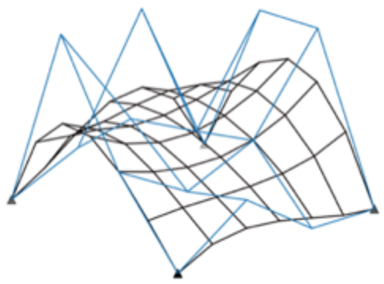


Figure 5.1: Bézier surface of the lattice shell (black lines) defined by Bézier control net (blue lines)

5.2.2 Geometry optimization of lattice shells with Bézier surface

Parametric surfaces are effective for representing the nodal height of the structure to obtain a smooth geometry of lattice shells. Utilizing these surfaces can also reduce the number of design variables in the optimization problem [130] and structural analysis [131]. A type of parametric surface that has most attracted researchers is the Bézier surface which defines the geometry through the tensor product of the Bernstein polynomials [132] and their control points. Pioneering research on geometry optimization of shells using Bézier surface can be found in Ref. [133] and is further refined in later years [134-136].

In this research, the Bézier control net determines the nodal heights of the lattice shell which consists of beam elements with 12 DoFs in the main axes of the grid and truss elements with 6 DoFs for bracing (i.e., diagonal elements in the grid). The numbers of grids in the Bézier control net and those of the lattice structure are independent of each other and can be determined separately based on designers' and engineers' preferences. This chapter focuses on the problem of minimizing the strain energy of lattice shells stiffened by diagonal braces subjected to static loads where the boundary conditions are predetermined for each structural node.

The total strain energy is computed using linear structural analysis. Let $\mathbf{k}_\mathrm{f} \in \mathbb{R}^{12\times 12}$ and $\mathbf{k}_\mathrm{e} \in \mathbb{R}^{6\times 6}$ denote stiffness matrices of the frame element and the brace element in the local coordinate system, respectively. The global stiffness matrix $\mathbf{K} \in \mathbb{R}^{n_\mathrm{D}\times n_\mathrm{D}}$ is the assembly of $\mathbf{k}_\mathrm{f}$ and $\mathbf{k}_\mathrm{e}$ into the global coordinate system, where n_D is the total number of DoFs, after specifying the boundary conditions. Let $\mathbf{p} \in \mathbb{R}^{n_\mathrm{D}}$ be the nodal load vector in the global coordinate system considering point loads and weights applied on every node and element, respectively. $\mathbf{d} \in \mathbb{R}^{n_\mathrm{D}}$ is the nodal displacement vector computed by solving the stiffness equation as

$$\mathbf{Kd} = \mathbf{p} \tag{5.4}$$

The total strain energy E is obtained from

$$E = \frac{1}{2}\mathbf{d}^{\mathrm{T}}\mathbf{Kd} \tag{5.5}$$

To optimize the total strain energy using Bézier control points as design variables, the geometry of a lattice shell with n^{free} nodes and $n_{\mathrm{b}}^{\mathrm{free}}$ control points is optimized by modifying the heights of control points. The vector of nodal heights and the vector of movable heights of Bézier control points are denoted as $\mathbf{z} = (z_1, z_2, \dots, z_{n^{\mathrm{free}}})$ and $\mathbf{z}^{\mathrm{b}} = (z_1^b, z_2^b, \dots, z_{n_{\mathrm{b}}^{\mathrm{free}}}^b)$, and the heights of supports are fixed. The vector of nodal heights is derived from the heights of control points as $\mathbf{z}(\mathbf{z}^{\mathrm{b}})$ when the values of parameters (u, v) are assigned to each node. Let z_{max}, z_{min}, $z_{\mathrm{min}}^{\mathrm{b}}$, and $z_{\mathrm{max}}^{\mathrm{b}}$ denote the upper bound of nodal height, the lower bound of nodal height, the predetermined upper bound value of the heights of Bézier control points, and the predetermined lower bound value of the heights of Bézier control points, respectively.

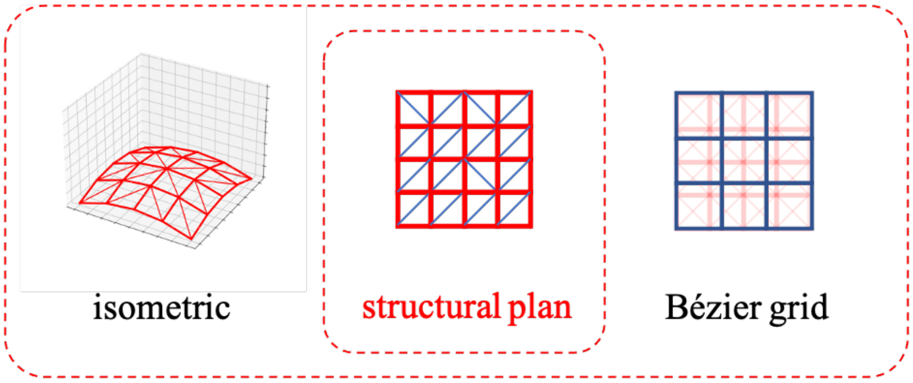
The global stiffness matrix of the structure and global displacement vector are functions of $\mathbf{z}^{\mathrm{b}}$, denoted as $\mathbf{K}(\mathbf{z}^{\mathrm{b}})$ and $\mathbf{d}(\mathbf{z}^{\mathrm{b}})$, respectively, and the geometry optimization problem to minimize the total strain energy (i.e., to maximize the stiffness of the structure), is defined as follows:

$$\begin{aligned} \text{minimize} \quad & E(\mathbf{z}^{\mathrm{b}}) = \frac{1}{2}\mathbf{d}(\mathbf{z}^{\mathrm{b}})^{\mathrm{T}}\mathbf{K}(\mathbf{z}^{\mathrm{b}})\mathbf{d}(\mathbf{z}^{\mathrm{b}}) \\ \text{subject to} \quad & z_{\mathrm{min}} \le z_i(\mathbf{z}^{\mathrm{b}}) \le z_{\mathrm{max}} \quad \left(i = 1, 2, \dots, n^{\mathrm{free}}\right) \\ & z_{\mathrm{min}}^{\mathrm{b}} \le z_j^{\mathrm{b}} \le z_{\mathrm{max}}^{\mathrm{b}} \left(j = 1, 2, \dots, n_{\mathrm{b}}^{\mathrm{free}}\right) \end{aligned} \tag{5.6}$$

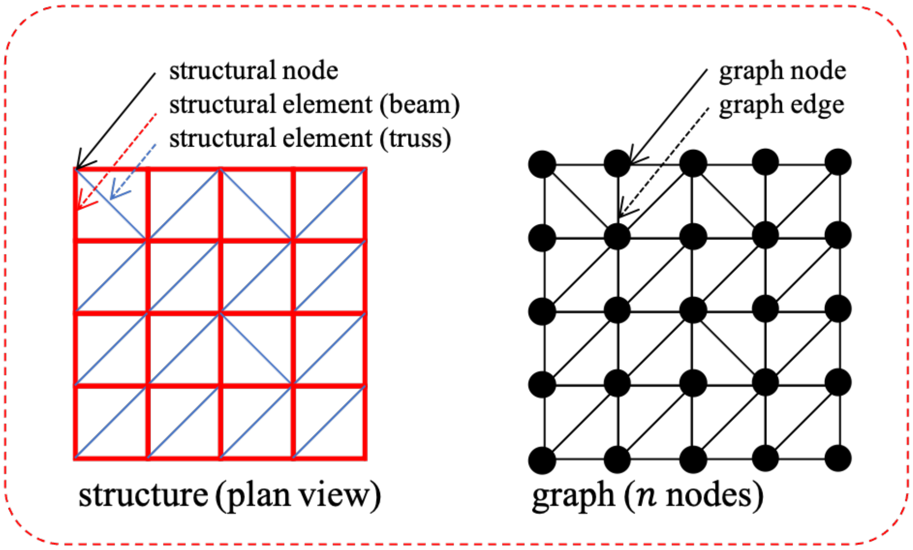
5.3 Geometry optimization of lattice shells using DDPG and GAT with Bézier surface

This section introduces the RL method for solving problems in Section 5.1.2. The RL agent in this problem is designed to process both data from the structure and the control nets. Since the modification of the geometry is done by modifying the control points, the output of the RL agent, made of GATs, is the embedded signal in the Bézier domain. Therefore, two graph representations of the structure and Bézier control net are presented. Figure 5.2 illustrates the graph representation of the lattice shell structure, whose geometry is defined by the Bézier control net (Figure 5.2(a)). The structure is represented by a graph (Figure 5.2(b)) similar to the method utilized in Section 3.2. Node feature matrices, adjacency matrices, and weighted adjacency matrices are utilized to represent structural configurations and internal forces (Figure 5.2(c)).

Figure 5.3 shows the graph representation of the Bézier control net. The configuration of the control net is represented as a graph where the graph nodes and edges represent Bézier control points and nets, respectively (Figure 5.3(b)). Information of the Bézier control points including coordinates and boundary conditions is stored in the graph node and represented using a node feature matrix. Information of the control net is represented using the adjacency matrix (Figure 5.3 (c)).



(a)



(b)

$$\mathbf{N} = \left.\overbrace{\begin{bmatrix} n_{1,1} & \cdots & n_{1,f} \\ \vdots & \ddots & \vdots \\ n_{n,1} & \cdots & n_{n,f} \end{bmatrix}}^{f \text{ features}}\right\} n \text{ nodes}$$

Node feature matrix

$$\mathbf{M} = \left.\overbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & \cdots & \vdots \\ 0 & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}}^{n \text{ nodes}}\right\} n \text{ nodes}$$

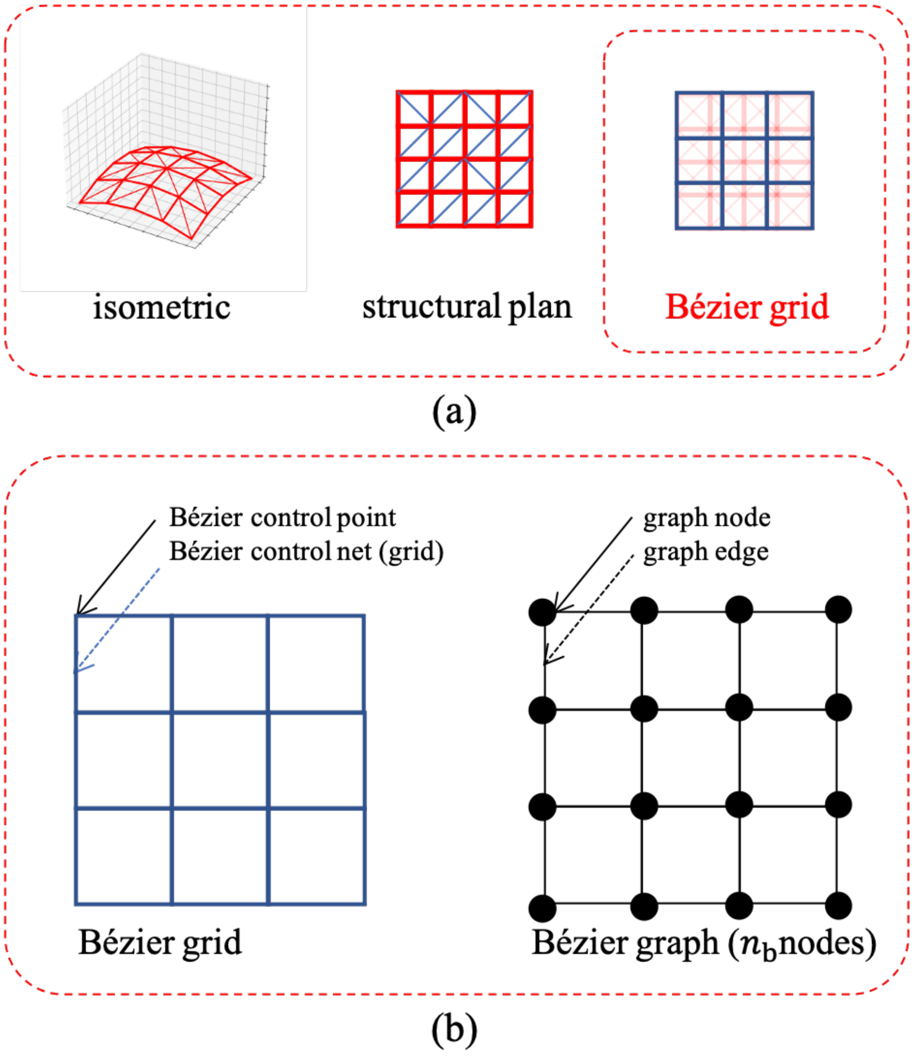
Adjacency matrix

$$\mathbf{P} = \left.\overbrace{\begin{bmatrix} 0 & p_{1,2} & 0 & 0 \\ p_{2,1} & 0 & \cdots & \vdots \\ 0 & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}}^{n \text{ nodes}}\right\} n \text{ nodes}$$

Weighted adjacency matrix

(c)

Figure 5.2: Graph representation of the lattice shell for geometry optimization of lattice shells



$$\mathbf{N}_\mathrm{b} = \left.\overbrace{\begin{bmatrix} n_{\mathrm{b}_{1,1}} & \cdots & n_{\mathrm{b}_{1,f}} \\ \vdots & \ddots & \vdots \\ n_{\mathrm{b}_{n,1}} & \cdots & n_{\mathrm{b}_{n,f}} \end{bmatrix}}^{f\ \text{features}}\right\} n_\mathrm{b}\ \text{nodes}$$

Node feature matrix

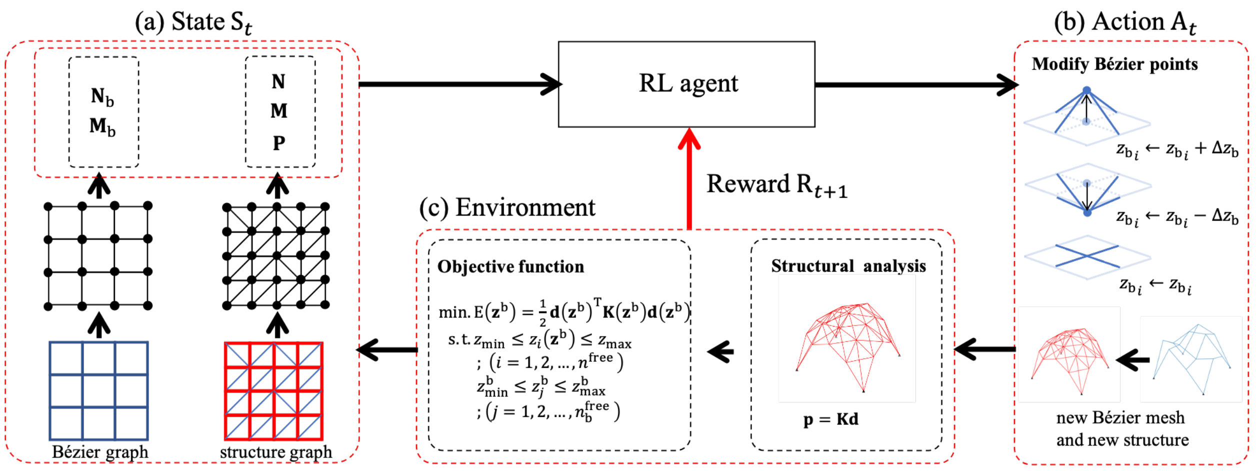
$$\mathbf{M}_\mathrm{b} = \left.\overbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & \cdots & \vdots \\ 0 & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}}^{n_\mathrm{b}\ \text{nodes}}\right\} n_\mathrm{b}\ \text{nodes}$$

Adjacency matrix

(c)

Figure 5.3: Graph representation of the Bézier control points for geometry optimization of lattice shells

The MDP of the geometry optimization of the lattice shells using the Bézier surface is illustrated in Figure 5.4. At time step t, the agent observes the state S_t which is the graph representation of the current structural configuration and Bézier surface illustrated in Figure 5.4(a). The agent does the action A_t which determines the vertical adjustment of the Bézier control points shown in Figure 5.4(b). The new structural geometry is defined from the modified Bézier surface (i.e., environment) and the objective function is evaluated. The agent obtains a reward R_{t+1} based on the change of the strain energy as shown in Figure 5.4(c) and the new step of MDP begins.



(a) State S_t
$\mathbf{N}_\mathrm{b}$
$\mathbf{M}_\mathrm{b}$
N
M
P
Bézier graph
structure graph
RL agent
(b) Action A_t
Modify Bézier points
$z_{\mathrm{b}_i} \leftarrow z_{\mathrm{b}_i} + \Delta z_\mathrm{b}$
$z_{\mathrm{b}_i} \leftarrow z_{\mathrm{b}_i} - \Delta z_\mathrm{b}$
$z_{\mathrm{b}_i} \leftarrow z_{\mathrm{b}_i}$
Reward R_{t+1}
(c) Environment
Objective function
$\min.\ \mathrm{E}(\mathbf{z}^\mathrm{b}) = \frac{1}{2}\mathbf{d}(\mathbf{z}^\mathrm{b})^\mathrm{T}\mathbf{K}(\mathbf{z}^\mathrm{b})\mathbf{d}(\mathbf{z}^\mathrm{b})$
$\mathrm{s.t.}\ z_\mathrm{min} \le z_i(\mathbf{z}^\mathrm{b}) \le z_\mathrm{max}$
$;\ (i = 1, 2, \dots, n^\mathrm{free})$
$z^\mathrm{b}_\mathrm{min} \le z^\mathrm{b}_j \le z^\mathrm{b}_\mathrm{max}$
$;\ (j = 1, 2, \dots, n^\mathrm{free}_\mathrm{b})$
Structural analysis
$\mathbf{p} = \mathbf{Kd}$
new Bézier mesh
and new structure

Figure 5.4: MDP of the geometry optimization of lattice shells

5.3.1 State

In this study, each graph node, derived from the structure, has five features, similar to those in Section 3.2.1, and is represented as the node feature matrix $\mathbf{N} \in \mathbb{R}^{n\times 5}$. Adjacency matrices and weighted adjacency matrices represent structural elements and their internal forces, respectively ($\mathbf{M}_1$, $\mathbf{M}_2$, $\mathbf{M}_3$, $\mathbf{P}_1$, and $\mathbf{P}_2$). In addition to the structural nodes and elements, the attributes of Bézier control points are represented as a graph of the Bézier control net. The control net feature matrix and the control net adjacency matrix are represented as $\mathbf{N}_\mathrm{b} \in \mathbb{R}^{n_\mathrm{b}\times 5}$ and $\mathbf{M}_\mathrm{b} \in \mathbb{R}^{n_\mathrm{b}\times n_\mathrm{b}}$, respectively.

Note that all values in these matrices of graph representations are in the range of [0,1], which mitigates the risk of numerical instability during the training of GAT models. Unlike GCN, no degree matrix is utilized for normalizing the adjacency matrices for the GAT models.

Table 5.1 summarizes entries in the node feature matrix **N**, and Table 5.2 shows entries in the adjacency matrix of beam elements $\mathbf{M_1}$, the adjacency matrix of truss elements $\mathbf{M_2}$, the adjacency matrix of beam and truss elements $\mathbf{M_3}$, the weighted adjacency matrix of beam $\mathbf{P_1}$, and the weighted adjacency matrix of truss $\mathbf{P_2}$. Tables 5.3 and 5.4 indicate entries in the node feature matrix of Bézier control points $\mathbf{N_b}$ and the control net adjacency matrix $\mathbf{M_b}$, respectively.

Table 5.1: Entries in node feature matrix (**N**) for geometry optimization of lattice shells

$n_{i,1}$	x-coordinate of node i
$n_{i,2}$	y-coordinate of node i
$n_{i,3}$	z-coordinate of node i
$n_{i,4}$	1 if the node has fixed support; else 0
$n_{i,5}$	1 if the node has no support; else 0

Table 5.2: Entries in adjacency and weighted adjacency matrices ($\mathbf{M_1}$, $\mathbf{M_2}$, $\mathbf{M_3}$, $\mathbf{P_1}$, and $\mathbf{P_2}$) for geometry optimization of lattice shells

${m_1}_{i,\mathrm{j}}$	1 if there is a beam element connecting nodes i and j; else 0
${m_2}_{i,\mathrm{j}}$	1 if there is a truss element connecting nodes i and j; else 0
${m_3}_{i,\mathrm{j}}$	${m_1}_{i,\mathrm{j}} + {m_2}_{i,\mathrm{j}}$
${p_1}_{i,j}$	ratio between the bending moment (b') and the axial force (a') of an element if there is a beam element connecting nodes i and j; else 0, where ${p_1}_{i,j} = ({m_1}_{i,\mathrm{j}}\,|b'|/(|a'| + 1)$
${p_2}_{i,j}$	axial force (a') of an element if there is a truss element connecting nodes i and j; else 0, where ${p_2}_{i,j} = {m_2}_{i,\mathrm{j}}|a'|$

Table 5.3: Entries in control net feature matrix ($\mathbf{N_b}$) for geometry optimization of lattice shells

$n_{i,1}$	x-coordinate of control point i
$n_{i,2}$	y-coordinate of control point i
$n_{i,3}$	z-coordinate of control point i
$n_{i,4}$	1 if the control point i has fixed support; else 0
$n_{i,5}$	1 if the control point i has no support; else 0

Table 5.4: Entries in the control net adjacency matrix ($\mathbf{M}_\mathrm{b}$) for geometry optimization of lattice shells

$m_{\mathrm{b}_{ij}}$	1 if there is a control net connecting control points i and j; else 0

5.3.2 Agent

The agent is the DDPG having policy and value functions made of GAT layers. The output of a GAT layer is transformed by activation functions. In general, the ReLU activation function is applied to all layers of both policy and value functions except the last layer of the policy function which is transformed by the Sigmoid activation function, scaling the output to be within [0,1]. This output represents the probability of taking action A_t^i in a state S_t.

The policy function $\boldsymbol{\pi}$ takes state data ($\mathbf{N}, \mathbf{M}_1, \mathbf{M}_2, \mathbf{M}_3, \mathbf{P}_1, \mathbf{P}_2, \mathbf{M}_\mathrm{b}, \mathbf{N}_\mathrm{b}$), explained in Section 5.2.1, as input to compute the output $\boldsymbol{\pi} \in \mathbb{R}^{n_\mathrm{b}\times 3}$. This output has the same number of rows as those of the control net feature matrix $\mathbf{N}_\mathrm{b} \in \mathbb{R}^{n_\mathrm{b}\times 5}$, as shown in Table 5.3, and is interpreted as the modification of the control point heights. Note that the number of columns (i.e., embedded node signal) in $\boldsymbol{\pi}$ is set to 3 each of which indicates the probability of adjusting the control point heights in different ways as further explained in Section 5.3.3.

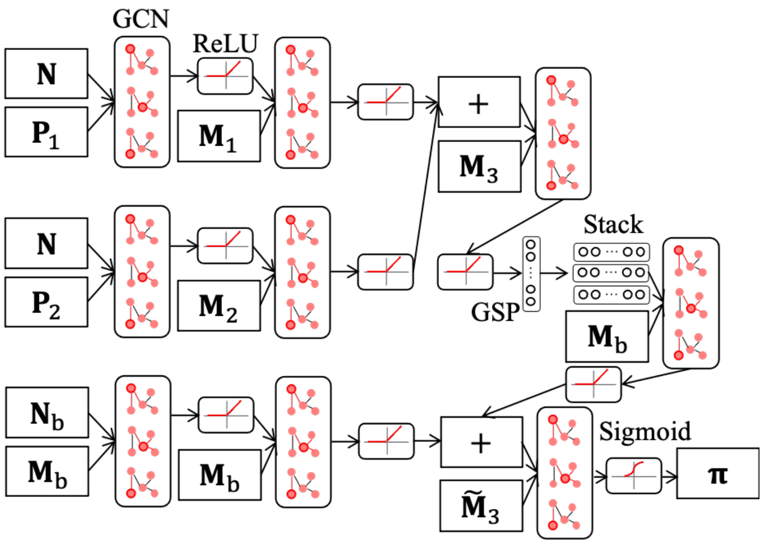
Two operations are introduced in this research to combine multiple graph signals with different dimensions (i.e., from n-node graph signals to n_b-node graph signals). The first operation is the GSP operation, as explained in Section 2.6.1 in Chapter 2, and the second operation is the stack operation which copies row vectors, made by GSP, into a matrix with n_b rows, as explained in Section 2.6.2 in Chapter 2.

Value function Q computes an estimation of the accumulated reward (i.e., Q-value) using state data and the output from the policy function as input. Similar to Chapters 3 and 4, GSP and an MLP are utilized for transforming the output matrix of the last GAT layers of the value function into a scalar value, and computing the estimation of accumulated reward, respectively.

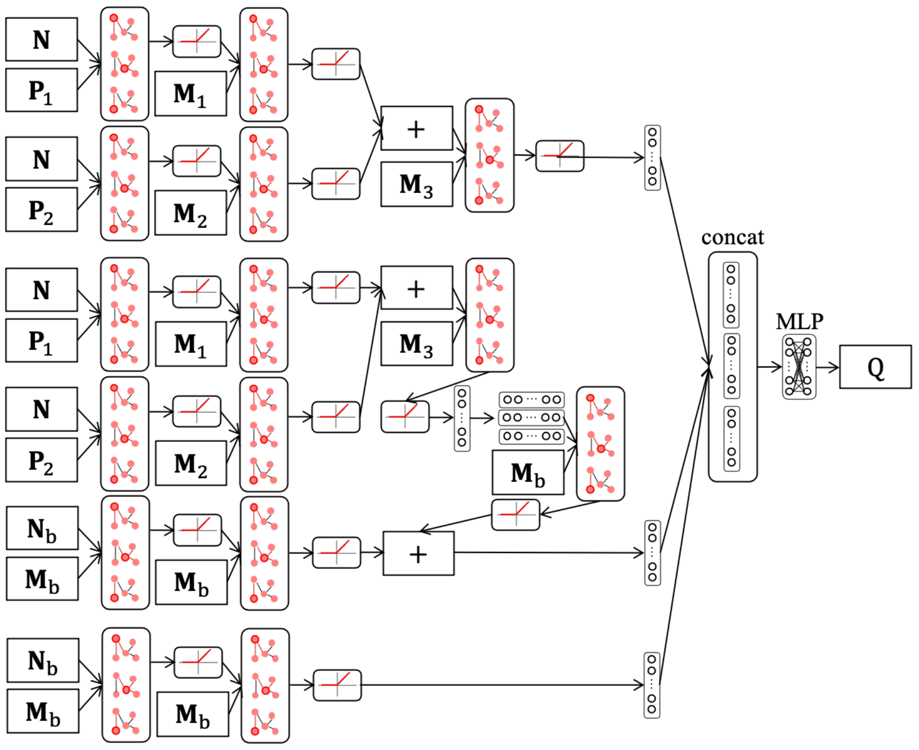
Table 5.5 summarizes the computation processes of the policy and value functions of the GAT-DDPG agent for geometry optimization. Figure 5.5 illustrates actor and critic networks in this research. Note that all equations and symbols in the computation processes are explained in Section 2.6 and Figure 2.6 in Chapter 2.

Table 5.5: Policy and value functions of GAT-DDPG for geometry optimization of lattice shells

policy function $\boldsymbol{\pi}$	value function Q
input: $\mathbf{N}$, $\mathbf{M}_1$, $\mathbf{M}_2$, $\mathbf{M}_3$, $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{M}_\mathrm{b}$, $\mathbf{N}_\mathrm{b}$	input: $\mathbf{N}$, $\mathbf{M}_1$, $\mathbf{M}_2$, $\mathbf{M}_3$, $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{M}_\mathrm{b}$, $\mathbf{N}_\mathrm{b}$, $\boldsymbol{\pi}$
computation: step 1: $\mathbf{N}_{1.1} = \mu(\mu(\mathbf{N}, \mathbf{P}_1), \mathbf{M}_1)$ $\mathbf{N}_{1.2} = \mu(\mu(\mathbf{N}, \mathbf{P}_2), \mathbf{M}_2)$ $\mathbf{N}_{1.3} = \mu(\mathbf{N}_{1.1} + \mathbf{N}_{1.2}, \mathbf{M}_3)$ step 2: $\mathbf{N}_{1.4} = \mathrm{Stack}^{n_\mathrm{b}}(\mathrm{GSP}(\mathbf{N}_{1.3}))$ $\mathbf{N}_1 = \mu(\mathbf{N}_{1.4}, \mathbf{M}_\mathrm{b})$ step 3: $\mathbf{N}_2 = \mu(\mu(\mathbf{N}_\mathrm{b}, \mathbf{M}_\mathrm{b}), \mathbf{M}_\mathrm{b})$ step 4: $\boldsymbol{\pi} = \sigma(\mathbf{N}_1 + \mathbf{N}_2, \mathbf{M}_\mathrm{b})$	computation: step 1: $\mathbf{N}_{1.1} = \mu(\mu(\mathbf{N}, \mathbf{P}_1), \mathbf{M}_1)$ $\mathbf{N}_{1.2} = \mu(\mu(\mathbf{N}, \mathbf{P}_2), \mathbf{M}_2)$ $\mathbf{n}_1 = \mathrm{GSP}(\mu(\mathbf{N}_{1.1} + \mathbf{N}_{1.2}, \mathbf{M}_3))$ step 2: $\mathbf{N}_{2.1} = \mu(\mu(\mathbf{N}, \mathbf{P}_1), \mathbf{M}_1)$ $\mathbf{N}_{2.2} = \mu(\mu(\mathbf{N}, \mathbf{P}_2), \mathbf{M}_2)$ $\mathbf{N}_{2.3} = \mu(\mathbf{N}_{2.1} + \mathbf{N}_{2.2}, \mathbf{M}_3)$ $\mathbf{N}_{2.4} = \mathrm{Stack}^{n_\mathrm{b}}(\mathrm{GSP}(\mathbf{N}_{2.3}))$ $\mathbf{N}_{2.5} = \mu(\mu(\mathbf{N}_\mathrm{b}, \mathbf{M}_\mathrm{b}), \mathbf{M}_\mathrm{b})$ $\mathbf{n}_2 = \mathrm{GSP}(\mathbf{N}_{2.4} + \mathbf{N}_{2.5})$ step 3: $\mathbf{n}_3 = \mathrm{GSP}(\mu(\mu(\boldsymbol{\pi}, \mathbf{M}_\mathrm{b}), \mathbf{M}_\mathrm{b}))$ step 4: $\mathbf{n}_4 = [\mathbf{n}_1 \parallel \mathbf{n}_2 \parallel \mathbf{n}_3]$ step 5: $\mathrm{Q} = \mathrm{f}_{\mathrm{NN}}(\mathbf{n}_4)$
output: $\boldsymbol{\pi} \in \mathbb{R}^{n_\mathrm{b} \times 3}$	output: $\mathrm{Q} \in \mathbb{R}^{1 \times 1}$



(a) Actor network



(b) Critic network

Figure 5.5: Actor and critic networks for geometry optimization of lattice shells

5.3.3 Action

After obtaining the output of the policy function $\boldsymbol{\pi}$ that has n_{b} rows and three columns, all n_{b} Bézier control points are adjusted based on their associate entries in $\boldsymbol{\pi}$. The adjustment of a Bézier control point i includes moving the control point upward, moving it downward, or keeping it at the same height, all of which is the action at step t, interpreted as follows:

$$\mathrm{A}_t^i = \begin{cases} z_i^{\mathrm{b}} \leftarrow z_i^{\mathrm{b}} + \Delta z^{\mathrm{b}} & : \text{if } \max(\boldsymbol{\pi}_{i,1}, \boldsymbol{\pi}_{i,2}, \boldsymbol{\pi}_{i,3}) = \boldsymbol{\pi}_{i,1} \\ z_i^{\mathrm{b}} \leftarrow z_i^{\mathrm{b}} - \Delta z^{\mathrm{b}} & : \text{if } \max(\boldsymbol{\pi}_{i,1}, \boldsymbol{\pi}_{i,2}, \boldsymbol{\pi}_{i,3}) = \boldsymbol{\pi}_{i,2} \\ z_i^{\mathrm{b}} \leftarrow z_i^{\mathrm{b}} & : \text{if } \max(\boldsymbol{\pi}_{i,1}, \boldsymbol{\pi}_{i,2}, \boldsymbol{\pi}_{i,3}) = \boldsymbol{\pi}_{i,3} \end{cases} \tag{5.7}$$

5.3.4 Reward

This chapter utilizes a reward function similar to those in Section 3.2.4. Let E_0 and E_t denote the initial strain energy and the strain energy at step t of the structure, respectively. Reward signal R_{t+1} is computed from the variation of the strain energy as

$$\mathrm{R}_{t+1} = (E_t - E_{t+1})/E_0 \tag{5.8}$$

5.3.5 Numerical examples

Similar to RLs in other chapters, the agent is trained using small structures in the training phase and its performance is evaluated using larger structures in the test phase. In the test phase, the agent performance is benchmarked with PSO and SA methods. In the RL agent, the number of feature maps h in all GCN layers, explained in Section 2.5 in Chapter 2, is 200. MLP has two hidden layers, each consisting of 200 neurons.

The lattice shells in this problem have a 1.0 m by 1.0 m grid, consisting of 12-DoF frame elements and 6-DoF truss elements for main grids and braces, respectively. The frame element has a hollow cylindrical section with an external diameter of 34 mm, a thickness of 2.3 mm, and a weight of 1.8 kg/m. The truss element has a solid circular section with a diameter of 12 mm and a weight of 0.9 kg/m. Young's modulus of both elements is 200 kN/mm^2. A vertical point load of 1 kN is applied to each node in the structure.

The program is developed using Python 3.6 environment. The computation is done on a PC with a CPU of Intel Core i5-6600 (3.3 GHz, four cores) and a GPU of AMD Radeon R9 M395 2 GB. Figure 5.6 shows the flowcharts of the algorithms for training (a) and test (b) phases. The optimization process, denoted as a game (i.e., episode), ends when the number of steps reaches the predetermined limit, and the data of the structural configurations including nodal coordinates, connections, properties of the elements, and loads are saved as an output.

The algorithm in the training phase (Figure 5.6(a)) is as follows; (0) The flat lattice shell is initialized with random bracing direction in each grid. (1) Information on the structure and Bézier control point are represented as graph data. (2) Graph representation becomes state data. (3) The agent takes graph representation as input to compute the action. (4) The heights of Bézier control points are modified according to the action. Geometry of the structure is also modified and the finite element analysis is performed to obtain the objective functions and structural responses. The reward is computed from the change of the objective function. (5) The agent is trained by collected state-action-reward data, and the parameters in the agent are optimized. (6) If the final step is not reached go to step (1); otherwise, terminate the process and save data.

Once the agent has undergone training for a certain number of games, it is then applied to optimize the geometry of other lattice shells utilizing the algorithm during the testing phase (Figure 5.6(b)). The process is explained as follows; (0) The lattice shell is initialized as the flat shape. (1) Structure and Bézier control point are translated into graph representations. (2) This graph representation is transformed into state data. (3) The agent, utilizing the state data as input, computes an action. (4) Geometry of the structure is modified according to the modification of the Bézier control points from the action. If the objective function is improved, then the structure is modified; otherwise, the lattice shell and Bézier control points are reverted to those before the modification. (5) If the final step is not reached return to step (1); otherwise, terminate the process, and save the data.

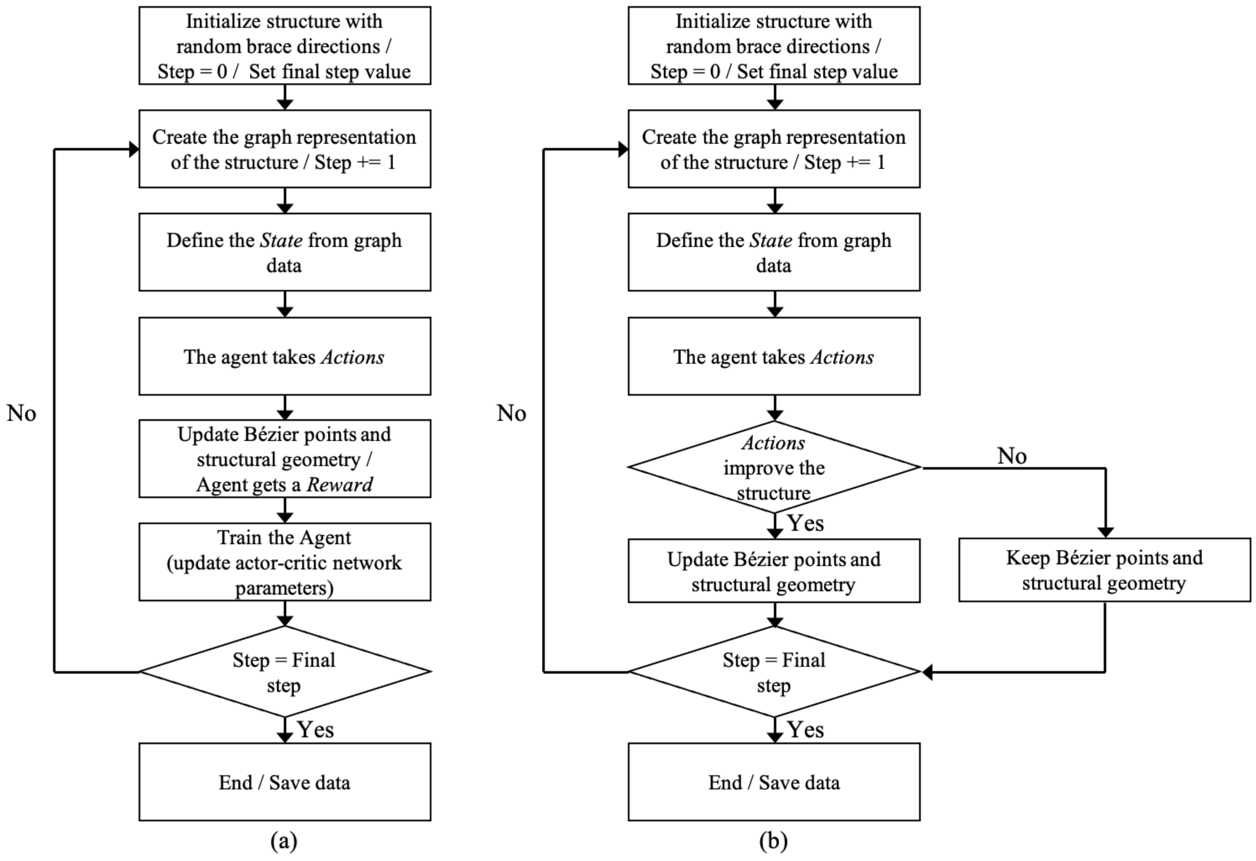

Initialize structure with random brace directions / Step = 0 / Set final step value
Create the graph representation of the structure / Step += 1
Define the State from graph data
The agent takes Actions
Update Bézier points and structural geometry / Agent gets a Reward
Train the Agent (update actor-critic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

Figure 5.6: Flowcharts of the algorithm for geometry optimization of lattice shells; (a) Training phase, (b) Test phase

<u>Training phase</u>

A 4×4-grid lattice shell structural model is initialized randomly in each training game. These training models are chosen from the structural models shown in Figure 5.7. The bracing direction in each grid is also randomly initialized and every node in the structure is initialized on a plane (i.e., height of 0). The heights of supports are fixed at 0.

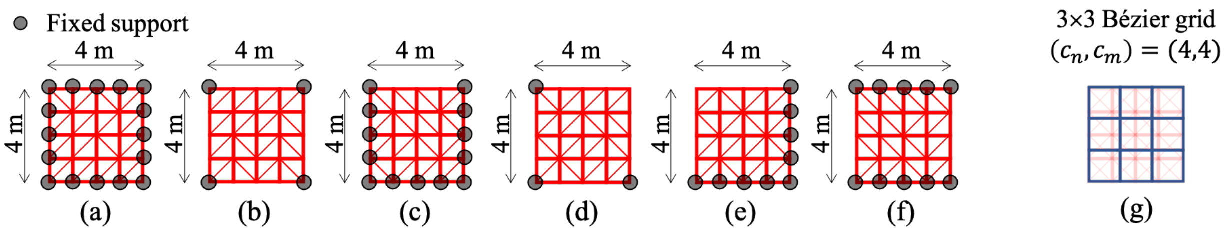


Figure 5.7: Structural models during training phase for geometry optimization of lattice shells

In this training phase, each game has a maximum number of steps of 20, and the heights of Bézier control points are adjusted at each step according to the actions of the agent with $\Delta z^{\mathrm{b}} = 0.1$ m so that the agent can thoroughly explore all possible configurations of the heights of control points. The values of $z_{\min}$, $z_{\max}$, $z^{\mathrm{b}}_{\min}$, and $z^{\mathrm{b}}_{\max}$ are 0, 1 m, 0, and 2 m, respectively. The replay buffer for training surrogate functions, as explained in Section 2.3.3 in Chapter 2, has a mini-batch size of 32. Adam optimizer, as explained in Section 2.5.3 in Chapter 2, modifies trainable parameters of the surrogate policy and value functions with the learning rates of 10^{-6} and 10^{-5}, respectively. Trainable parameters of surrogate functions are updated to online functions with $\tau = 0.05$, as explained in Section 2.3.3 in Chapter 2. The agent has been trained through 1000 games. In the value function, MSE, as explained in Eq. (2.23) in Chapter 2, is used as the loss function.

Figure 5.8 represents the accumulated reward obtained in the game and the game number using the vertical and horizontal axes, respectively. The thick red line indicates the moving average of the accumulated reward with a window size set to 50. From this figure, the moving average of the reward has increased with a fluctuation period during the first 400 games of training, then maintained relatively high values afterward. Note that the structural models are changed every training game. Hence, the history of maintaining high rewards implies that the trained agent is capable of optimizing the geometry of structures with different structural models and different bracing directions. This capability is further investigated in the test phase. The number of structural analyses of GAT-DDPG in the training phase is 20000.

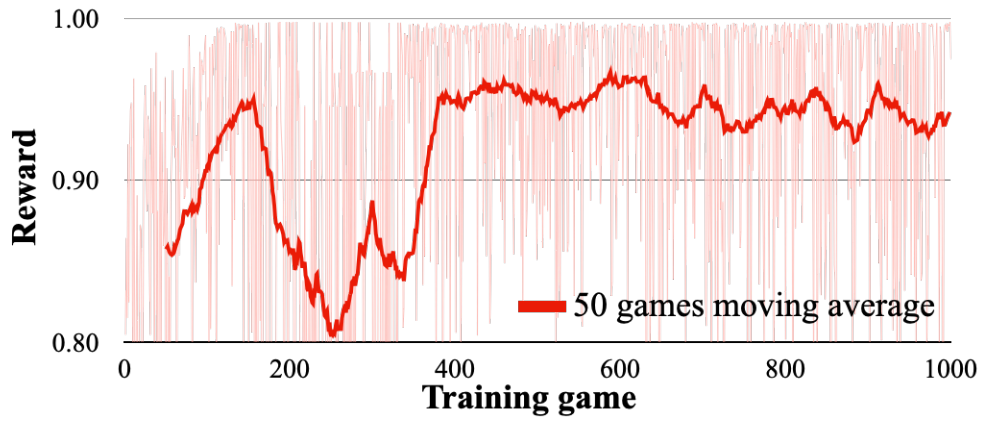


Figure 5.8: Variation of reward and its moving average in training phase for geometry optimization of lattice shells

<u>**Test phase**</u>

In this phase, the trained agent is applied to large and diverse structures to verify its applicability for geometry optimization problems. Three experiments are conducted to verify the ability of the agent compared to other algorithms, the usefulness of the proposed method for geometry optimization, and the ability of the agent when the type of design variables and initial geometry are changed.

(1) Verifying ability of the agent for geometry optimization compared to PSO and SA

In this experiment, the trained agent is applied to six fully braced 6×6-grid lattice shells with three cases of c_n and c_m of Bernstein basis functions as shown in Figure 5.9, and two fully braced 10×10-grid lattice shells with one case of c_n and c_m, as shown in Figure 5.10. The number of steps for the optimization is 500 and 1000 for 6×6-grid and 10×10-grid shells, respectively. $z_{\min}$ and $z^{\mathrm{b}}_{\min}$ are 0 for every structure. In 6×6-grid shells, $z_{\max}$ and $z^{\mathrm{b}}_{\max}$ are 1 m and 2 m, respectively. In 10×10-grid shells, $z_{\max}$ and $z^{\mathrm{b}}_{\max}$ are 2 m and 4 m, respectively. Δz^{b} is set to 0.1 m for both 6×6 and 10×10-grid shells.

Optimization is carried out for each structure by RL ten times, and the results are compared to those obtained from PSO and SA. Numbers of populations and generations of PSO are set based on the number of structural analyses of GAT-DDPG. In this study, the number of particles in PSO is 20, and the algorithm is terminated after 10 and 20 steps of optimization for 6×6-grid and 10×10-grid structures, respectively. The PSO algorithm is made from DEAP Python library which implements the original PSO algorithm proposed by Poli et al. [29]. Five schemes of weight coefficients for PSO, (ϕ_1, ϕ_2) = (0.5,0.5), (0.4,0.6), (0.6,0.4), (0.2,0.8), and (0.8,0.2) are utilized. The SA algorithm is the dual annealing from the Python library named SciPy. SA is terminated at the specified maximum number of iterations that is also set so that its computational cost is equivalent to that of GAT-DDPG which are 1000 and 2000 iterations for 6×6-grid and 10×10-grid structures, respectively. Table 5.6 shows the computational costs of RL (10 times), PSO (5 schemes) with 5 trials using different random seeds, and SA with 5 trials using different random seeds for each grid size.

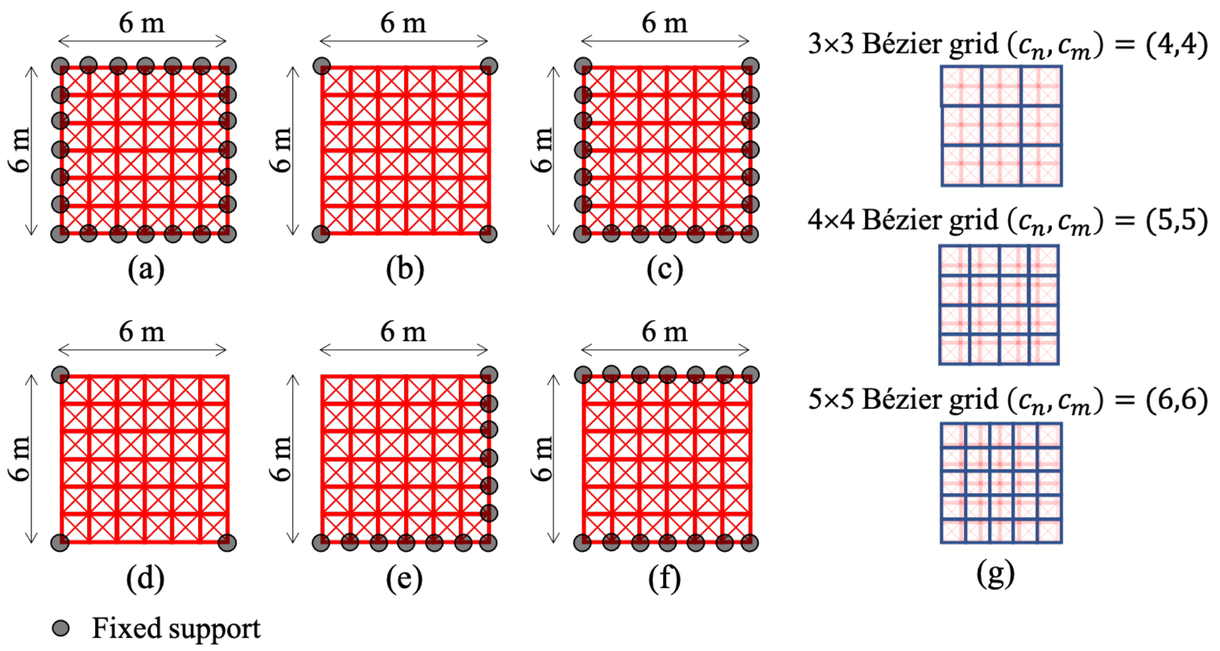

6 m
(a)
(b)
(c)
(d)
(e)
(f)
(g)
3×3 Bézier grid $(c_n, c_m) = (4,4)$
4×4 Bézier grid $(c_n, c_m) = (5,5)$
5×5 Bézier grid $(c_n, c_m) = (6,6)$
● Fixed support

Figure 5.9: 6×6-grid lattice shells (fully braced) with (c_n, c_m) = (4,4), (5,5), and (6,6) for geometry optimization of lattice shells; (a) structural model 1, (b) structural model 2, (c) structural model 3, (d) structural model 4, (e) structural model 5, (f) structural model 6, (g) Bézier control nets

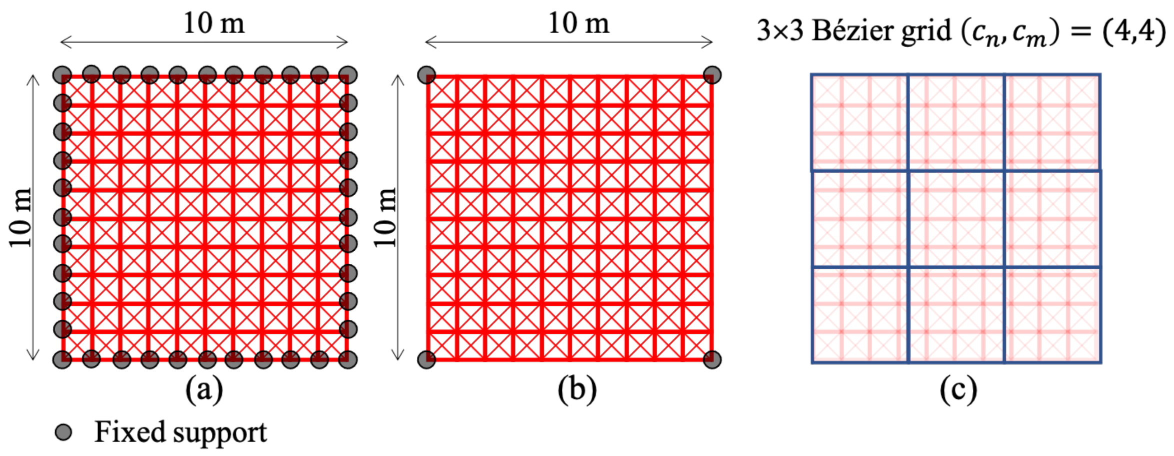


Figure 5.10: 10×10-grid lattice shells (fully braced) with $(c_n, c_m) = (4,4)$ for geometry optimization of lattice shells; (a) structural model 1, (b) structural model 2, (c) Bézier control net

Table 5.6: Total computational cost of each method for geometry optimization of lattice shells

grid size	RL (10 tests)	PSO (5 schemes, 5 trials)	SA (5 trials)
6×6	5000	5000	5000
10×10	10000	10000	10000

Table 5.7 indicates the best results of RL, PSO, and SA in each structural model and case of Bernstein basis functions. The obtained results from RL are competitive compared to those from PSO and SA. The experiment verifies the quality of the solutions of the GAT-DDPG agent. It is worth noting that the RL method can obtain good results even though the structural grid size and Bézier grid size differ from those in the training phase.

Table 5.7: Strain energy of the best results obtained from RL, PSO, and SA for geometry optimization of lattice shells

structural model	grid size	(c_n, c_m)	RL (N·m)	PSO (N·m)	SA (N·m)
1	6×6	(4,4)	3.41	3.43	3.40
		(5,5)	2.77	2.94	2.87
		(6,6)	2.73	3.03	2.77
	10×10	(4,4)	21.92	22.08	21.92
2	6×6	(4,4)	65.21	80.96	70.88
		(5,5)	60.14	135.38	53.49
		(6,6)	75.79	97.33	54.20
	10×10	(4,4)	424.97	1017.94	437.22
3	6×6	(4,4)	7.30	7.82	7.28
		(5,5)	6.73	6.89	6.95
		(6,6)	7.07	9.65	7.59
4	6×6	(4,4)	259.41	492.97	274.85
		(5,5)	280.22	715.01	281.22
		(6,6)	372.72	732.34	278.29
5	6×6	(4,4)	195.15	206.85	200.07
		(5,5)	196.86	271.62	192.23
		(6,6)	182.71	260.00	194.73
6	6×6	(4,4)	9.32	14.14	10.62
		(5,5)	9.47	11.39	10.78
		(6,6)	10.91	16.02	12.23

The best results of RL in 10×10-grid shells with structural models 1 and 2 using $(c_n, c_m) = (4,4)$ are illustrated in Figures 5.11 and 5.12, respectively. From these figures, the trained RL agent can minimize the strain energy by adjusting structural geometry through Bézier control points and obtain smooth geometries despite different structural configurations from those used in the training phase are considered, confirming that the agent can be trained using small structural models and deployed into larger ones. The figures also show that strain energy reduction in the early steps is large since the initial geometry is flat while the strain energy reduction is small in the later steps of optimization (Figures 5.11(e) and 5.12(e)).

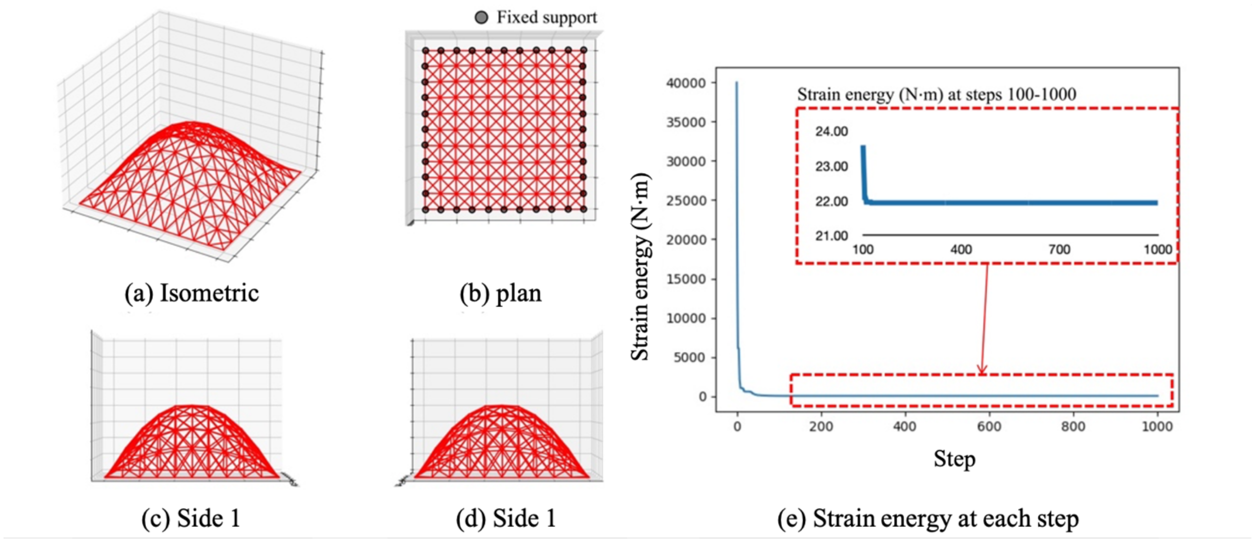


Figure 5.11: 10×10-grid shell for geometry optimization of lattice shells – Structural model 1

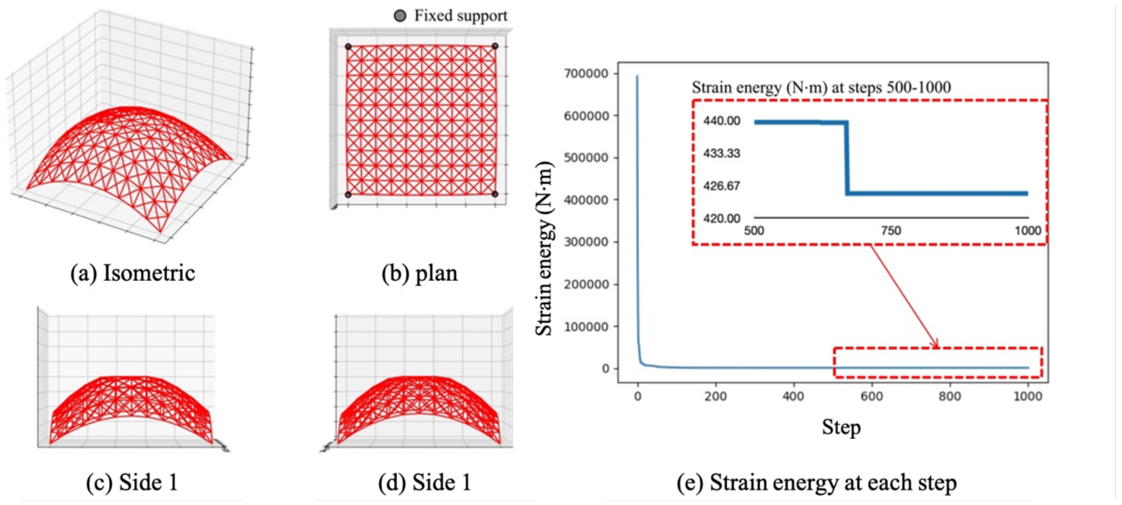


Figure 5.12: 10×10-grid shell for geometry optimization of lattice shells – Structural model 2

(2) Verifying the usefulness of the agent for geometry optimization

This experiment aims to confirm the usefulness of the agent for the geometry optimization of lattice shells focusing on the computational cost. The trained agent is applied to a 20×20-grid lattice shell using the structural model 1 with one case of Bernstein basis functions where $(c_n, c_m) = (6,6)$, as illustrated in Figure 5.13. The number of steps, $z_{\min}$, $z_{\max}$, $z^{\mathrm{b}}_{\min}$, $z^{\mathrm{b}}_{\max}$, and Δz^{b} are set as 400, 0, 4 m, 0, 8 m, and 0.1 m, respectively.

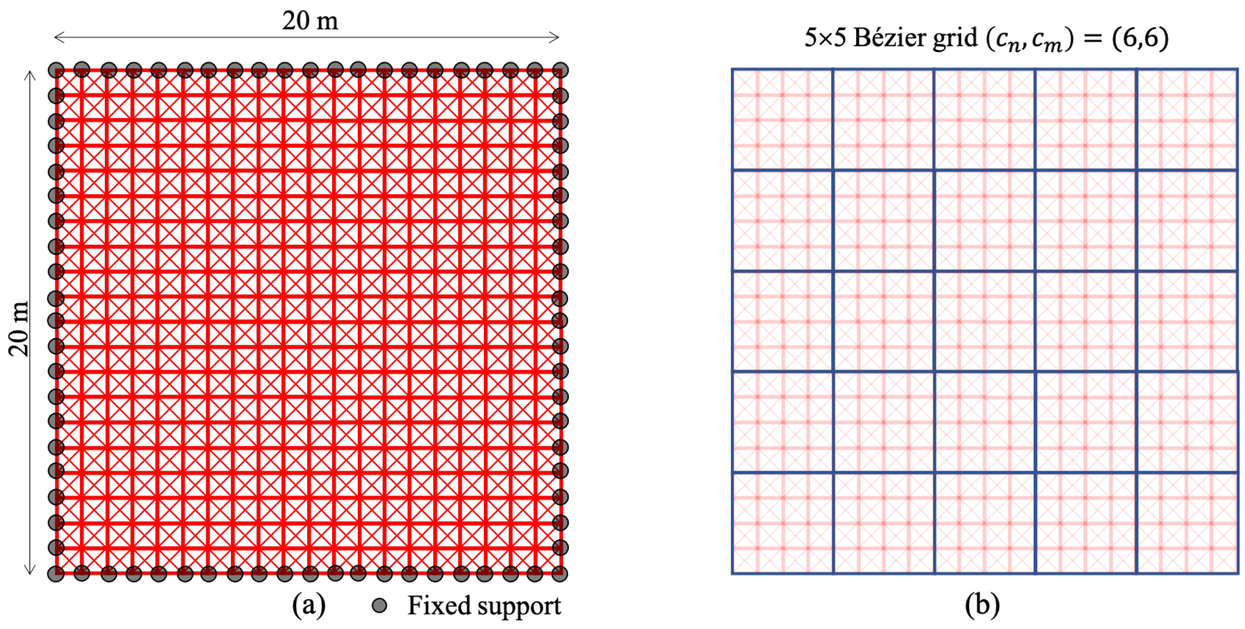


Figure 5.13: 20×20-grid lattice shell (fully braced) with $(c_n, c_m) = (6,6)$ for geometry optimization of lattice shells; (a) Structural model 1, (b) Bézier control net

Results of RL obtained from 10 tests are compared with those of PSO and SA utilizing similar parameters, schemes, and trials as in the test phase. The number of particles and optimization steps in PSO are 20 and 40, respectively. The maximum number of SA iterations is 4000. The total number of structural analyses of the RL method is 4000 while those of PSO and SA are 20000 and 20000, respectively. Note that the computational costs of PSO and SA are larger than those of RL to confirm the usefulness of the RL for geometry optimization using small computational costs.

The comparison between the best result of RL and the best results obtained by PSO and SA among different random seeds is shown in Table 5.8. The RL-based approach yields a better result than PSO and SA. It is worth noting that the RL method utilizes smaller computational costs and can be effective when there are multiple structures to be optimized; e.g., in the preliminary design process. The final geometry and the history of strain energy of the best result obtained by RL is illustrated in Figure 5.14.

Table 5.8: Strain energy of the best results obtained from RL, PSO, and SA for geometry optimization of lattice shells

structural model	grid size	(c_n, c_m)	RL (N·m)	PSO (N·m)	SA (N·m)
1	20×20	(6,6)	284.94	294.49	291.94

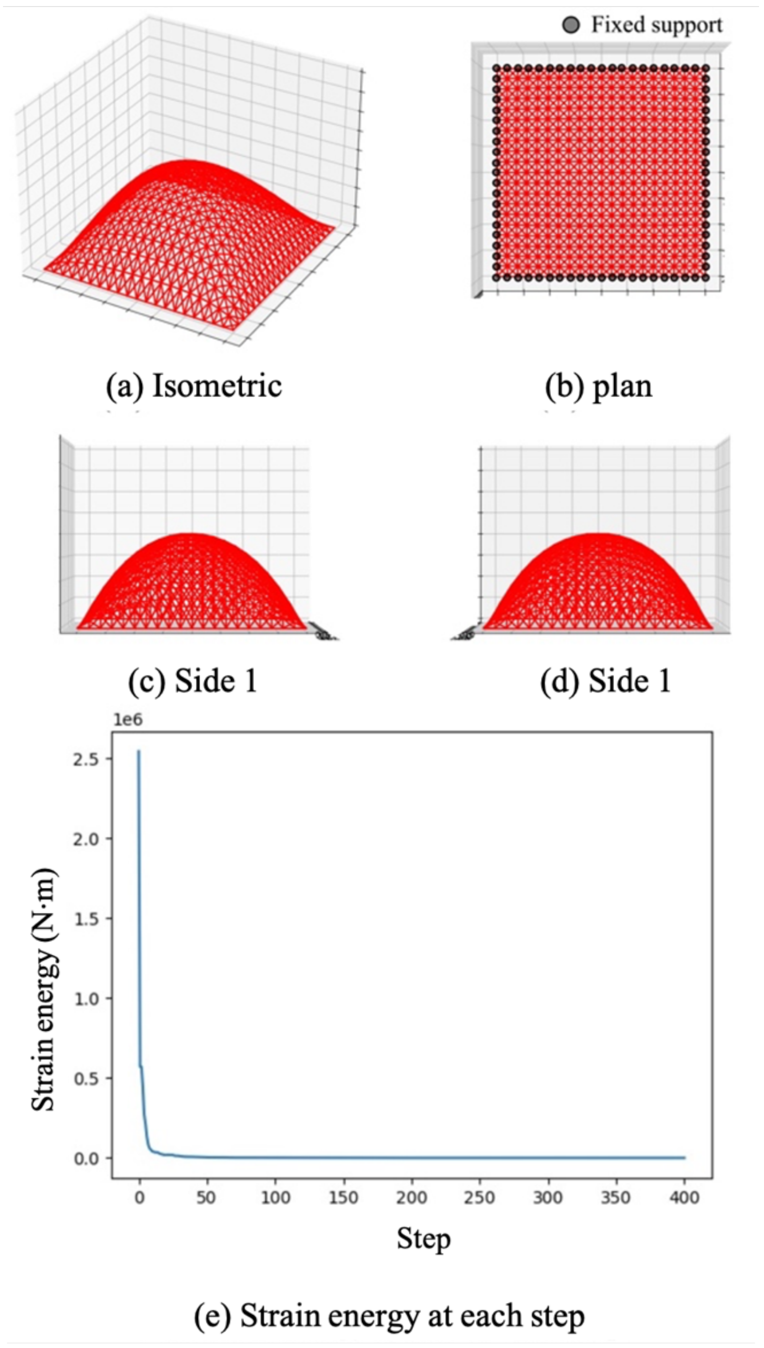


Figure 5.14: 20×20-grid shell for geometry optimization of lattice shells - Verifying the usefulness of the agent for geometry optimization – Structural model 1

(3) Verifying ability of the agent when the type of design variables and initial geometry are changed

This experiment presents the applicability of the already trained RL agent for solving geometry optimization of fully braced structures that have initial irregular geometries using different types of design variables. The structural models are 10×10-grid lattice shells with structural models 1 and 2 and $c_n = c_m = 3$ for the Bernstein basis functions, similar to Figure 5.10. The number of steps, $z_{\min}$, $z_{\max}$, $z_{\min}^{\mathrm{b}}$, and $z_{\max}^{\mathrm{b}}$ are 200 steps, 0, 2 m, 0, and 4 m respectively. Results of ten trials are presented for each structural model.

In this experiment, the Bézier control points are randomly initialized in the range [0,2] m. The action is also modified so that the agent at step t adjusts the Bézier control point i using the value directly derived from the output of the policy function $\boldsymbol{\pi}$, computed as

$$\mathrm{A}_t^i = \begin{cases} z_i^{\mathrm{b}} \leftarrow z_i^{\mathrm{b}} + \left(\pi_{i,1}\Delta z^{\mathrm{b}}\right) & : \text{if } \max\left(\pi_{i,1}, \pi_{i,2}, \pi_{i,3}\right) = \pi_{i,1} \\ z_i^{\mathrm{b}} \leftarrow z_i^{\mathrm{b}} - \left(\pi_{i,2}\Delta z^{\mathrm{b}}\right) & : \text{if } \max\left(\pi_{i,1}, \pi_{i,2}, \pi_{i,3}\right) = \pi_{i,2} \\ z_i^{\mathrm{b}} \leftarrow z_i^{\mathrm{b}} & : \text{if } \max\left(\pi_{i,1}, \pi_{i,2}, \pi_{i,3}\right) = \pi_{i,3} \end{cases} \tag{5.9}$$

where Δz^{b} is 0.1 m. Note that the design variables in this section are continuous whereas those of the training phase and other test phases are discrete.

Table 5.9 shows the minimum (min.), mean, and standard deviation (std.) of the strain energy for each structural model. In structural model 1, the best result has strain energy similar to those in Table 5.7 using the same structural model of training and test phase. In structural model 2, the best result has strain energy larger than those in Table 5.7 by around 10%. However, the computational cost of this section is 20% of those in Table 5.7 because the number of steps is smaller in this Section. Figures 5.15 and 5.16 illustrate the initial geometry, the final geometry, and the history of strain energy of the best results of 10×10-grid lattice shells with structural models 1 and 2, respectively. These figures emphasize that the RL method yields reasonably optimized geometries despite using different initial geometries and types of design variables.

Table 5.9: Results from initial irregular geometries for geometry optimization of lattice shells

structural model	grid size	(c_n, c_m)	min. (N·m)	mean (N·m)	std. (N·m)
1	10×10	(4,4)	21.90	21.92	0.02
2	10×10	(4,4)	468.23	621.06	70.19

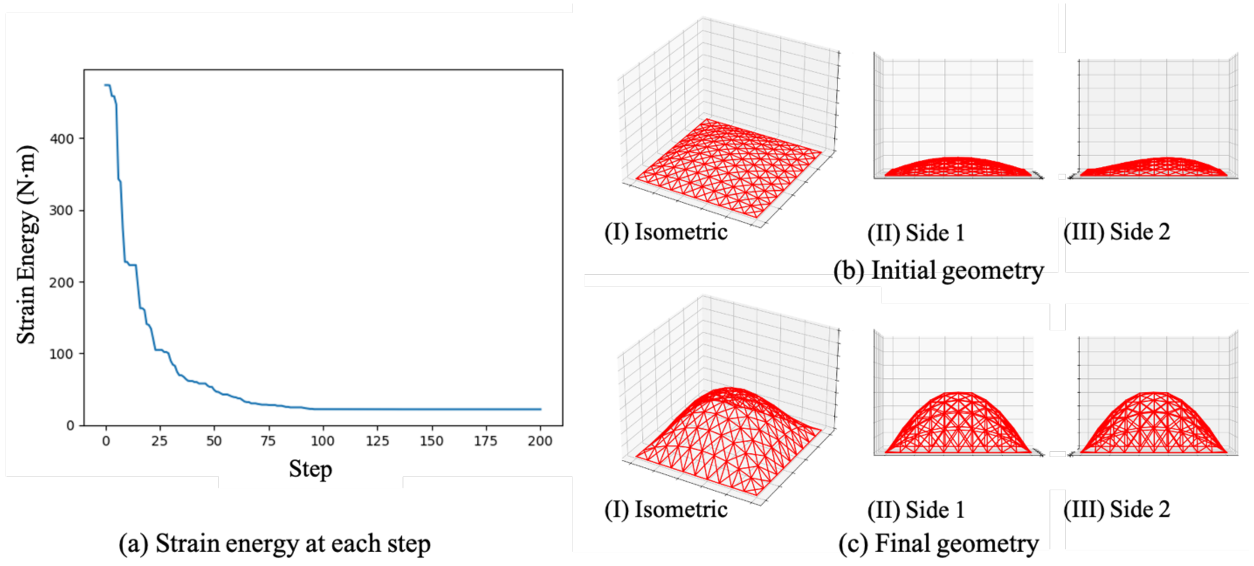


Figure 5.15: 10×10-grid lattice shell with initial irregular geometry for geometry optimization of lattice shells – Structural model 1

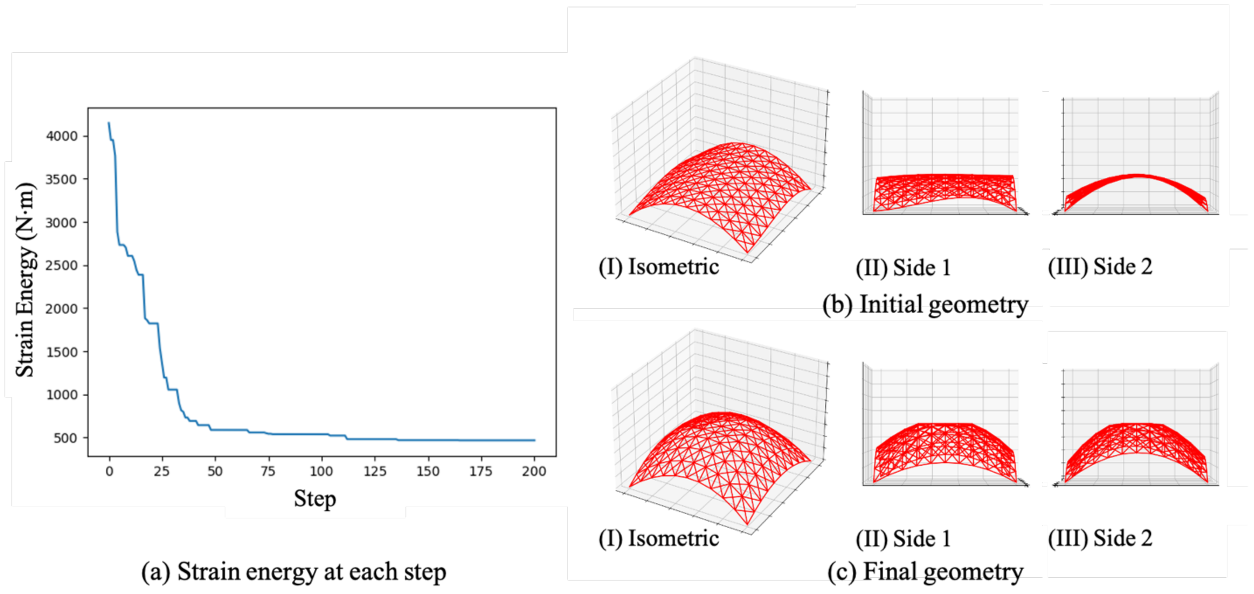


Figure 5.16: 10×10-grid lattice shell with initial irregular geometry for geometry optimization of lattice shells – Structural model 2

5.4 Conclusion

In this chapter, a novel approach to geometry optimization of lattice shells using a GAT-DDPG agent is introduced. The objective of this optimization is to generate the Bézier surface of the lattice shell in a way that minimizes the strain energy. The concept of representing both the structural information and the configuration of the Bézier control net as graph data is proposed. These parallel data representations are effectively processed by the RL agent, which makes informed decisions to modify the Bézier surface, finally optimizing the geometry of the lattice shell for improved structural performance. The key findings of this chapter are summarized as follows:

(1) An innovation lies in the introduction of new graph representations for both the lattice shell structure and the Bézier control net. These representations enable the GAT-DDPG agent to comprehensively observe and analyze both the structural configurations and responses together with the configurations of the Bézier control net. This dual-domain representation enriches the agent to understand the intricate interplay between the Bézier control net, the structural geometry, and structural performance.

(2) The RL agent exhibits the capability to directly manipulate the Bézier control net, leveraging specially designed neural network architectures. This capability empowers the agent to guide the Bézier surface towards configurations that minimize strain energy.

(3) The GAT-DDPG agent is able to learn how to solve the geometry optimization problem for lattice shells. It accomplishes this by minimizing the strain energy through a reward function derived from the changes in strain energy resulting from the actions.

(4) The flexibility and transferability of the trained RL agent are shown as it can be trained using small-scale lattice shells with modestly sized Bézier control nets and then applied to lattice shells of different sizes, distinct Bézier surface configurations, and various optimization problem settings, all without the need for retraining. This adaptability further enhances the practical applicability of the RL-based approach.

(5) A significant result in this chapter is the demonstrated superiority of the trained RL agent in obtaining optimal solutions when compared to conventional optimization methods such as SA and PSO. This finding highlights the potential of RL in enhancing the efficiency and

effectiveness of geometry optimization for lattice shells, with promising implications for structural engineering design practices.

In conclusion, this chapter proposed an optimization method of geometry optimization for lattice shells, utilizing the novel integration of the RL method and multi-domain graph representation. In this method, the strain energy, which briefly indicates the structural performance of the lattice shells, is utilized as an objective function. The result also reaffirms the adaptability and transferability of RL agents in engineering applications, offering an optimization method for the problem of geometry optimization. Note that this research considers only vertical loads, which affect the optimal geometry of the lattice shells. However, it is important to consider other loads and failure scenarios in actual lattice shell design. Future research should endeavor the application of the proposed method in these scenarios.

Chapter 6
Multi-objective optimization of truss structure using MARL and graph representation

6.1 Introduction

This chapter introduces a novel approach to multi-objective optimization of trusses, leveraging MARL and graph representation techniques. In the context of structural engineering, multi-objective optimization entails the simultaneous consideration of multiple conflicting objectives, such as minimizing material usage and cost and maximizing structural performance. To address these intricate optimization problems, MADDPG agents are collaboratively trained. These agents work in accordance to obtain Pareto optimal solutions, which represent the trade-offs between competing objectives. Crucially, both structural information and solutions within the objective space are represented as graphs, enabling a more comprehensive and intuitive approach to multi-objective optimization. The MADDPG agents utilize these graph representations to modify the structures aiming to expand the non-dominated solutions. All agents obtained rewards based on the improvement of the non-dominated solutions according to their actions. Similar to RL algorithms in previous chapters, agents are trained in the training phase and applied to different structures in the test phase.

The proposed algorithm is evaluated across three distinct problems, showcasing its versatility and effectiveness. These problems encompass a multi-objective mathematical problem, a multi-objective 10-bar truss problem, and, notably, a multi-objective optimization of truss design. The findings presented in the context of truss design highlight the superiority of the trained MADDPG agents, particularly when applied to truss configurations that were previously untrained. This chapter is structured to provide a holistic view of the MARL-based multi-objective optimization approach. Section 6.1 introduces the optimization problems tackled in this chapter. Section 6.2 provides a detailed explanation of the MARL method, encompassing the algorithmic framework, the graph-based representation of solutions within the objective space, the formulation of reward functions, and general numerical settings. Sections 6.3 to 6.5 comprehensively present the results achieved by the algorithm when applied to the multi-objective mathematical problem, the 10-bar truss problem, and the truss design optimization problem, respectively. Each of these sections provides observation formulations, action interpretations, agent architectures, and the outcomes of the training and testing phases. Finally, Section 6.6 summarizes the key findings and contributions of this research, underlining the potential of MARL to solve multi-objective optimization in structural engineering.

6.2 Formulation of optimization problems

This research utilizes three multi-objective optimization problems to verify the effectiveness of the proposed MARL algorithm. The first and second problems are mathematical and structural problems, respectively. Note that these problems have known Pareto fronts. Therefore, the proposed MARL algorithm can obtain exact solutions. The third problem is a multi-objective optimization problem of truss trade-off design, having the structural volume and geometry deviation from the preference as the objective functions. Note that the Pareto fronts of this problem vary depending on the preference of geometry, resulting in different Pareto fronts.

6.2.1 Multi-objective mathematical problem: Constr-Ex

Consider the Constr-Ex problem [137] as the mathematical problem to be solved. The problem is defined as follows:

$$\begin{aligned} \text{minimize} \quad & f_1(\mathbf{x}) = x_1 \\ & f_2(\mathbf{x}) = (1 + x_2)/x_1 \\ \text{subject to} \quad & 9x_1 + x_2 \geq 6 \\ & 9x_1 - x_2 \geq 1 \end{aligned} \tag{6.1}$$

where $0.1 \leq x_1 \leq 1$, $0 \leq x_2 \leq 5$, and $\mathbf{x} = (x_1, x_2)$.

6.2.2 Multi-objective structural optimization problem: 10-bar truss

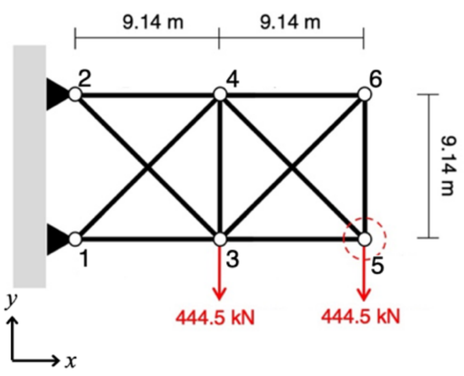
Let $c_i \in \{0,1,\dots,n_\mathrm{c}\}$ and $\mathbf{c}$ denote predetermined indices of the cross-sectional properties and a vector that represents all indices of cross-sectional properties in the structure, respectively. The length of structural element i is l_i. The structural volume W of a truss, having n_e elements and n_s nodes, is obtained from the cross-sectional areas $H(c_i)$ and the lengths l_i as follows:

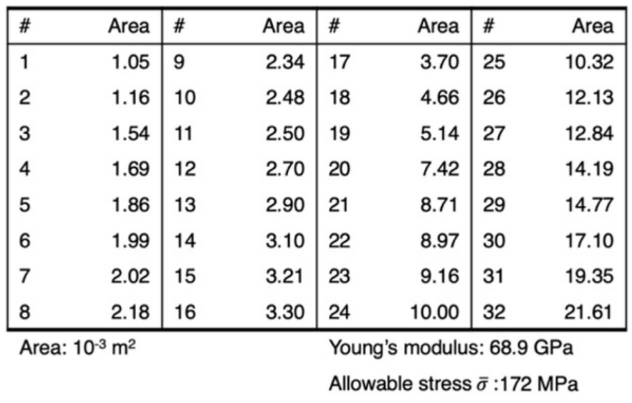
$$W(\mathbf{c}) = \sum_{i=1}^{n_\mathrm{e}} H(c_i) l_i \tag{6.2}$$

The problem of a 10-bar truss is shown in Figure 6.1, and aims to minimize the structural volume and the displacement of node 5 under stress constraint. This multi-objective optimization problem is formulated as follows:

$$\begin{aligned} \text{minimize} \quad & W(\mathbf{c}) \\ & \delta_5(\mathbf{c}) \\ \text{subject to} \quad & \sigma_{\max} \leq \bar{\sigma} \end{aligned} \tag{6.3}$$

where $\sigma_{\max}$, $\bar{\sigma}$, and δ_5 are the maximum absolute value of stress, the upper bound stress, and the absolute value of y-directional displacement of node 5, respectively.





#	Area	#	Area	#	Area	#	Area
1	1.05	9	2.34	17	3.70	25	10.32
2	1.16	10	2.48	18	4.66	26	12.13
3	1.54	11	2.50	19	5.14	27	12.84
4	1.69	12	2.70	20	7.42	28	14.19
5	1.86	13	2.90	21	8.71	29	14.77
6	1.99	14	3.10	22	8.97	30	17.10
7	2.02	15	3.21	23	9.16	31	19.35
8	2.18	16	3.30	24	10.00	32	21.61

Area: 10^{-3} m^2

Young's modulus: 68.9 GPa

Allowable stress $\bar{\sigma}$:172 MPa

Figure 6.1: Multi-objective 10-bar truss optimization problem

6.2.3 Multi-objective optimization of truss with shape preferences: trade-off design

This optimization problem aims to minimize the shape deviation from the predetermined geometry, denoted as *target geometry*, and structural volume. Let the height and target height of node i be y_i and $\hat{y}_i$, respectively. The shape deviation is measured as a linear difference D computed as

$$D(\mathbf{y}) = \sum_{i=1}^{n_\mathrm{s}} |y_i - \hat{y}_i| \tag{6.4}$$

where $\mathbf{y}$ and n_s are the vector of nodal heights and the number of nodes in the truss, respectively.

Another objective function to be minimized is the structural volume. Let $c_j \in \{0,1,\dots,n_\mathrm{c}\}$ and L_j be the indices of cross-sectional properties and lengths of truss elements j, respectively. Structural volume W of a truss is obtained from cross-sectional areas $H(c_j)$ when specifying the cross-sectional indices c_j, and lengths. Note that, in this problem, W is also a function of nodal coordinate vector $\mathbf{y}$ because it determines the lengths of elements. Let the number of truss elements be n_e. W is computed as

$$W(\mathbf{c},\mathbf{y}) = \sum_{j=1}^{n_\mathrm{e}} H(c_j) \cdot L_j \tag{6.5}$$

Let $\sigma_{\max}$, $\delta_{\max}$, $\bar{\sigma}$, and $\bar{\delta}$ be the maximum absolute value of stress, the maximum absolute value of displacement, the upper limit stress, and the upper limit displacement, respectively. The multi-objective optimization problem of truss to minimize the structural volume and the shape deviation is formulated as

$$\begin{aligned} \text{minimize} \quad & W(\mathbf{c},\mathbf{y}) \\ & D(\mathbf{y}) \\ \text{subject to} \quad & \sigma_{\max} \le \bar{\sigma} \\ & \delta_{\max} \le \bar{\delta} \end{aligned} \tag{6.6}$$

6.3 Multi-objective optimization using MARL

This section explains a general method for solving multi-objective optimization problems using MARL. In MARL, the agents, observations, actions, and rewards must be specified. However, the structure of MARL depends on types of multi-objective optimization problems consisting of variables, that are observed and modified by the agents. Therefore, only the parts of the proposed method, that can be generalized are explained in this section. These parts include the algorithm, observation of the objective space, and the reward, which are explained in Sections 6.2.1, 6.2.2, and 6.2.3, respectively.

6.3.1 Algorithm

Suppose objective function whose values are normalized to be in the range of [0,1]. At the beginning of the optimization algorithm, a feasible initial solution is initialized. The objective functions of this initial solution are computed, normalized, and mapped to the objective space. Let the *solution pool* denote all the solutions at an optimization step. Hence, the initial solution has only one solution in the *solution pool* with one solution.

At each optimization step, each solution in the solution pool is selected to be modified (Figure 6.2(a)). The selected solution and current non-dominated solutions are represented as observation data for the MADDPG agents (Figure 6.2(b)). Each agent observes this data and modifies the design variables to create a neighborhood solution (Figure 6.2(c)). Created neighborhood solutions are evaluated for their objective functions and constraints, and the feasible neighborhood solutions are mapped into the objective space (Figure 6.2(d)) and kept for computing the new solution pool. Each agent obtains a reward based on the improvements in the solution pool (Figure 6.2(e)). The observation, action, and reward of all agents are stored in the replay buffer (Figure 6.2(f)), which will be used for training the agents (Figure 6.2(g)). Note that this research utilizes three differently parameterized MADDPG agents. Since all agents have diverse initialized parameters and receive different rewards, they learn to make three neighborhood solutions from each solution to generate diverse solutions.

After iterating through all solutions in the solution pool, a new solution pool is derived from the non-dominated solutions of the current solution pool and the created neighborhood solutions. To avoid a large computational cost, the maximum number of non-dominated solutions in the new solution pool in each step is set as $n_{\max}$. If the number exceeds this value, outer-edge solutions, which are non-dominated solutions that lie at both rims of the front, are selected first, and the other solutions are randomly chosen from the new solution pool (Figure 6.2(h)). Figure 6.3 illustrates the flowchart of the optimization algorithm.

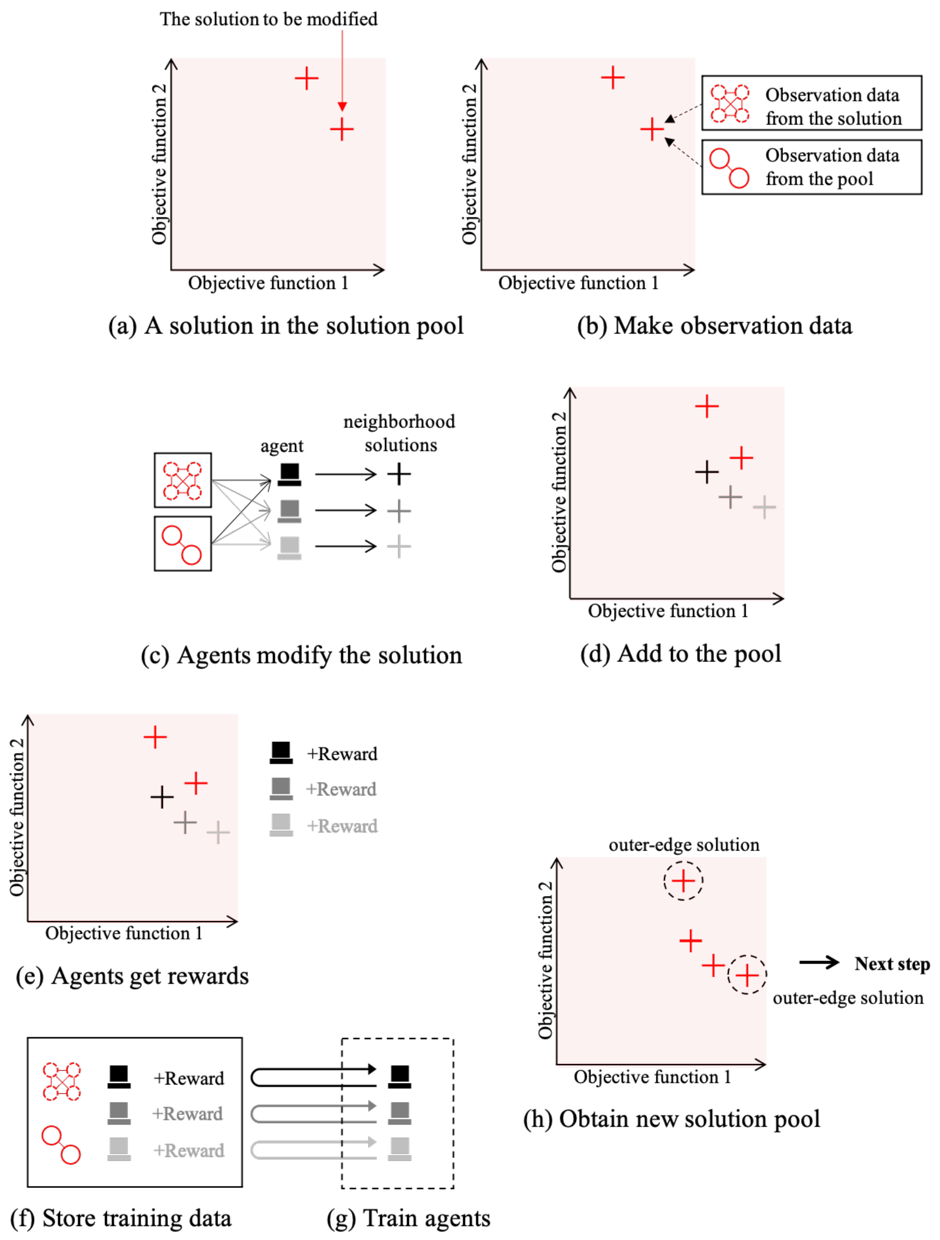


Figure 6.2: The proposed MARL algorithm in each optimization step for multi-objective optimization

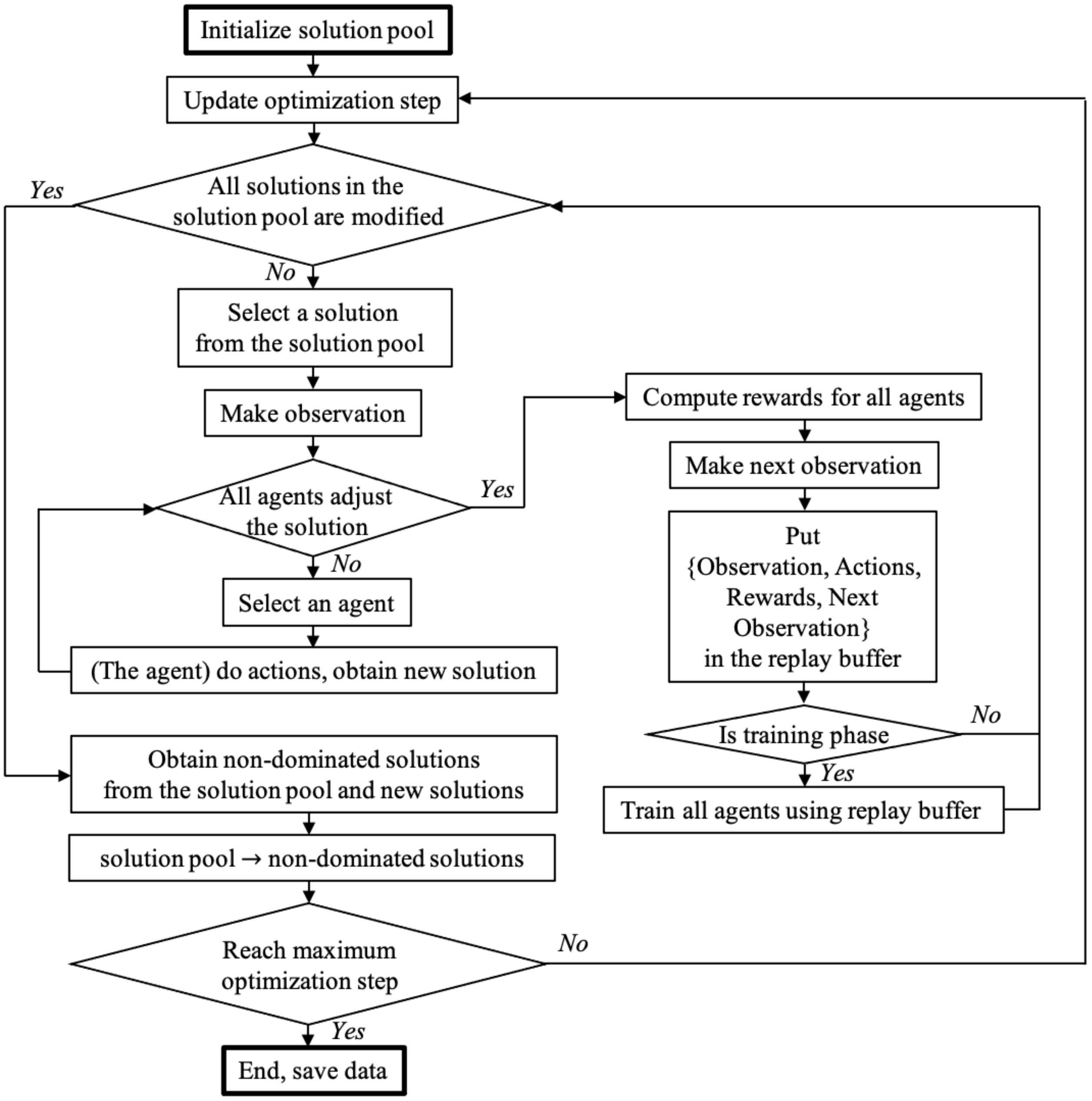
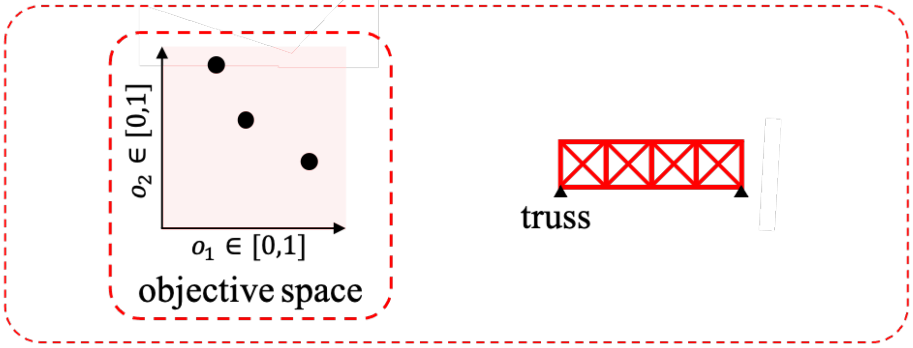

Initialize solution pool
Update optimization step
All solutions in the solution pool are modified
Yes
No
Select a solution from the solution pool
Make observation
All agents adjust the solution
Yes
No
Select an agent
(The agent) do actions, obtain new solution
Compute rewards for all agents
Make next observation
Put {Observation, Actions, Rewards, Next Observation} in the replay buffer
Is training phase
No
Yes
Train all agents using replay buffer
Obtain non-dominated solutions from the solution pool and new solutions
solution pool → non-dominated solutions
Reach maximum optimization step
No
Yes
End, save data

Figure 6.3: Flowchart of the algorithm for multi-objective optimization

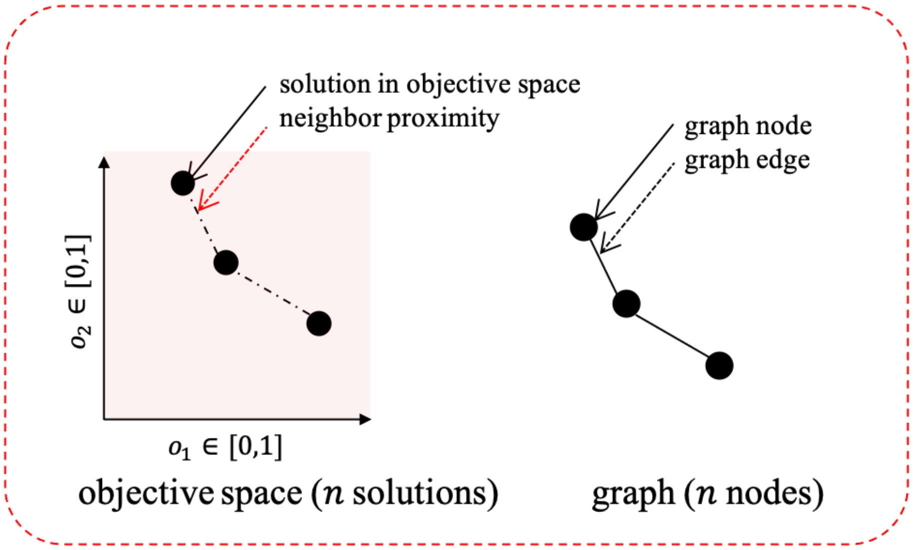
6.3.2 Graph representations

This research proposes a method that lets the agents observe the solution pool in the objective space through the graph representation. In each optimization step, each non-dominated solution in the objective space (Figure 6.4(a)) is represented as graph nodes, as illustrated in Figure 6.4(b).

The boundary of the region for computing current hypervolume, obtained from the non-dominated solutions, consists of lines connecting neighboring non-dominated solutions. These lines are the edges of the graph of the solution pool (Figure 6.4(b)). The graph of the solution pool is represented utilizing a node feature matrix of the solution pool $\mathbf{N}_\mathrm{p}$ and an adjacency matrix of the solution pool $\mathbf{M}_\mathrm{p}$, as shown in Figure 6.4(c). This graph of the solution pool is updated in each optimization step according to the non-dominated solutions in the solution pool. Tables 6.1 and 6.2 show entries in $\mathbf{N}_\mathrm{p}$ and $\mathbf{M}_\mathrm{p}$, respectively. Note that this graph representation can be utilized in all multi-objective optimization problems throughout this chapter.



(a)



(b)

$$\mathbf{N}_{\mathrm{p}} = \overbrace{\begin{bmatrix} n_{\mathrm{p}_{1,1}} & \cdots & n_{\mathrm{p}_{1,f}} \\ \vdots & \ddots & \vdots \\ n_{\mathrm{p}_{n,1}} & \cdots & n_{\mathrm{p}_{n,f}} \end{bmatrix}}^{f \text{ features}} \left.\vphantom{\begin{bmatrix} a \\ a \\ a \end{bmatrix}}\right\} n \text{ nodes}$$

Node feature matrix

$$\mathbf{M}_{\mathrm{p}} = \overbrace{\begin{bmatrix} 0 & \color{red}{1} & 0 & 0 \\ \color{red}{1} & 0 & \cdots & \vdots \\ 0 & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}}^{n \text{ nodes}} \left.\vphantom{\begin{bmatrix} a \\ a \\ a \\ a \end{bmatrix}}\right\} n \text{ nodes}$$

Adjacency matrix

(c)

Figure 6.4: Graph representation of the non-dominated solutions for multi-objective optimization

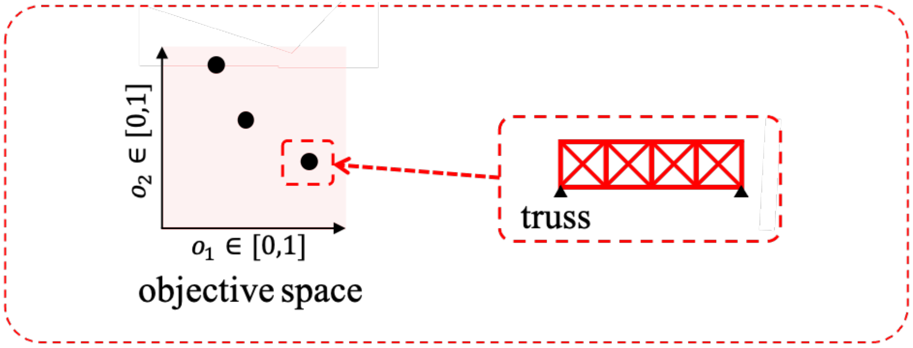
Table 6.1: Values in node feature matrix of the non-dominated solutions $\mathbf{N}_\mathrm{p}$ for multi-objective optimization

$n_{\mathrm{p}_{i,1}}$	o_1: value of objective function 1 of the solution i
$n_{\mathrm{p}_{i,2}}$	o_2: value of objective function 2 of the solution i
$n_{\mathrm{p}_{i,3}}$	current number of the non-dominated solutions/ $n_{\max}$
$n_{\mathrm{p}_{i,4}}$	1 if agents are adjusting this solution; else 0

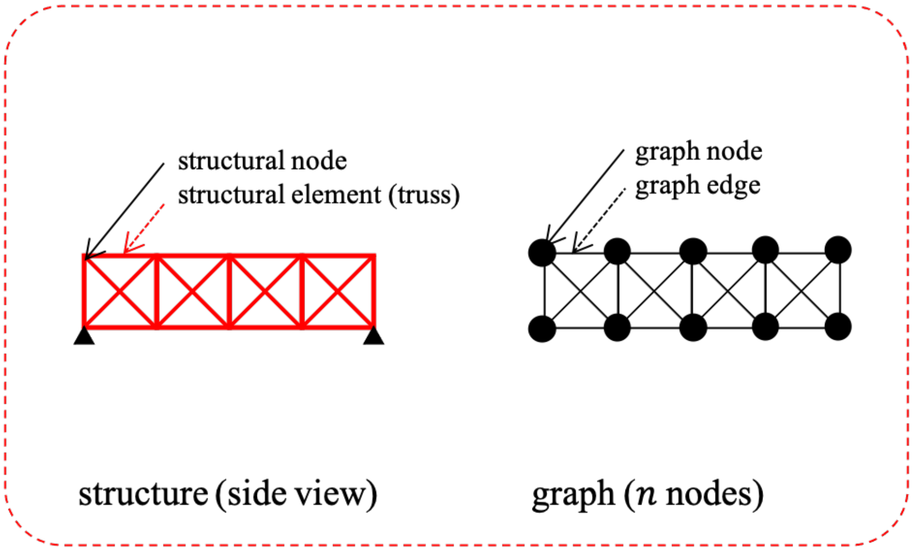
Table 6.2: Values in adjacency matrix of the non-dominated solutions $\mathbf{M}_\mathrm{p}$ for multi-objective optimization

$m_{\mathrm{p}_{i,j}}$	1 if $j = i + 1$ or $i = j + 1$; else 0

In the multi-objective optimization problems of 10-bar truss and trade-off design, the structure (Figure 6.5(a)) is represented as a graph where structural nodes and elements are graph nodes and edges (Figure 6.5(b)), respectively. The information of structural nodes and the existence of the structural element between the nodes are represented using the node feature matrix of truss **N** and the adjacency matrix of truss **M**, respectively. The sectional area and internal forces in the truss elements are represented utilizing weighted adjacency matrices **P** (Figure 6.5(c)).



(a)



(b)

$$\mathbf{N} = \left.\overbrace{\begin{bmatrix} n_{1,1} & \cdots & n_{1,f} \\ \vdots & \ddots & \vdots \\ n_{n,1} & \cdots & n_{n,f} \end{bmatrix}}^{f\ \text{features}}\right\} n\ \text{nodes} \qquad \textbf{Node feature matrix}$$

$$\mathbf{M} = \left.\overbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & \cdots & \vdots \\ 0 & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}}^{n\ \text{nodes}}\right\} n\ \text{nodes} \qquad \textbf{Adjacency matrix}$$

$$\mathbf{P} = \left.\overbrace{\begin{bmatrix} 0 & p_{1,2} & 0 & 0 \\ p_{2,1} & 0 & \cdots & \vdots \\ 0 & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}}^{n\ \text{nodes}}\right\} n\ \text{nodes} \qquad \textbf{Weighted adjacency matrix}$$

(c)

Figure 6.5: Graph representation of the truss solutions for multi-objective optimization

6.3.3 Reward

The MARL agents, in the proposed method, are trained to work together to improve the non-dominated solutions in the objective space. Unlike single agent RL, it is difficult to assign rewards based on the improvement of the solutions in MARL because each agent contributes to the improvement of the solutions differently. In order to address this contribution by each agent, this chapter utilizes the *difference reward* method [138] that measures the contribution of each agent. The effectiveness of this method in the applications in MARL is shown by Refs. [139-141].

Let $\mathcal{D}_{t+1}^{u}(\mathrm{S}_t, \mathrm{A}_t^u)$ be the difference reward agent u obtained when it does action A_t^u in a state S_t. $\mathcal{G}(\mathrm{S}_t, \mathcal{A}_t)$ and $\mathcal{G}(\mathrm{S}_t, \mathcal{A}_t - \mathrm{A}_t^u)$ are the utility indicators, considering actions from all agents in S_t and actions from all agents except A_t^u in S_t, respectively. The difference reward method states that $\mathcal{D}_{t+1}^{u}(\mathrm{S}_t, \mathrm{A}_t^u)$ is computed by removing the contribution of agent u from the system defined as follows:

$$\mathcal{D}_{t+1}^{u}(\mathrm{S}_t, \mathrm{A}_t^u) = \mathcal{G}(\mathrm{S}_t, \mathcal{A}_t) - \mathcal{G}(\mathrm{S}_t, \mathcal{A}_t - \mathrm{A}_t^u) \tag{6.7}$$

Note that once the agents obtain non-dominated solutions that are close to the Pareto front, the improvement of the obtained non-dominated solutions becomes small. Hence, the hypervolume, distribution, and spread of the current non-dominated solutions should be integrated into the reward. This research proposes a reward function considering all these indicators that can guide the MARL agents to keep improving the non-dominated solutions. To manipulate different agents to improve the same solution differently, agents also obtain rewards based on the improvement of the objective functions by assigning different weights for the reward, associated with objective functions, to different agents. Let $u \in \{u_1, u_2, u_3\}$ be agents in the MARL system and $\boldsymbol{r}^u = (r_1, r_2, r_3, r_4, r_5)$ be a vector of all reward components. All agents receive their reward components after all agents modify a solution **x** in the solution pool. Table 6.3 explains all reward components of agent u in detail. Note again that the difference reward method is utilized only for obtaining r_1 which is associated with the improvement of the size of the hypervolume.

Table 6.3: Reward components of agent u for multi-objective optimization

r_1	Improvement of the hypervolume contributed by the action of agent $u \in \{u_1, u_2, u_3\}$ computed as follows: $\Delta\mathrm{HV}_{ref} - \Delta\mathrm{HV}_{-A_t^u\mid ref}$ where $\Delta\mathrm{HV}_{ref}$ and $\Delta\mathrm{HV}_{-A_t^u\mid ref}$ are the improvement of the hypervolume considering actions from all agents and those that consider actions from all agents except agent u, respectively. ref denotes the reference point for computing the hypervolume, having the coordinate (1,1) in the objective space.
r_2	Improvement of the objective functions by the action of agent u computed as follows: $w^u\left[\max\left(0, o_1^{\mathbf{x}} - o_1^{\mathbf{x}_u}\right)\right] + (1 - w^u)\left[\max\left(0, o_2^{\mathbf{x}} - o_2^{\mathbf{x}_u}\right)\right]$ where $o_1^{\mathbf{x}}$, $o_2^{\mathbf{x}}$, $o_1^{\mathbf{x}_u}$, and $o_2^{\mathbf{x}_u}$ are the first and second objective functions before and after adjustment by the agent u, respectively. Note that these values of objective functions are normalized by dividing by the maximum possible values of the objective functions and are 0 if constraints are not satisfied. w^u is a weight assigned to the agent u.
r_3	Hypervolume of the current non-dominated solutions.
r_4	Distribution of the current non-dominated solutions measured by the *standard deviation of crowd distance of the non-dominated solutions* [137]
r_5	Spread (i.e., length) of the current non-dominated solutions measured using the *accumulated Euclidean distances of the non-dominated solutions* [142]

It is important to note that each reward component also depends on the stage of the optimization. During the early optimization stage when the hypervolume is small, the agents should focus on improving r_1 and r_2. However, after obtaining a relatively large hypervolume, the agents should maintain that large hypervolume and acquire a high r_3 value. The value r_5, representing the spread, should also be relative to the hypervolume and have the largest possible value as $2\sqrt{\mathrm{HV}}$ for a bi-objective problem. From these assumptions, this research also proposes a weight vector $\boldsymbol{\phi}$ for the reward components which enlarges r_1 and r_2 when the size of the hypervolume is small, enlarges r_3 when the hypervolume is large, and normalizes r_5 using its largest values. The proposed weight vector $\boldsymbol{\phi}$ is shown in Eq. (6.8), where constants in this equation are obtained from trial and error, to enhance the learning ability of the agents to obtain larger rewards and hypervolume.

$$\boldsymbol{\phi} = \left(\frac{0.25}{\max(0.25, \mathrm{HV})}, \frac{0.25}{\max(0.25, \mathrm{HV})}, 10, -0.05, \frac{0.05}{2 \times \sqrt{\max(0.25, \mathrm{HV})}}\right) \tag{6.8}$$

Let n_{d} be the number of solutions in the solution pool of the current optimization step. By utilizing all reward components and the weights, R_{t+1}^{u} for agent u is computed as follows:

$$\mathrm{R}_{t+1}^{u} = \boldsymbol{r}^{u}\boldsymbol{\phi}^{\mathrm{T}}/n_{\mathrm{d}} \tag{6.9}$$

6.3.4 Numerical settings

This research utilizes three multi-objective optimization problems to validate the effectiveness of the proposed MARL algorithm. In each problems, three different agents denoted as $u \in \{u_1, u_2, u_3\}$ are trained together, utilizing w^{u_1}, w^{u_2}, and w^{u_3} of $\boldsymbol{r}^{u}$ as 1, 0.5, and 0, respectively. Numerical examples of a multi-objective mathematical problem, and a multi-objective 10-bar truss problem are performed on a PC with a CPU of Intel Core i5-6600 (3.3 GHz, four cores) and a GPU of AMD Radeon R9 M395 2 GB. An example of multi-objective optimization of truss design is presented using a PC with a CPU of Intel Core i9-11900 (2.5 GHz, 8 cores) and a GPU of Nvidia GeForce RTX3060 12GB. All programs are implemented in the Python 3.6 environment.

6.4 Multi-objective mathematical problem using MARL and graph representation

6.4.1 Observation

In the Constr-Ex problem in Eq. (6.1), the agents observe the solution pool and the $\mathbf{x}$ values of the solution to be modified. Table 6.4 shows the observation data of the solution and the solution pool for the agents in this problem. In Table 6.5, row i in the node feature matrix of the non-dominated solutions contains entries utilizing a proposed graph representation in Section 6.2.2, similar to Table 6.1. These entries are the position of the solution i, denoted as $\mathbf{x}_i$, in the objective space and the current objective space, the current number of the non-dominated solutions, and a value to indicate that the solution is being modified, respectively. Note again that this research normalizes the values of the objective functions. Hence, the value of $f_2(\mathbf{x}_i)$ is divided by 10 in Table 6.5. The observation data of $\mathbf{M}_{\mathrm{p}}$ is similar to Table 6.2 explained in Section 6.2.2.

Table 6.4: Values in the normalized feature vector of $\mathbf{x}$ ($\mathbf{n}$) for multi-objective mathematical optimization problem

n_1	x_1
n_2	$x_2/5$

Table 6.5: Values in row i of the node feature matrix of the non-dominated solutions $\mathbf{N}_\mathrm{p}$ for multi-objective mathematical optimization problem

$n_{\mathrm{p}_{i,1}}$	$f_1(\mathbf{x}_i)$
$n_{\mathrm{p}_{i,2}}$	$f_2(\mathbf{x}_i)/10$
$n_{\mathrm{p}_{i,3}}$	current number of the non-dominated solutions/ $n_{\max}$
$n_{\mathrm{p}_{i,4}}$	1 if agents are adjusting this solution; else 0

6.4.2 Action

In this multi-objective optimization problem, each agent u modifies the values of x_1 and x_2 utilizing its output $\boldsymbol{\pi}^u_{i\in\{1,2\}}$. The outputs are obtained from the hyperbolic tangent activation function $\tanh(z) = (e^z - e^{-z})/(e^z + e^z)$ where $-1 \leq \tanh(z) \leq 1$ is satisfied. The modification of x_1 and x_2 using $\boldsymbol{\pi}^u_{i\in\{1,2\}}$ is defined as follows:

$$\begin{aligned} x_1 &\leftarrow \max(0.1, \min(1, x_1 + 0.2\boldsymbol{\pi}^u_1)) \\ x_2 &\leftarrow \max(0, \min(5, x_2 + 0.2\boldsymbol{\pi}^u_2)) \end{aligned} \tag{6.10}$$

6.4.3 Agents

Let $\{u_1, u_2, u_3\}$ be a set of MARL agents that consists of similar policy and value functions that are made of MLPs and GCNs. The policy function of agent u_1 takes $\mathbf{n}$, $\mathbf{N}_\mathrm{p}$, and $\widetilde{\mathbf{M}}_\mathrm{p}$ as inputs to compute output $\boldsymbol{\pi}^{u_1}$, defined as

$$\boldsymbol{\pi}^{u_1} = \boldsymbol{\pi}^{u_1}\left(\mathbf{n}, \mathbf{N}_\mathrm{p}, \widetilde{\mathbf{M}}_\mathrm{p}\right) \tag{6.11}$$

The value function of agent u_1 computes the expected reward (i.e., the Q-value), which is denoted as Q^{u_1}, by taking $\mathbf{n}$, $\mathbf{N}_\mathrm{p}$, $\widetilde{\mathbf{M}}_\mathrm{p}$, $\boldsymbol{\pi}^{u_1}$ and outputs of all other policy functions $\boldsymbol{\pi}^{u_2}$ and $\boldsymbol{\pi}^{u_3}$ as inputs as

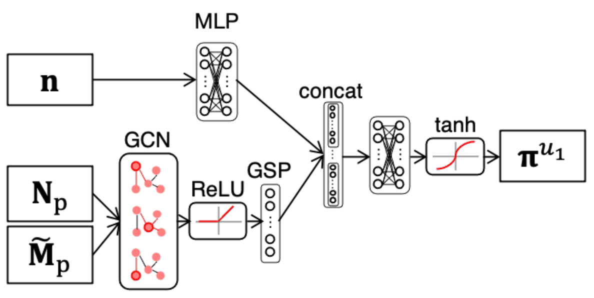
$$\mathrm{Q}^{u_1} = \mathrm{Q}^{u_1}\left(\mathbf{n}, \mathbf{N}_\mathrm{p}, \widetilde{\mathbf{M}}_\mathrm{p}, \boldsymbol{\pi}^{u_1}, \boldsymbol{\pi}^{u_2}, \boldsymbol{\pi}^{u_3}\right) \tag{6.12}$$

In this research, every agent takes the same observation data as input. Therefore, a simple neural network architecture of the value function can be made compared to the original MADDPG which takes different observations in each agent [86].

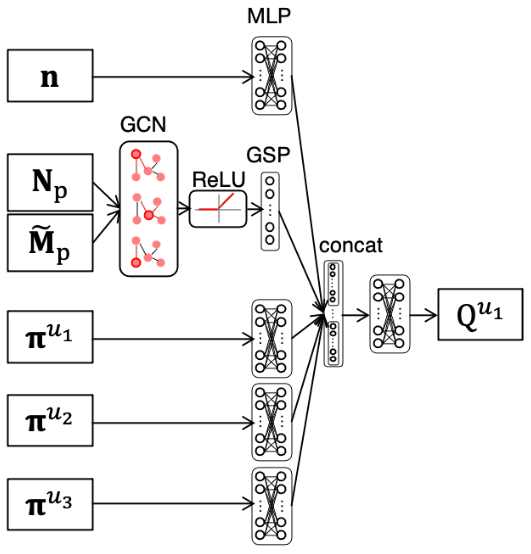
Table 6.6 summarizes the computation processes of the policy and value functions of the GCN-MADDPG u_1 agent for the Constr-Ex problem. The neural network architecture of the policy and value functions (i.e., actor and critic networks) of each agent are shown in Figures 6.6(a) and 6.6(b), respectively. Inputs, shown on the left, are sent into the actor and critic networks, which compute $\boldsymbol{\pi}^{u_1}$ and Q^{u_1}, respectively, as shown on the right side of the networks. In these neural networks, the GCNs with ReLU activation function returns $\mathbf{N}' \in \mathbb{R}^{n \times h}$ with h =100 (i.e., the number of feature maps, explained in Section 2.5 in Chapter 2), and the MLPs have 2 layers with 100 neurons in each layer. Note that all equations and symbols in the computation processes are explained in Section 2.6 and Figure 2.6 in Chapter 2.

Table 6.6: Policy and value functions of GCN-MADDPG (u_1 agent) for multi-objective mathematical optimization problem

policy function $\boldsymbol{\pi}^{u_1}$	value function Q^{u_1}
input: $\mathbf{n}, \mathbf{N}_\mathrm{p}, \widetilde{\mathbf{M}}_\mathrm{p}$	input: $\mathbf{n}, \mathbf{N}_\mathrm{p}, \widetilde{\mathbf{M}}_\mathrm{p}, \boldsymbol{\pi}^{u_1}, \boldsymbol{\pi}^{u_2}, \boldsymbol{\pi}^{u_3}$
computation: step 1: $\mathbf{n}_{1.1} = \mu(\mathrm{f}_\mathrm{NN}(\mathbf{n}))$ step 2: $\mathbf{N}_{1.2} = \mu(\mathbf{N}_\mathrm{p}, \widetilde{\mathbf{M}}_\mathrm{p})$ $\mathbf{n}_{1.3} = \mathrm{GSP}(\mathbf{N}_{1.2})$ step 3: $\mathbf{n}_2 = [\mathbf{n}_{1.1} \parallel \mathbf{n}_{1.3}]$ step 4: $\boldsymbol{\pi}^{u_1} = \tanh(\mathrm{f}_\mathrm{NN}(\mathbf{n}_2))$	computation: step 1: $\mathbf{n}_{1.1} = \mu(\mathrm{f}_\mathrm{NN}(\mathbf{n}))$ step 2: $\mathbf{N}_{1.2} = \mu(\mathbf{N}_\mathrm{p}, \widetilde{\mathbf{M}}_\mathrm{p})$ $\mathbf{n}_{1.3} = \mathrm{GSP}(\mathbf{N}_{1.2})$ step 3: $\mathbf{n}_{1.4} = \mu(\mathrm{f}_\mathrm{NN}(\boldsymbol{\pi}^{u_1}))$ step 4: $\mathbf{n}_{1.5} = \mu(\mathrm{f}_\mathrm{NN}(\boldsymbol{\pi}^{u_2}))$ step 5: $\mathbf{n}_{1.6} = \mu(\mathrm{f}_\mathrm{NN}(\boldsymbol{\pi}^{u_3}))$ step 6: $\mathbf{n}_2 = [\mathbf{n}_{1.1} \parallel \mathbf{n}_{1.3} \parallel \mathbf{n}_{1.4} \parallel \mathbf{n}_{1.5} \parallel \mathbf{n}_{1.6}]$ step 7: $\mathrm{Q}^{u_1} = \mathrm{f}_\mathrm{NN}(\mathbf{n}_2)$
output: $\boldsymbol{\pi}^{u_1} \in \mathbb{R}^{2\times 1}$	output: $\mathrm{Q}^{u_1} \in \mathbb{R}^{1\times 1}$



(a) Actor network



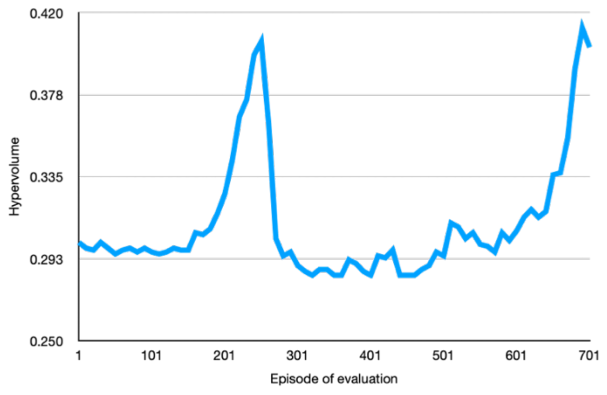
(b) Critic network

Figure 6.6: Actor and critic networks for multi-objective mathematical optimization problem

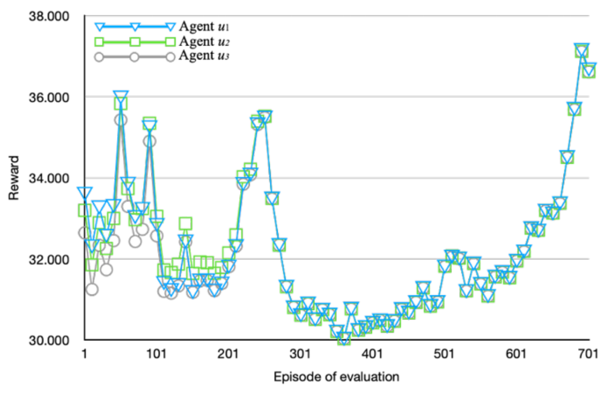
6.4.4 Training phase

Agents are trained for 701 episodes and each episode terminates after 10 steps to obtain non-dominated solutions to the problem in Eq. (6.1) using $n_\mathrm{max} = 20$ to prevent the algorithm from creating too many solutions. The initial solution is $\mathbf{x} = (0.75, 0.25)$ and the values in $\mathbf{x}$ are rounded to the second decimal. During the training, two *edge solutions*, which might not be the non-dominated solutions but lie at the rim of the objective space, are added to the solution pool to improve the exploration of the agents. This research evaluates the improvements of the agent performances using the same function every 10 training episodes. The agent surrogate functions, as explained in Section 2.3.3 in Chapter 2, are adjusted using the Adam optimizer, as explained in Section 2.5.3. The mini-batch size is set to 32 and the learning rates are 10^{-6} and 10^{-5} for policy and value functions, respectively. Note that the details of the training process are explained in Section 2.5.2. Weights and biases of the surrogate functions are synchronized with those of the online functions every 300 steps using τ=0.05, as explained in Section 2.3.3.

Figure 6.7 shows the variation of the final hypervolume (a) and the obtained rewards (b) in each evaluation episode, represented as the vertical axis, and the evaluation episode represented as the horizontal axis. It is seen from Figure 6.7(a), MADDPG agents cannot obtain large hypervolume in the early 200 episodes. Their ability to obtain a large hypervolume is shown around the first 250 episodes. However, their performances drops but increases afterward. This spike in performance is also visible in the rewards obtained by each agent in Figure 6.7(b). Note that the spikes could happen during the training because the trainable parameters of agents were modified. After 700 episodes, MADDPG agents obtains larger hypervolume sizes and higher rewards than those around 250th episode.



(a) Hypervolume

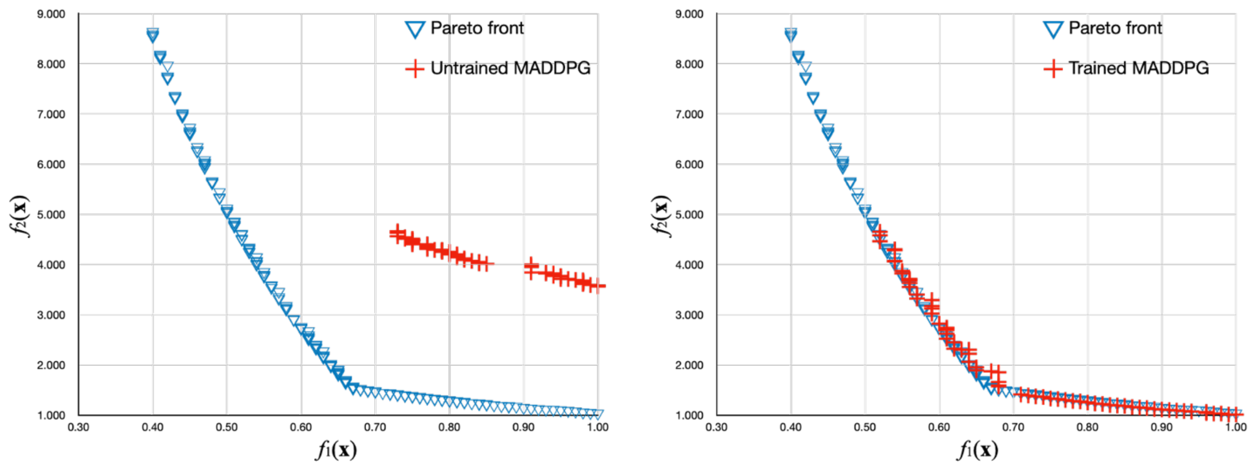


(b) Reward

Figure 6.7: Variation of hypervolume and reward in training phase for multi-objective mathematical optimization problem; (a) Hypervolume, (b) Reward

6.4.5 Test phase

The trained MADDPG agents with the weights saved at the 700th episode in the training phase are utilized for optimizing the Constr-Ex problem ten times in the test phase. In this phase, each episode is terminated after 20 steps, $n_{\max}$ is set to 50, $\mathbf{x} = (0.75, 0.25)$ is the initial solution, and ten edge solutions are added to the solution pool in each optimization step. The obtained Pareto fronts by the untrained and trained MADDPG agents are compared to the true Pareto front to verify the effectiveness of the algorithm, as illustrated in Figure 6.8. From this figure, the trained MADDPG agents using the proposed algorithm can obtain solutions close to the Pareto front except those with large f_2 values. Hence, the effectiveness of the proposed method to obtain the Pareto front is verified. The obtained Pareto fronts by the trained agents are clearly better than those of the untrained agents. Therefore, the effectiveness of the training algorithm is also validated.



(a) Untrained agents (b) Trained agents

Figure 6.8: Solutions obtained by the untrained and trained agents compared to the true Pareto front for multi-objective mathematical optimization problem;
(a) Untrained agents, (b) Trained agents

6.5 Multi-objective 10-bar truss problem using MARL and graph representation

6.5.1 Observation

In this example, the observation data are the solution to be adjusted and the set of non-dominated solutions. These data are the graph representation of the truss $(\mathbf{N}, \widetilde{\mathbf{M}}, \mathbf{P}_\mathrm{s}, \mathbf{P}_\mathrm{t}, \mathbf{P}_\mathrm{c})$ and the graph representations of the solution on the Pareto front $(\mathbf{N}_\mathrm{p}, \widetilde{\mathbf{M}}_\mathrm{p})$ explained in Section 6.2.2. Note that the information of the solution to be adjusted is represented utilizing the node feature matrix of the truss node $\mathbf{N}$, the adjacency matrix of the truss node $\mathbf{M}$, the weighted adjacency matrix of the cross-sectional areas $\mathbf{P}_\mathrm{s}$, the weighted adjacency matrix of the internal tensile stress tension $\mathbf{P}_\mathrm{t}$, and the weighted adjacency matrix of the internal compressive stress $\mathbf{P}_\mathrm{c}$. Tables 6.7 and 6.8 show the entries in row i in the node feature matrix $\mathbf{N}$, and the values in the adjacency and weighted adjacency matrices $(\mathbf{M}, \mathbf{P}_\mathrm{s}, \mathbf{P}_\mathrm{t}, \mathbf{P}_\mathrm{c})$, respectively.

Similar to the previous example, the entries in row i of the node feature matrix of the non-dominated solutions $\mathbf{N}_\mathrm{p}$ are the position of solution i in the objective space and information on current non-dominated solutions, as shown in Table 6.9. In this table, W_i and δ_{5_i} are the structural volume and the absolute y-directional displacement of node 5 of the solution i, respectively where W_max and δ_{5_max} are the maximum possible values of the structural volume and the absolute y-directional displacement of node 5, respectively. $\mathbf{M}_\mathrm{p}$ is formulated as explained in Section 6.2.2.

Table 6.7: Values in row i of the node feature matrix of truss node ($\mathbf{N}$) for multi-objective 10-bar truss optimization problem

$n_{i,1}$	x-coordinate
$n_{i,2}$	y-coordinate
$n_{i,3}$	1 if there is a support in the x-axis of structural node i; else 0
$n_{i,4}$	1 if there is a support in the y-axis of structural node i; else 0
$n_{i,5}$	load in the y-direction
$n_{i,6}$	displacement in the y-direction

Table 6.8: Values in the adjacency and weighted adjacency matrices of truss $(\mathbf{M}, \mathbf{P}_\mathrm{s}, \mathbf{P}_\mathrm{t}, \mathbf{P}_\mathrm{c})$ for multi-objective 10-bar truss optimization problem

$m_{i,j}$	1 if there is an element that connects structural nodes i and j; else 0
$p_{\mathrm{s}_{i,j}}$	H/H_max if there is an element that connects structural nodes i and j; else 0 H and H_max denote the area and maximum area of the structural element
$p_{\mathrm{t}_{i,j}}$	1 if an element that connects structural nodes i and j violates the stress constraint in tension; else $0.5\sigma_\mathrm{t}/\bar{\sigma}$ σ_t denotes the tensile internal stress of the element
$p_{\mathrm{c}_{i,j}}$	1 if an element that connects structural nodes i and j violates the stress constraint in compression; else $0.5\sigma_\mathrm{c}/\bar{\sigma}$ σ_c denotes the compressive internal stress of the element

Table 6.9: Values in row i of the node feature matrix of the non-dominated solutions $(\mathbf{N}_\mathrm{p})$ for multi-objective 10-bar truss optimization problem

$n_{\mathrm{p}_{i,1}}$	W_i/W_max
$n_{\mathrm{p}_{i,2}}$	$\delta_{5_i}/\delta_{5_\mathrm{max}}$
$n_{\mathrm{p}_{i,3}}$	current number of the non-dominated solutions/ n_max
$n_{\mathrm{p}_{i,4}}$	1 if agents are adjusting this solution; else 0

6.5.2 Action

In this problem, each MADDPG agent creates neighborhood solutions by modifying the cross-sectional areas of multiple truss elements simultaneously. Let f be the number of truss elements. In this research, the *connectivity matrix* $\mathbf{C} \in \mathbb{R}^{n_\mathrm{e} \times n_\mathrm{s}}$, indicating elements and their end nodes, is utilized for transforming the structural node feature matrix into the structural element feature matrix. The output of each policy function $\boldsymbol{\pi}^u \in \mathbb{R}^{n_\mathrm{s} \times 3}$ is premultiplied by $\mathbf{C}$ and each cross-sectional size c_i of element i is interpreted from the product of two matrices $\mathbf{C}\boldsymbol{\pi}^u$. This product is $\boldsymbol{\Pi}^u \in \mathbb{R}^{n_\mathrm{e} \times 3}$ and is utilized for determining the modification of the elements as follows:

$$c_i = \begin{cases} \min(32, c_i + 1) & : \text{if } \underset{j}{\operatorname{argmax}}(\boldsymbol{\Pi}^u{}_{i,j\in\{1,2,3\}}) = 1 \\ c_i & : \text{if } \underset{j}{\operatorname{argmax}}(\boldsymbol{\Pi}^u{}_{i,j\in\{1,2,3\}}) = 2 \\ \max(1, c_i - 1) & : \text{if } \underset{j}{\operatorname{argmax}}(\boldsymbol{\Pi}^u{}_{i,j\in\{1,2,3\}}) = 3 \end{cases} \tag{6.13}$$

6.5.3 Agents

Let u_1, u_2, and u_3 be the agents. The policy function of agent u_1 that takes $\mathbf{N}$, $\widetilde{\mathbf{M}}$, $\mathbf{P}_\mathrm{s}$, $\mathbf{P}_\mathrm{t}$, $\mathbf{P}_\mathrm{c}$, $\mathbf{N}_\mathrm{p}$, and $\widetilde{\mathbf{M}}_\mathrm{p}$ as inputs to computed $\boldsymbol{\pi}^u$ is defined as

$$\boldsymbol{\pi}^{u_1} = \boldsymbol{\pi}^{u_1}\left(\mathbf{N}, \widetilde{\mathbf{M}}, \mathbf{P}_\mathrm{s}, \mathbf{P}_\mathrm{t}, \mathbf{P}_\mathrm{c}, \mathbf{N}_\mathrm{p}, \widetilde{\mathbf{M}}_\mathrm{p}\right) \tag{6.14}$$

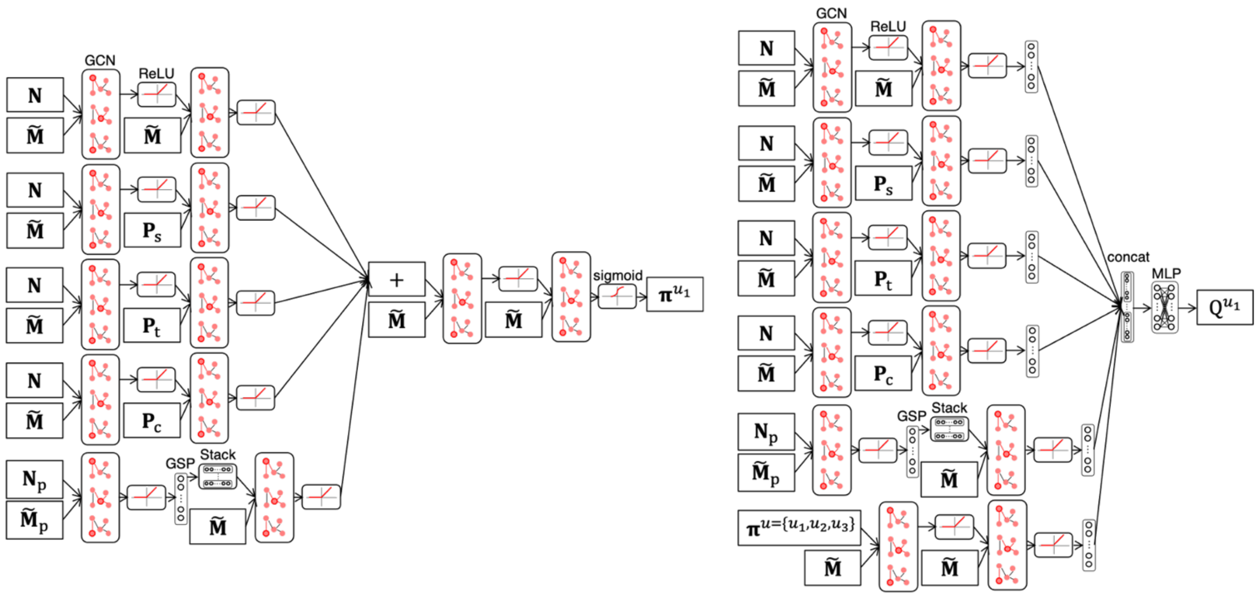
The value function of agent u_1 takes the observation data, outputs of $\boldsymbol{\pi}^{u_1}$, and outputs of all other policy functions to compute Q^{u_1} as

$$\mathrm{Q}^{u_1} = \mathrm{Q}^{u_1}\left(\mathbf{N}, \widetilde{\mathbf{M}}, \mathbf{P}_\mathrm{s}, \mathbf{P}_\mathrm{t}, \mathbf{P}_\mathrm{c}, \mathbf{N}_\mathrm{p}, \widetilde{\mathbf{M}}_\mathrm{p}, \boldsymbol{\pi}^{u_1}, \boldsymbol{\pi}^{u_2}, \boldsymbol{\pi}^{u_3}\right) \tag{6.15}$$

Table 6.10 shows the computation processes of the policy and value functions of the GCN-MADDPG u_1 agent in this 10-bar truss problem. The network architectures of the actor and critic networks in each agent are also shown in Figures 6.9(a) and 6.9(b), respectively. Note that GCNs can take the embedded node signal of the previous GCN layer as input instead of the node feature matrix, and can take a weight adjacency matrix instead of a normalized adjacency matrix as input, in this experiment. This example utilizes GCNs with the ReLU activation function that returns $\mathbf{N}' \in \mathbb{R}^{n \times h}$ with $h = 200$ (i.e., the number of feature maps, explained in Section 2.5 in Chapter 2) and the MLPs that have 2 layers with 200 neurons in each layer. The example also utilizes GSP and stack operations for combining data observed from the specific truss and the set of non-dominated solutions. All equations and symbols in the computation processes are explained in Section 2.6 and Figure 2.6 in Chapter 2.

Table 6.10: Policy and value functions of GCN-MADDPG (u_1 agent) for multi-objective 10-bar truss optimization problem

policy function $\boldsymbol{\pi}^{u_1}$	value function Q^{u_1}
input: $\mathbf{N}, \widetilde{\mathbf{M}}, \mathbf{P}_\mathrm{s}, \mathbf{P}_\mathrm{t}, \mathbf{P}_\mathrm{c}, \mathbf{P}_\mathrm{p}, \widetilde{\mathbf{M}}_\mathrm{p}$	input: $\mathbf{N}, \widetilde{\mathbf{M}}, \mathbf{P}_\mathrm{s}, \mathbf{P}_\mathrm{t}, \mathbf{P}_\mathrm{c}, \mathbf{P}_\mathrm{p}, \widetilde{\mathbf{M}}_\mathrm{p}, \boldsymbol{\pi}^{u_1}, \boldsymbol{\pi}^{u_2}, \boldsymbol{\pi}^{u_3}$
computation: step 1: $\mathbf{N}_{1.1} = \mu(\mu(\mathbf{N}, \widetilde{\mathbf{M}}), \widetilde{\mathbf{M}})$ step 2: $\mathbf{N}_{1.2} = \mu(\mu(\mathbf{N}, \widetilde{\mathbf{M}}), \mathbf{P}_\mathrm{s})$ step 3: $\mathbf{N}_{1.3} = \mu(\mu(\mathbf{N}, \widetilde{\mathbf{M}}), \mathbf{P}_\mathrm{t})$ step 4: $\mathbf{N}_{1.4} = \mu(\mu(\mathbf{N}, \widetilde{\mathbf{M}}), \mathbf{P}_\mathrm{c})$ step 5: $\mathbf{N}_{1.5} = \mu(\mathbf{N}_\mathrm{p}, \widetilde{\mathbf{M}}_\mathrm{p})$ $\mathbf{N}_{1.6} = \mathrm{Stack}^{n}(\mathrm{GSP}(\mathbf{N}_{1.5}))$ $\mathbf{N}_{1.7} = \mu(\mathbf{N}_{1.6}, \widetilde{\mathbf{M}})$ step 6: $\mathbf{N}_{2.1} = \mathbf{N}_{1.1} + \mathbf{N}_{1.2} + \mathbf{N}_{1.3} + \mathbf{N}_{1.4} + \mathbf{N}_{1.7}$ $\mathbf{N}_{2.2} = \mu(\mathbf{N}_{2.1}, \widetilde{\mathbf{M}})$ step 7: $\boldsymbol{\pi}^{u_1} = \sigma(\mathbf{N}_{2.2}, \widetilde{\mathbf{M}})$	computation: step 1: $\mathbf{n}_{1.1} = \mathrm{GSP}\left(\mu(\mu(\mathbf{N}, \widetilde{\mathbf{M}}), \widetilde{\mathbf{M}})\right)$ step 2: $\mathbf{n}_{1.2} = \mathrm{GSP}\left(\mu(\mu(\mathbf{N}, \widetilde{\mathbf{M}}), \mathbf{P}_\mathrm{s})\right)$ step 3: $\mathbf{n}_{1.3} = \mathrm{GSP}\left(\mu(\mu(\mathbf{N}, \widetilde{\mathbf{M}}), \mathbf{P}_\mathrm{t})\right)$ step 4: $\mathbf{n}_{1.4} = \mathrm{GSP}\left(\mu(\mu(\mathbf{N}, \widetilde{\mathbf{M}}), \mathbf{P}_\mathrm{c})\right)$ step 5: $\mathbf{N}_{1.5} = \mu(\mathbf{N}_\mathrm{p}, \widetilde{\mathbf{M}}_\mathrm{p})$ $\mathbf{N}_{1.6} = \mathrm{Stack}^{n}(\mathrm{GSP}(\mathbf{N}_{1.5}))$ $\mathbf{n}_{1.7} = \mathrm{GSP}\left(\mu(\mathbf{N}_{1.6}, \widetilde{\mathbf{M}})\right)$ step 6: $\mathbf{n}_{1.8} = \mathrm{GSP}\left(\mu(\mu(\boldsymbol{\pi}^{u_1}, \widetilde{\mathbf{M}}), \widetilde{\mathbf{M}})\right)$ step 7: $\mathbf{n}_{1.9} = \mathrm{GSP}\left(\mu(\mu(\boldsymbol{\pi}^{u_2}, \widetilde{\mathbf{M}}), \widetilde{\mathbf{M}})\right)$ step 8: $\mathbf{n}_{1.10} = \mathrm{GSP}\left(\mu(\mu(\boldsymbol{\pi}^{u_3}, \widetilde{\mathbf{M}}), \widetilde{\mathbf{M}})\right)$ step 9: $\mathbf{n}_2 = [\mathbf{n}_{1.1} \parallel \mathbf{n}_{1.2} \parallel \mathbf{n}_{1.3} \parallel \mathbf{n}_{1.4} \parallel$ $\mathbf{n}_{1.7} \parallel \mathbf{n}_{1.8} \parallel \mathbf{n}_{1.9} \parallel \mathbf{n}_{1.10}]$ step 10: $\mathrm{Q}^{u_1} = \mathrm{f}_\mathrm{NN}(\mathbf{n}_2)$
output: $\boldsymbol{\pi}^{u_1} \in \mathbb{R}^{n_\mathrm{s} \times 3}$	output: $\mathrm{Q}^{u_1} \in \mathbb{R}^{1 \times 1}$



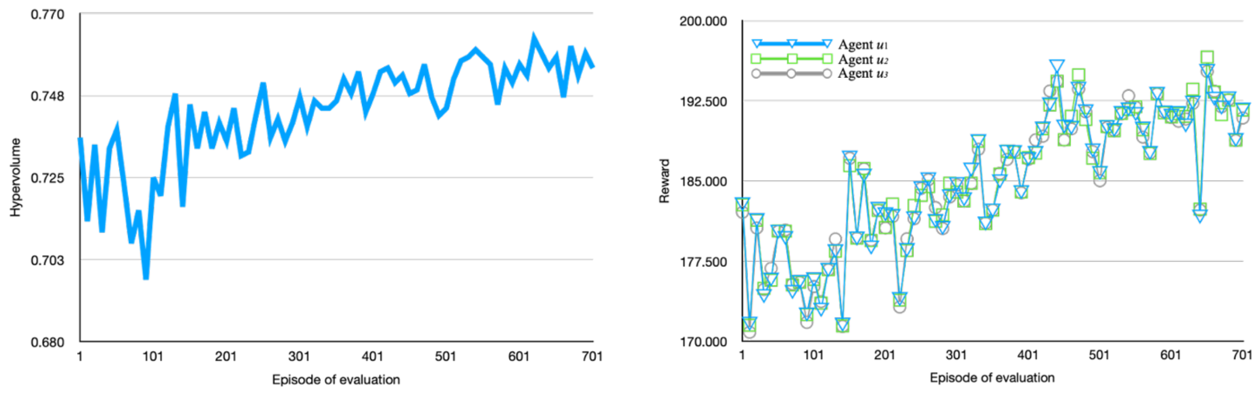
(a) Actor network (b) Critic network

Figure 6.9: Actor and critic networks for multi-objective 10-bar truss optimization problem

6.5.4 Training phase

In this training phase, all elements in the initial structure in Figure 6.1 have cross-sectional indices of #20, indicating the cross-sectional area of 7.42×10^{-3} m^2, $n_{\max}$, as explained in Section 6.3.1, is 20, and two edge solutions are added to the solution pool in each optimization step. The MADDPG agents are trained for 701 episodes and each episode terminates after 50 steps. We evaluate the ability of agents to improve their policies every 10 episodes. Adam optimizer with the mini-batch size set to 32, as explained in Section 2.5.3 in Chapter 2, adjusted the weights of the agent surrogate functions, as explained in Section 2.3.3. The learning rates are 10^{-8} and 10^{-7} for policy and value functions, respectively. Details of the training process are explained in Section 2.5.2. Weights and biases of the surrogate functions are synchronized with those of the online functions every 300 steps using τ=0.005, as explained in Section 2.3.3.

Figure 6.10 illustrates the size of hypervolume (a) and the rewards (b) in the vertical axis, where the horizontal axis represents the episodes that we evaluate. From this figure, MADDPG agents gradually increases the hypervolume during the training (Figure 6.10(a)) and also improves their obtained rewards (Figure 6.10(b)). Note again that the agents are adjusting their trainable parameters in both policy and value functions while considering the policies of other agents as well, during this training phase. Hence, some oscillations of the obtained hypervolume and rewards appear in these figures.



(a) Hypervolume (b) Reward

Figure 6.10: Variation of hypervolume and reward in training phase for multi-objective 10-bar truss optimization problem; (a) Hypervolume, (b) Reward

6.5.5 Test phase

In this test phase, the trained MADDPG agents having the weights saved at episode 700 of the training phase are used for optimizing the truss ten times without further training. In this phase, the initial structure also has cross-sectional indices of #20, each episode is terminated after 200 steps, and ten edge solutions are added to the solution pool in each optimization step. $n_{\max}$ is set to 50 which is larger than those of the training phase because the test phase has fewer trials and a smaller computational cost

The obtained results by the proposed MARL algorithm are compared to those obtained by NSGA-II using a Python library named DEAP. NSGA-II yields a similar Pareto front obtained by Kumar et al. [143] who used the multi-objective modified heat transfer search method. Figure 6.11 illustrates the obtained Pareto front by MADDPG and NSGA-II utilizing 290253 and 300000 (1500 populations, 20 generations, and 10 trials) times of structural analyses, respectively. It is observed from this figure that the proposed MADDPG agents obtained competitive solutions with NSGA-II which is regarded to generate an accurate Pareto front. This result confirms the effectiveness of the proposed MARL algorithm for the multi-objective optimization of truss structures.

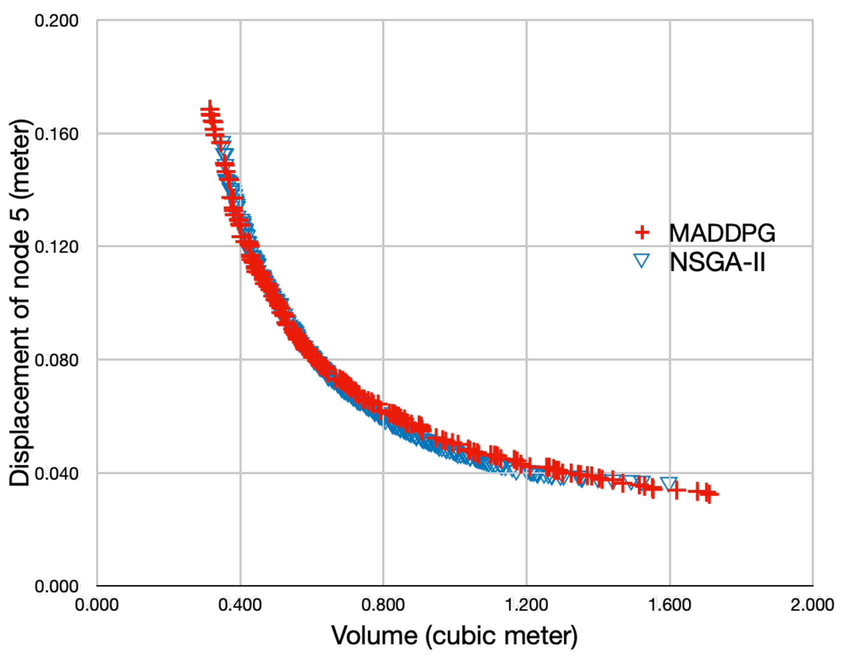


Figure 6.11: Pareto fronts obtained by MADDPG and NSGA-II for multi-objective 10-bar truss optimization problem

6.6 Trade-off design problem using MARL and graph representation

6.6.1 Observation

Observation data in this experiment are formulated similarly to those in Section 6.4.1 with some additional information about the structural node i of the truss in the node feature matrix of truss node ($\mathbf{N}$), as shown in Table 6.11. All entries in the adjacency and weighted adjacency matrices ($\mathbf{M}, \mathbf{P}_\mathrm{s}, \mathbf{P}_\mathrm{t}, \mathbf{P}_\mathrm{c}$) are similar to those in Table 6.8 in Section 6.4.1. Entries in the node feature matrix of the non-dominated solutions ($\mathbf{N}_\mathrm{p}$) are shown in Table 6.12, where W_i and D_i indicate the structural volume and the shape deviation of solution i from the target shape, respectively. Also in this table, W_max and D_max are the maximum possible values of the structural volume and the shape deviation, respectively. $\mathbf{M}_\mathrm{p}$ is the same as explained in Section 6.2.2.

Table 6.11: Values in row i of the node feature matrix of truss node ($\mathbf{N}$) for trade-off design problem

$n_{i,1}$	x-coordinate
$n_{i,2}$	y-coordinate
$n_{i,3}$	1 if there is a support in the x-axis of structural node i; else 0
$n_{i,4}$	1 if there is a support in the y-axis of structural node i; else 0
$n_{i,5}$	load in the y-direction
$n_{i,6}$	1 if structural node i is the upper node; else 0
$n_{i,7}$	1 if structural node i is the lower node; else 0
$n_{i,8}$	maximum adjustable height of structural node i in positive y-direction
$n_{i,9}$	maximum adjustable height of structural node i in negative y-direction
$n_{i,10}$	$n_{i,6}\hat{y}_i/y_i$
$n_{i,11}$	displacement in the y-direction
$n_{i,12}$	1 if structural node i violates displacement constraint; else 0
$n_{i,13}$	1 if structural node i violates displacement constraint; else $0.5n_{i,11}/\bar{\delta}$

Table 6.12: Values in row i of the node feature matrix of the non-dominated solutions ($\mathbf{N}_\mathrm{p}$) for trade-off design problem

$n_{\mathrm{p}_{i,1}}$	$W_i/W_{\max}$
$n_{\mathrm{p}_{i,2}}$	$D_i/D_{\max}$
$n_{\mathrm{p}_{i,3}}$	the current number of the non-dominated solutions/ $n_{\max}$
$n_{\mathrm{p}_{i,4}}$	1 if agents are adjusting this solution, else 0

6.6.2 Action

There are two design variables in this multi-objective optimization problem, including the nodal heights and the cross-sectional areas of the truss. Therefore, each agent u computes two outputs $\{\boldsymbol{\pi}^{u,\mathrm{G}}, \boldsymbol{\pi}^{u,\mathrm{T}}\}$, associated with each design variable, to create neighborhood solutions. The agent modifies the nodal height of node i utilizing the output $\boldsymbol{\pi}^{u,\mathrm{G}} \in \mathbb{R}^{n_\mathrm{s}\times 2}$, interpreted as follows:

$$y_i = \begin{cases} y_i + n_{i,8}k\boldsymbol{\pi}^{u,\mathrm{G}}_{i,1} & : \text{if } \underset{j}{\operatorname{argmax}}\left(\boldsymbol{\pi}^{u,\mathrm{G}}_{i,j\in\{1,2\}}\right) = 1 \\ y_i - n_{i,9}k\boldsymbol{\pi}^{u,\mathrm{G}}_{i,2} & : \text{if } \underset{j}{\operatorname{argmax}}\left(\boldsymbol{\pi}^{u,\mathrm{G}}_{i,j\in\{1,2\}}\right) = 2 \end{cases} \tag{6.16}$$

where $0 < k \leq 1$ is a coefficient to scale the adjustments.

The agents modify the cross-sectional area of multiple elements similarly to the example in Section 6.4 where each cross-sectional index c_j of an element j is derived from the product of $\boldsymbol{\pi}^{u,\mathrm{T}} \in \mathbb{R}^{n_\mathrm{s} \times 3}$ and the connectivity matrix $\mathbf{C} \in \mathbb{R}^{n_\mathrm{e} \times n_\mathrm{s}}$, which is denoted as $\boldsymbol{\Pi}^u \in \mathbb{R}^{n_\mathrm{e} \times 3}$, and determines how the cross-sectional area of element j is modified as follows:

$$c_j = \begin{cases} \min(n_\mathrm{c}, c_j + 1) & : \text{if } \underset{k}{\operatorname{argmax}}\left(\boldsymbol{\Pi}^u{}_{j,k=\{1,2,3\}}\right) = 1 \\ c_j & : \text{if } \underset{k}{\operatorname{argmax}}\left(\boldsymbol{\Pi}^u{}_{j,k=\{1,2,3\}}\right) = 2 \\ \max(0, c_j - 1) & : \text{if } \underset{k}{\operatorname{argmax}}\left(\boldsymbol{\Pi}^u{}_{j,k=\{1,2,3\}}\right) = 3 \end{cases} \tag{6.17}$$

6.6.3 Agents

Let u_1, u_2, and u_3 be the agents. Similar to the example in Section 6.4.3, the policy function of the agent u_1 takes $\mathbf{N}$, $\widetilde{\mathbf{M}}$, $\mathbf{P}_\mathrm{s}$, $\mathbf{P}_\mathrm{t}$, $\mathbf{P}_\mathrm{c}$, $\mathbf{N}_\mathrm{p}$, and $\widetilde{\mathbf{M}}_\mathrm{p}$ as observation data and computes the modifications of geometry and member sizes of the truss interpreted utilizing Eqs. (6.16) and (6.17) as follows:

$$\{\boldsymbol{\pi}^{u_1,\mathrm{G}}, \boldsymbol{\pi}^{u_1,\mathrm{T}}\} = \boldsymbol{\pi}^{u_1}\left(\mathbf{N}, \widetilde{\mathbf{M}}, \mathbf{P}_\mathrm{s}, \mathbf{P}_\mathrm{t}, \mathbf{P}_\mathrm{c}, \mathbf{N}_\mathrm{p}, \widetilde{\mathbf{M}}_\mathrm{p}\right) \tag{6.18}$$

The value function of agent u_1 computes the Q^{u_1} utilizing the observation data, outputs of $\boldsymbol{\pi}^{u_1}$, and outputs of all the other policy functions as inputs, respectively, as

$$\mathrm{Q}^{u_1} = \mathrm{Q}^{u_1}\left(\mathbf{N}, \widetilde{\mathbf{M}}, \mathbf{P}_\mathrm{s}, \mathbf{P}_\mathrm{t}, \mathbf{P}_\mathrm{c}, \mathbf{N}_\mathrm{p}, \widetilde{\mathbf{M}}_\mathrm{p}, \boldsymbol{\pi}^{u_1,\mathrm{G}}, \boldsymbol{\pi}^{u_1,\mathrm{T}}, \boldsymbol{\pi}^{u_2,\mathrm{G}}, \boldsymbol{\pi}^{u_2,\mathrm{T}}, \boldsymbol{\pi}^{u_3,\mathrm{G}}, \boldsymbol{\pi}^{u_3,\mathrm{T}}\right) \tag{6.19}$$

Table 6.13 summarizes the computation processes of the policy and value functions of the GCN-MADDPG u_1 agent for this trade-off design problem. The architectures of actor and critic networks are illustrated in Figures 6.12(a) and 6.12(b), respectively. The GCNs with ReLU activation function return $\mathbf{N}' \in \mathbb{R}^{n \times h}$ with h as 200 (i.e., the number of feature maps, explained in Section 2.5), and MLPs have 2 layers with 200 neurons in each layer. Note that all equations and symbols in the computation processes are explained in Section 2.6 and Figure 2.6 in Chapter 2.

Table 6.13: Policy and value functions of GCN-MADDPG (u_1 agent) for trade-off design problem

policy function $\boldsymbol{\pi}^{u_1}$	value function Q^{u_1}
input: $\mathbf{N}, \widetilde{\mathbf{M}}, \mathbf{P}_\mathrm{s}, \mathbf{P}_\mathrm{t}, \mathbf{P}_\mathrm{c}, \mathbf{N}_\mathrm{p}, \widetilde{\mathbf{M}}_\mathrm{p}$	input: $\mathbf{N}, \widetilde{\mathbf{M}}, \mathbf{P}_\mathrm{s}, \mathbf{P}_\mathrm{t}, \mathbf{P}_\mathrm{c}, \mathbf{N}_\mathrm{p}, \widetilde{\mathbf{M}}_\mathrm{p}, \boldsymbol{\pi}^{u_1,\mathrm{G}}, \boldsymbol{\pi}^{u_1,\mathrm{T}}, \boldsymbol{\pi}^{u_2,\mathrm{G}}, \boldsymbol{\pi}^{u_2,\mathrm{T}}, \boldsymbol{\pi}^{u_3,\mathrm{G}}, \boldsymbol{\pi}^{u_3,\mathrm{T}}$
computation: step 1: $\mathbf{N}_{1.1} = \mu(\mu(\mathbf{N}, \widetilde{\mathbf{M}}), \widetilde{\mathbf{M}})$ step 2: $\mathbf{N}_{1.2} = \mu(\mu(\mathbf{N}, \widetilde{\mathbf{M}}), \mathbf{P}_\mathrm{s})$ step 3: $\mathbf{N}_{1.3} = \mu(\mu(\mathbf{N}, \widetilde{\mathbf{M}}), \mathbf{P}_\mathrm{t})$ step 4: $\mathbf{N}_{1.4} = \mu(\mu(\mathbf{N}, \widetilde{\mathbf{M}}), \mathbf{P}_\mathrm{c})$ step 5: $\mathbf{N}_{1.5} = \mu(\mathbf{N}_\mathrm{p}, \widetilde{\mathbf{M}}_\mathrm{p})$ $\mathbf{N}_{1.6} = \mathrm{Stack}^n(\mathrm{GSP}(\mathbf{N}_{1.5}))$ $\mathbf{N}_{1.7} = \mu(\mathbf{N}_{1.6}, \widetilde{\mathbf{M}})$ step 6: $\mathbf{N}_{2.1} = \mathbf{N}_{1.1} + \mathbf{N}_{1.2} + \mathbf{N}_{1.3} + \mathbf{N}_{1.4} + \mathbf{N}_{1.7}$ $\mathbf{N}_{2.2} = \mu(\mathbf{N}_{2.1}, \widetilde{\mathbf{M}})$ step 7: $\mathbf{N}_{3.1} = \mathbf{N}_{1.1} + \mathbf{N}_{1.2} + \mathbf{N}_{1.3} + \mathbf{N}_{1.4} + \mathbf{N}_{1.7}$ $\mathbf{N}_{3.2} = \mu(\mathbf{N}_{2.1}, \widetilde{\mathbf{M}})$ step 7: $\boldsymbol{\pi}^{u_1,\mathrm{G}} = \sigma(\mathbf{N}_{2.2}, \widetilde{\mathbf{M}})$ $\boldsymbol{\pi}^{u_1,\mathrm{T}} = \sigma(\mathbf{N}_{3.2}, \widetilde{\mathbf{M}})$	computation: step 1: $\mathbf{n}_{1.1} = \mathrm{GSP}\left(\mu(\mu(\mathbf{N}, \widetilde{\mathbf{M}}), \widetilde{\mathbf{M}})\right)$ step 2: $\mathbf{n}_{1.2} = \mathrm{GSP}\left(\mu(\mu(\mathbf{N}, \widetilde{\mathbf{M}}), \mathbf{P}_\mathrm{s})\right)$ step 3: $\mathbf{n}_{1.3} = \mathrm{GSP}\left(\mu(\mu(\mathbf{N}, \widetilde{\mathbf{M}}), \mathbf{P}_\mathrm{t})\right)$ step 4: $\mathbf{n}_{1.4} = \mathrm{GSP}\left(\mu(\mu(\mathbf{N}, \widetilde{\mathbf{M}}), \mathbf{P}_\mathrm{c})\right)$ step 5: $\mathbf{N}_{1.5} = \mu(\mathbf{N}_\mathrm{p}, \widetilde{\mathbf{M}}_\mathrm{p})$ $\mathbf{N}_{1.6} = \mathrm{Stack}^n(\mathrm{GSP}(\mathbf{N}_{1.5}))$ $\mathbf{n}_{1.7} = \mathrm{GSP}\left(\mu(\mathbf{N}_{1.6}, \widetilde{\mathbf{M}})\right)$ step 6: $\mathbf{n}_{1.8} = \mathrm{GSP}\left(\mu(\mu(\boldsymbol{\pi}^{u_1,\mathrm{G}}, \widetilde{\mathbf{M}}), \widetilde{\mathbf{M}})\right)$ step 7: $\mathbf{n}_{1.9} = \mathrm{GSP}\left(\mu(\mu(\boldsymbol{\pi}^{u_1,\mathrm{T}}, \widetilde{\mathbf{M}}), \widetilde{\mathbf{M}})\right)$ step 8: $\mathbf{n}_{1.10} = \mathrm{GSP}\left(\mu(\mu(\boldsymbol{\pi}^{u_2,\mathrm{G}}, \widetilde{\mathbf{M}}), \widetilde{\mathbf{M}})\right)$ step 9: $\mathbf{n}_{1.11} = \mathrm{GSP}\left(\mu(\mu(\boldsymbol{\pi}^{u_2,\mathrm{T}}, \widetilde{\mathbf{M}}), \widetilde{\mathbf{M}})\right)$ step 10: $\mathbf{n}_{1.12} = \mathrm{GSP}\left(\mu(\mu(\boldsymbol{\pi}^{u_3,\mathrm{G}}, \widetilde{\mathbf{M}}), \widetilde{\mathbf{M}})\right)$ step 11: $\mathbf{n}_{1.13} = \mathrm{GSP}\left(\mu(\mu(\boldsymbol{\pi}^{u_3,\mathrm{T}}, \widetilde{\mathbf{M}}), \widetilde{\mathbf{M}})\right)$ step 12: $\mathbf{n}_2 = [\mathbf{n}_{1.1} \parallel \mathbf{n}_{1.2} \parallel \mathbf{n}_{1.3} \parallel \mathbf{n}_{1.4} \parallel \mathbf{n}_{1.7} \parallel \mathbf{n}_{1.8} \parallel \mathbf{n}_{1.9} \parallel \mathbf{n}_{1.10} \parallel \mathbf{n}_{1.11} \parallel \mathbf{n}_{1.12} \parallel \mathbf{n}_{1.13}]$ step 13: $\mathrm{Q}^{u_1} = \mathrm{f}_{\mathrm{NN}}(\mathbf{n}_2)$
output: $\boldsymbol{\pi}^{u_1,\mathrm{G}} \in \mathbb{R}^{n_\mathrm{s} \times 3}, \boldsymbol{\pi}^{u_1,\mathrm{T}} \in \mathbb{R}^{n_\mathrm{s} \times 3}$	output: $\mathrm{Q}^{u_1} \in \mathbb{R}^{1 \times 1}$

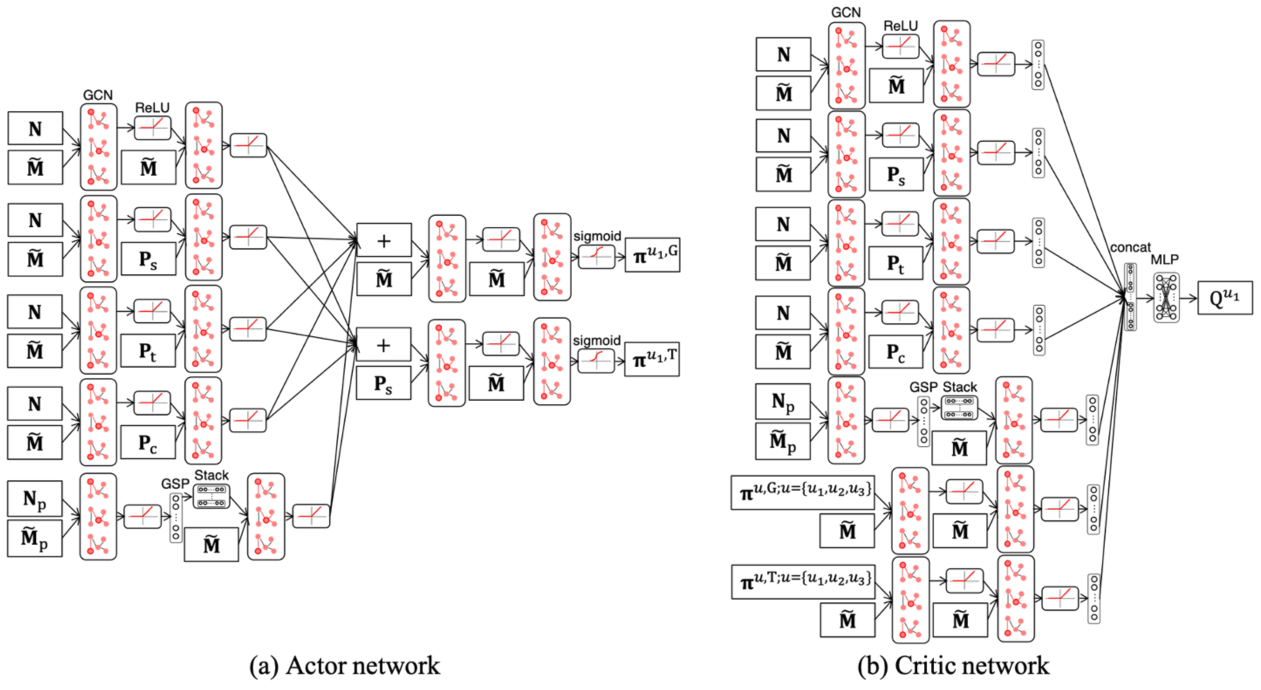


Figure 6.12: Actor and critic networks for trade-off design problem

6.6.4 Numerical settings

In this example, truss elements have Young's modulus and allowable stress for both tension and compression of 200 kN/mm^2 and 235/1.5 N/mm^2, respectively, with a list of sections shown in Table 6.14. The allowable nodal deformation is 0.001×length of the total truss span (m). The agents modify the nodal heights using $k = 0.25$ in Eq. (6.16). Note that target heights $\hat{y}_i$ in Eq. (6.4) are assigned only to the upper nodes of the truss structures.

Table 6.14: List of truss sections for trade-off design problem

section no.	diameter (mm)	thickness (mm)	area (cm^2)	I (cm^4)	Z (cm^3)
1	76.3	4.0	9.085	5.95×10^1	15.6
2	114.3	6.0	20.41	3.00×10^2	52.5
3	139.8	9.5	38.89	8.30×10^2	119
4	216.3	12.7	81.23	4.23×10^3	391
5	318.5	17.4	164.6	1.87×10^4	1180

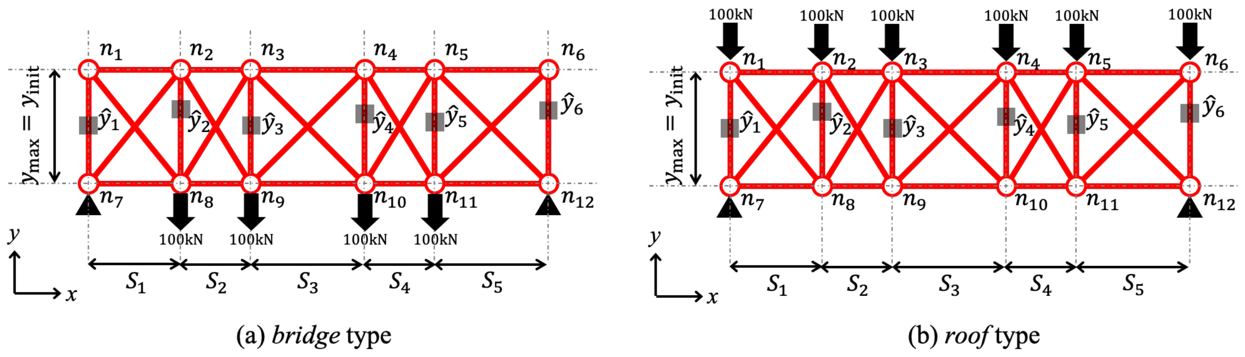
6.6.5 Training phase

The agents are trained to optimize five different trusses shown in Table 12. In this table, y_{init}, y_{max}, $\{S_{i\in\{1,\ldots,5\}}\}$, $\{\hat{y}_{i\in\{1,\ldots,6\}}\}$, and Δ denote the initial upper nodal heights, the maximum nodal heights, span lengths, target nodal heights, and the minimum depth value, respectively. Two types of loading conditions are referred to as *bridge* and *roof* types as shown in Figures 6.13(a) and 6.13(b), respectively. In *bridge* type, the truss is subjected to a downward nodal load of 100kN at each free lower node and all lower nodal heights are fixed. In *roof* type, the truss is subjected to a downward nodal load of 100kN at all upper nodes. Note that heights of all lower nodes are initialized as 0, and Δ is set so that the upper and lower nodes are not coalescent, and the minimum nodal height is 0. The nodal heights with the support are set to 0 and cannot be modified. A *roof* type truss, shown in Figure 6.14, is utilized to evaluate the abilities of agents to improve their policies, every 10 training episodes.

The agent surrogate functions, as explained in Section 2.3.3 in Chapter 2, are adjusted using the Adam optimizer, as explained in Section 2.5.3. The mini-batch size is set to 32 and the learning rates are 10^{-8} and 10^{-7} for policy and value functions, respectively. Note that the details of the training process are explained in Section 2.5.2. Weights and biases of the surrogate functions are synchronized with those of the online functions every 300 steps using τ=0.005, as explained in Section 2.3.3.

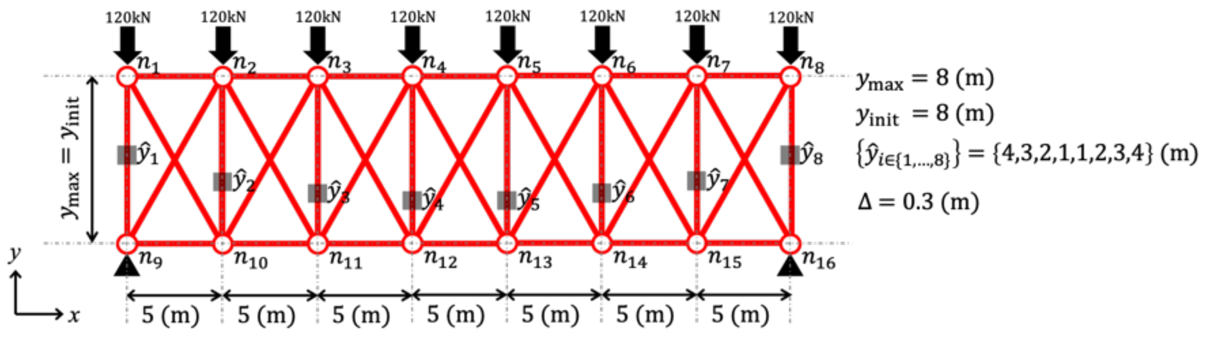
Table 6.15: Trusses for training MADDPG agents for trade-off design problem

	truss 1	truss 2	truss 3	truss 4	truss 5
y_{init} (m)	5	5	5	5	5
y_{max} (m)	5	5	5	5	5
$\{S_{i\in\{1,\ldots,5\}}\}$ (m)	{4,3,5,3,5}	{4,3,5,3,5}	{4,3,5,3,5}	{4,3,5,3,5}	{4,3,5,3,5}
$\{\hat{y}_{i\in\{1,\ldots,6\}}\}$ (m)	{1,1.5,2,2,1.5,1}	{1,3,3,2,1.5,1}	{1,1.5,2,3,3,1}	{1,3,2,2,3,1}	{3,2,1,1,2,3}
Δ (m)	0.2	0.2	0.2	0.2	0.2



(a) *bridge* type
(b) *roof* type
n_1 n_2 n_3 n_4 n_5 n_6
n_7 n_8 n_9 n_{10} n_{11} n_{12}
$\hat{y}_1$ $\hat{y}_2$ $\hat{y}_3$ $\hat{y}_4$ $\hat{y}_5$ $\hat{y}_6$
$y_{\max} = y_{\text{init}}$
100kN
S_1 S_2 S_3 S_4 S_5
y x

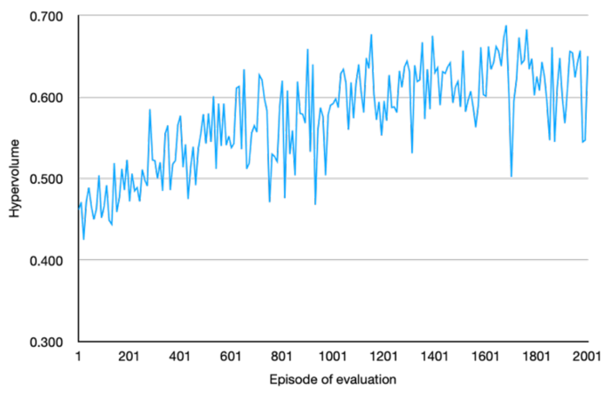
Figure 6.13: Trusses for training for trade-off design problem



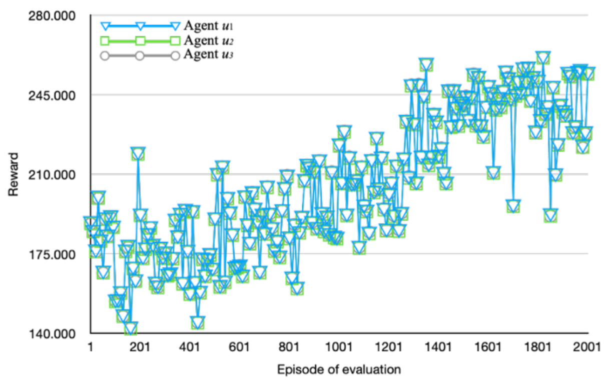
120kN
n_1 n_2 n_3 n_4 n_5 n_6 n_7 n_8
n_9 n_{10} n_{11} n_{12} n_{13} n_{14} n_{15} n_{16}
$\hat{y}_1$ $\hat{y}_2$ $\hat{y}_3$ $\hat{y}_4$ $\hat{y}_5$ $\hat{y}_6$ $\hat{y}_7$ $\hat{y}_8$
$y_{\max} = y_{\text{init}}$
$y_{\max} = 8$ (m)
$y_{\text{init}} = 8$ (m)
$\{\hat{y}_{i\in\{1,\dots,8\}}\} = \{4,3,2,1,1,2,3,4\}$ (m)
$\Delta = 0.3$ (m)
5 (m)
y x

Figure 6.14: Truss for evaluation (*roof* type) for trade-off design problem

In this training phase, the MADDPG agents are trained for 2001 episodes, each episode terminates after 50 steps, and two edge solutions are added to the solution pool at each optimization step. Figure 6.15 illustrates the history of the performance indicators, including the obtained hypervolume (a) and the reward (b), of the agents in each evaluation episode where horizontal and vertical axes represent episodes and indicators, respectively. It is seen from this figure that both the obtained hypervolume and rewards of all agents increase during the training, verifying the effectiveness of the training. It is worth noting that, in this experiment, the truss in evaluation is larger than those utilized for the training. Hence, larger oscillations compared to those in Section 6.4 are observed in both hypervolume and rewards.



(a) Hypervolume



(b) Reward

Figure 6.15: Variation of hypervolume and reward in training phase for trade-off design problem; (a) Hypervolume, (b) Reward

6.6.6 Test phase

Trained MADDPG agents are utilized for optimizing the trusses shown in Table 6.16 ten times without further training. In this table, trusses A and B have the same y_{init}, y_{max}, number of spans, $\{S_{i\in\{1,\ldots,7\}}\}$, target shape $\{\hat{y}_{i\in\{1,\ldots,8\}}\}$, and Δ whereas the values of trusses C and D are the same. Similar to those in the training phase, all the lower nodal heights are fixed, and a downward nodal load is applied at each free lower node of the *bridge* type trusses while a downward nodal load is applied at each upper node for the *roof* type trusses. In both types, the minimum nodal height is set to 0. The heights of supports cannot be modified.

Table 6.16: Trusses for test for trade-off design problem

	truss A	truss B	truss C	truss D
y_{init} (m)	8		6	
$y_{\max}$ (m)	8		6	
number of spans	7		15	
$\{S_i\}$ (m)	{5,5,5,5,5,5,5}		{5,5,5,5,5,5,5,5,5,5,5,5,5,5,5}	
$\{\hat{y}_i\}$ (m)	{4, 3, 2.5, 2, 2, 2.5, 3, 4}		{3, 2.75, 2.5, 2.25, 2.25, 2, 2, 2, 2, 2, 2, 2.25, 2.25, 2.5, 2.75, 3}	
Δ (m)	0.3		0.3	
nodal load (kN)	75	120	7.5	8.0
type	*bridge*	*roof*	*bridge*	*roof*

In this test phase, the trained MADDPG agents have the weights saved at episode 2000th from the training phase. Each test episode is terminated after 500 steps, $n_{\max}$ is set to 50, and ten edge solutions are added to the solution pool at each optimization step. Note that $n_{\max}$ is larger than those of the training phase because the test phase aims for the performance to obtain Pareto solutions. A larger number of edge solutions added in this phase are also from this rationale. Unlike those in the training phase, the symmetry conditions are assigned to the modification of the trusses in this test phase. The symmetrically located nodes and elements have the same height and section, which are modified using the action output $\boldsymbol{\pi}^{u,\mathrm{G}}$ selected randomly from one of the pairs. Sectional indices of members in each symmetric pair are chosen from the smaller index in the pair.

The obtained results from MADDPG agents are compared to those of NSGA-II using DEAP with the same computational cost. In this problem, the nodal heights and the section indices of the elements are represented as the genes of NSGA-II. Note that symmetry nodes and elements are also set in NSGA-II similar to MARL. The NSGA-II algorithm is applied 10 times with different random seeds. The number of generations is set to 20.

Table 6.17 shows the minimum (min.), the mean, the maximum (max.), and the standard deviation (std.) of the size of hypervolume, the average spreads (i.e., lengths of the Pareto front) ($\mathrm{L_{avg}}$), the coefficient of variation of the crowd distances of the Pareto front ($\mathrm{CD_{avg}}$), and the numbers of structural analyses utilized by the MADDPG agents and NSGA-II. In NSGA-II cases, the number of populations is provided in the last column for NSGA-II. Results from this Table indicate that MADDPG agents obtained larger hypervolumes in all the cases with low standard deviation, showing the stability of the MARL. The proposed method also obtained larger spreads and lower differences in crowd distances (i.e., better distribution).

Table 6.17: Pareto front indicators of test trusses obtained by MARL and NSGA-II for trade-off design problem

truss	method	min.	mean	max.	std.	L_{avg}	CD_{avg}	analyses	population
A	MARL	0.823	0.853	0.886	0.020	0.307	0.807	174831	-
	NSGA-II	0.747	0.785	0.831	0.028	0.339	1.104	180000	900
B	MARL	0.794	0.830	0.868	0.027	0.332	0.853	126564	-
	NSGA-II	0.559	0.625	0.709	0.046	0.230	1.050	128000	640
C	MARL	0.715	0.763	0.811	0.030	0.333	0.678	119340	-
	NSGA-II	0.455	0.557	0.633	0.056	0.176	0.975	120000	600
D	MARL	0.706	0.808	0.850	0.044	0.471	0.779	238539	-
	NSGA-II	0.301	0.443	0.640	0.093	0.163	0.668	240000	12000

Figures 6.16-6.19 illustrate the results obtained by MARL for each truss in the test phase. In these figures, subfigures (a) are Pareto fronts in the objective space obtained by MARL (color × marks) and NSGA-II (grey ▼marks). These subfigures indicate that MARL generally outperforms NSGA-II in obtaining large hypervolumes measured by the Pareto fronts and their proximity to the point (0,0) in the objective spaces. Subfigures (b) illustrate the solutions obtained by MARL, where Roman numbers indicate their positions on the objective space in subfigures (a), the width of each element is proportional to the cross-sectional area, and the transparent red dots represent the target heights.

It is confirmed from these subfigures, the obtained Pareto solutions by MARL consist of the solution that has a small structural volume (Roman number i), the one with a small shape deviation (Roman number vii in Figures 6.16 and 6.17, and number ix in Figures 6.18 and 6.19), and other non-dominated solutions in between. It is worth noting that the target shapes are specified so that they are not optimal geometries for minimizing the structural volume under stress and displacement constraints. Hence, there are trade-off relations between minimizing the volumes and shape deviations. The solutions obtained by MARL show that trusses with small volumes have arch-like shapes, which are near-optimal geometries, but have large shape deviations, whereas solutions with small shape deviations have large internal stresses, that require large sectional areas of the elements and large structural volumes.

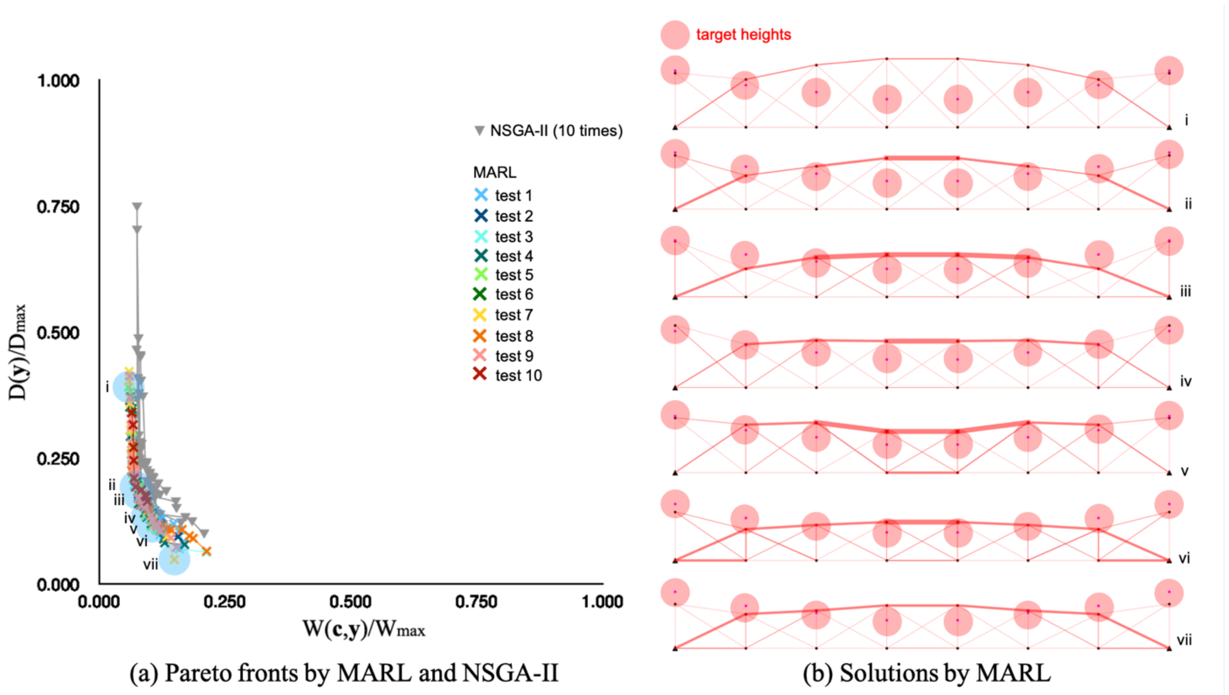


Figure 6.16: Pareto front obtained by MARL for trade-off design problem – truss A (*bridge* type)

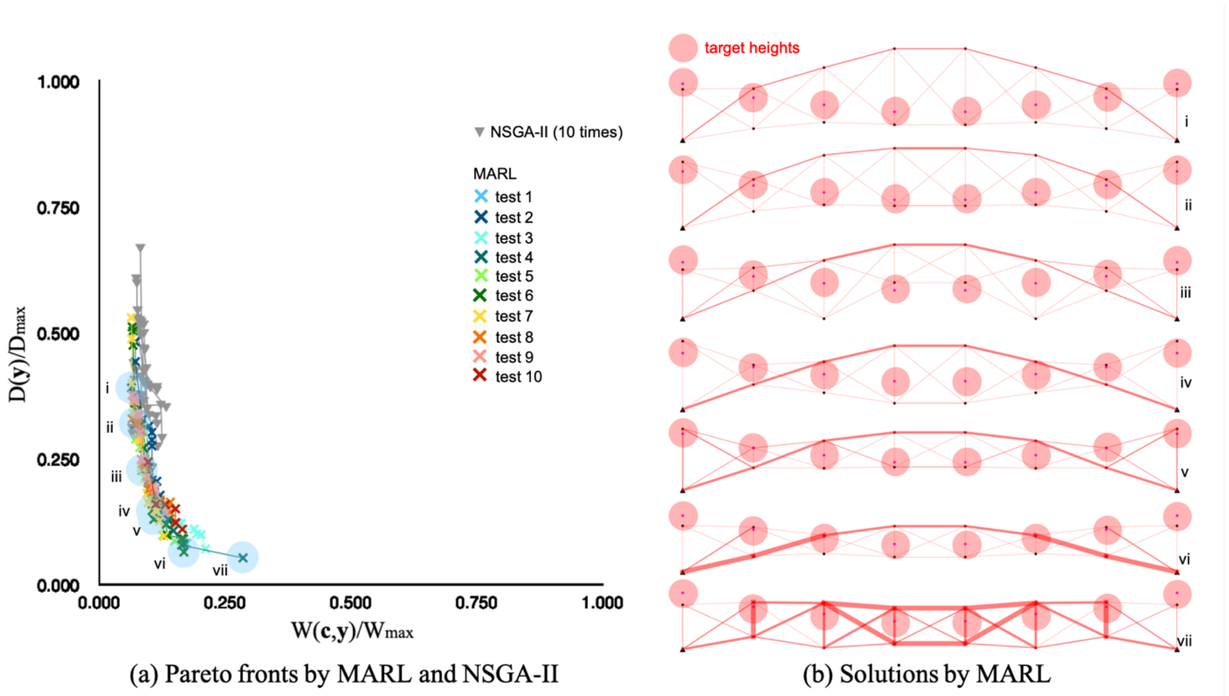
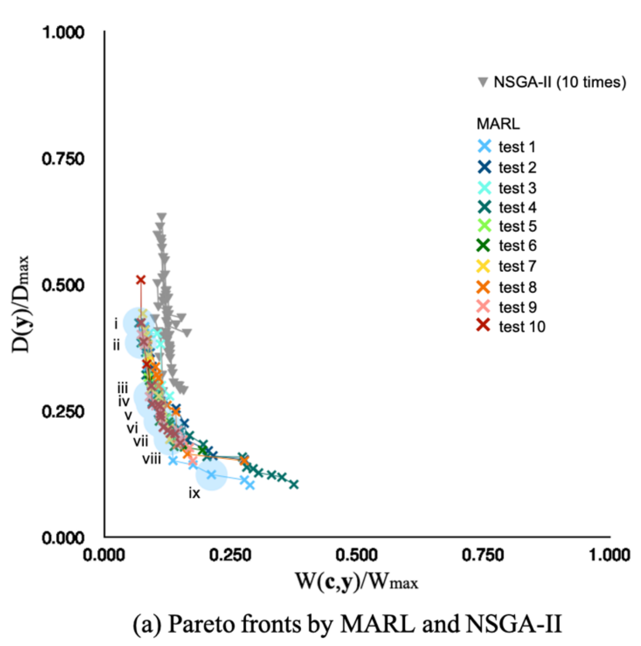


Figure 6.17: Pareto front obtained by MARL for trade-off design problem – truss B (*roof* type)



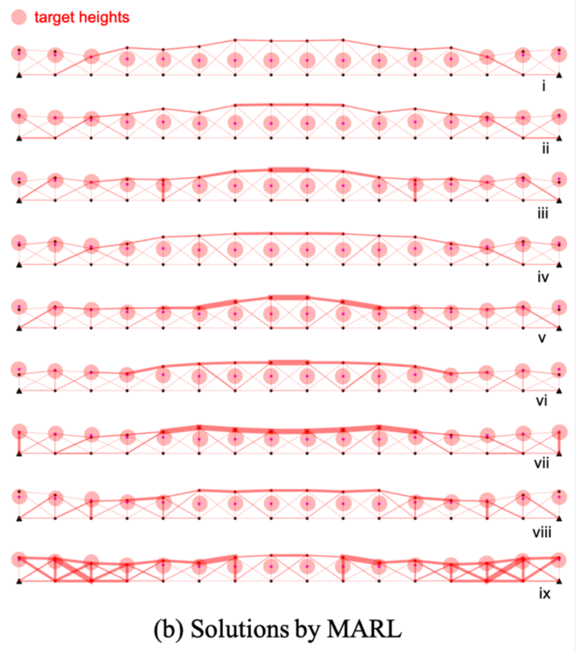
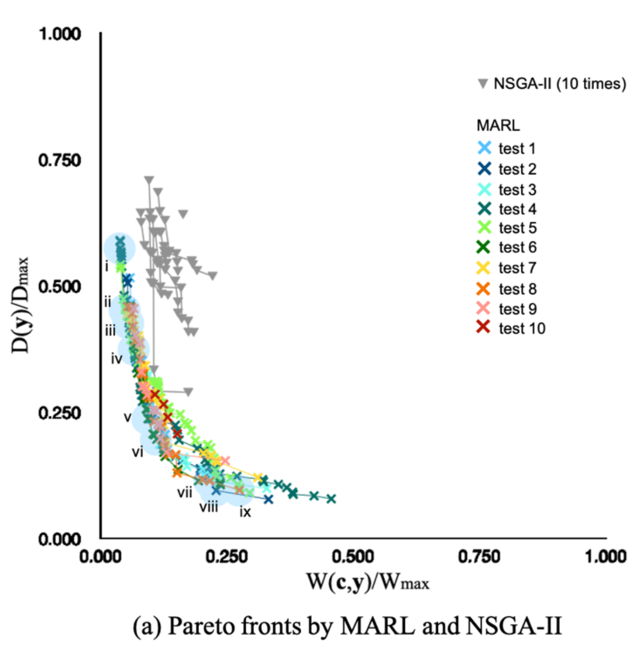


Figure 6.18: Pareto front obtained by MARL for trade-off design problem – truss C (*bridge* type)



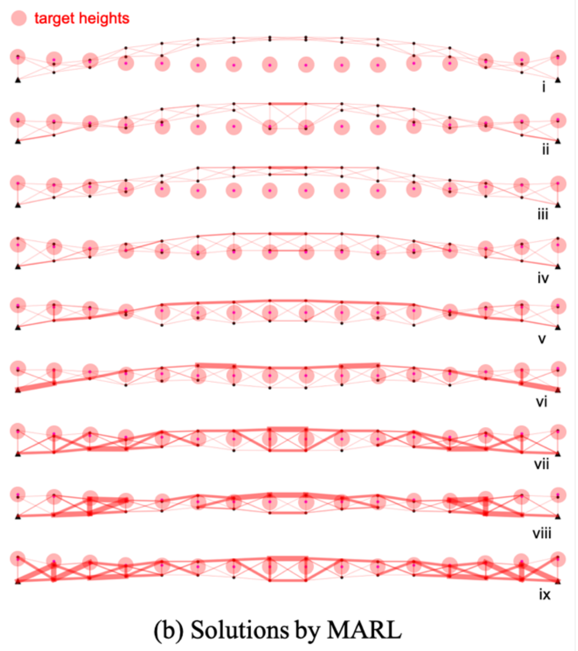


Figure 6.19: Pareto front obtained by MARL for trade-off design problem – truss D (*roof* type)

6.7 Conclusion

This chapter introduced an innovative approach to address the complex challenges of multi-objective optimization in truss structures by leveraging MARL. The methodology employs MADDPG agents which iteratively improve solutions in the solution pool to obtain the Pareto front. In each optimization step, the agents observe the non-dominated solutions in the objective space utilizing the proposed graph representation. The optimization problem is formulated as a Markov game where multiple agents learn to work together to obtain Pareto fronts. The objective is to collaboratively refine solutions within the objective space, ultimately yielding Pareto optimal solutions for three distinct multi-objective optimization problems.

(1) A novel graph representation is introduced to encapsulate the current set of non-dominated solutions. This representation allows the MADDPG agents to concurrently observe both the present state of the objective space and the modified truss structures. These agents are made from GCNs and MLPs, providing them with the versatility to process various forms of state data.

(2) A comprehensive reward function is formulated, taking into account multiple facets of optimization improvement. It considers enhancements in the hypervolume, objective functions, hypervolume size, front spread, and distribution. This reward function captures the essence of multi-objective optimization, enabling the agents not only to seek Pareto optimality but also to consider the distribution and diversity of solutions. Numerical examples in this chapter show that, utilizing this reward formulation, the agent can improve their obtained rewards and the size s of the hypervolume.

(3) The adaptability and utility of the MADDPG agents are demonstrated across various problems, including mathematical and 10-bar truss multi-objective optimization problems. In these domains, the agents are successfully trained to obtain both true and approximate Pareto solutions, showcasing their versatility and applicability across diverse optimization scenarios. This adaptability underscores the broad range of problems that can benefit from the proposed MARL-based approach.

(4) In the context of the multi-objective optimization problem involving the trade-off design of trusses, the trained MADDPG agents exhibit their capacity to enhance the obtained

hypervolume and reward. This practical application highlights the efficacy of the proposed method in addressing real-world engineering optimization challenges, particularly in scenarios where trade-off relations among multiple objectives are essential.

(5) The most significant finding is the consistent outperformance of the trained MADDPG agents compared to the well-established NSGA-II method across all structural problems in the test phase in the trade-off design of trusses. This robust performance underscores the potential of the proposed MARL-based approach as an effective optimization tool for solving multi-objective optimization problems in the field of practical structural engineering. Notably, the RL approach not only delivers superior results but also demonstrates its stability in obtaining large hypervolume shown in the standard deviations which can be observed in all chapters in this dissertation.

In summary, this chapter represents an advancement in the domain of multi-objective optimization for truss structures. The innovative combination of MARL techniques, including MADDPG agents, with novel graph representations and comprehensive reward functions, offers a powerful approach to tackling complex optimization challenges. The adaptability and applicability of the approach across various optimization contexts highlight its potential in structural engineering, where optimizing multiple and often conflicting objectives is a common challenge. The consistent outperformance of the MADDPG agents compared to NSGA-II reaffirms the potential of the proposed approach to solve multi-objective optimization problems, offering efficient and effective solutions for complex structural engineering problems. Note that this chapter considers only vertical loads applied to the structure, which has a direct effect on the optimal geometry and size of the trusses.

Chapter 7
Conclusion and Future directions

7.1 Summary

This dissertation represents a comprehensive investigation into the application of RL approaches to address a spectrum of optimization problems in truss and frame structures. These endeavors span topology, size, and geometry optimizations, all within the context of diverse objective functions and constraints. The research objective is to mitigate the computational costs inherently associated with iterative finite element simulations throughout the optimization process. This becomes particularly important as structural complexity escalates, resulting in larger structures or highly discretized finite element models. Additionally, the dissertation seeks to contribute to the field by introducing RL optimization methods capable of seamlessly accommodating both discrete and continuous design variables within structural optimization, regardless of the presence or absence of a gradient between the objective functions and design variables. The proposed methods in this research relies on two important foundations including the RL and the graph representation. The first foundation allows the creation of learnable model that can be utilized for solving structural optimization problems while the second foundation has a main influence from Hayashi et al. [50] which represents structure as a graph data. This research has also influenced by different graph represents for architectural and building engineering data such as Langenhean et al. [144], Abualdenien and Borrmann [97], and Vestartas [145] which represent room types, patterns of building elements, and structural elements and joints using graph data, respectively.

In Chapter 1, the research background is presented, laying the groundwork for the subsequent investigations. This chapter illuminates the pressing need for computational efficiency in structural optimization, especially in the face of increasingly intricate and expansive structures. Chapter 2 serves as an educational cornerstone, offering a discerning review of fundamental concepts in ML and RL. This review serves the dual purpose of acquainting readers unfamiliar with these methodologies with the requisite knowledge and providing an academic foundation upon which the ensuing chapters build. Chapters 3 through 6 embody the essence of the research, wherein innovative RL-based methodologies are introduced and elucidated, addressing topology, size, and geometry optimization problems in truss and frame structures. These chapters represent the core of the dissertation, encompassing the development, analysis, and evaluation of novel methodologies. The findings and contributions of this dissertation are summarized as follows.

7.1.1 Topology and sizing optimizations of lattice shells

In Chapter 3, Topology and sizing optimization problems are formulated to minimize the strain energy and structural volume, respectively. The graph representation of lattice shells is utilized for making state data for the DDPG agent. In this representation, structural nodes and elements become graph nodes and edges, respectively. Structural nodes, elements, and internal forces are represented using the node feature matrix, adjacency matrix, and weight adjacency matrix, respectively. The RL agents in both problems are made of GCNs and MLPs enabling the agent to process graph representations of the structures and modify all graph nodes, which subsequently modify all structural elements simultaneously. The optimization starts by having a lattice shell with random brace directions and a lattice shell with large lattice elements in the topology and sizing problems, respectively.

In each problem, the agent observes the graph representation data as the state, by taking it as inputs, to compute the actions that are choosing bracing directions and reducing the sectional indices in the topology and sizing problems, respectively. After the lattice shell is modified, the agents in the topology and sizing problems receive a quantitative reward computed from the change of the strain energy and structural volume, respectively. The agents in both problems keep on modifying the structure until a predetermined optimization step is reached.

In both problems, the agent is trained to optimize the small multiple structures in the training phase. The obtained rewards are tracked to confirm the improvement of the agents to optimize the structure. After the training, the agent is applied to optimize large multiple structures whose topology, geometry, and number of DoFs differ from those utilized during the training in the test phase. In both problems, the results obtained by the RL agent in this phase are compared to stochastic- and population-based optimization approaches. The comparative results show that the RL agent provides better results by utilizing fewer computational costs. Note that the limitation of the proposed method on the topology optimization of lattice shells is that it might yield inferior results compared to GA if large GA computation is allowed.

7.1.2 Simultaneous topology and sizing optimization of building frames

A simultaneous topology and sizing optimization method for building frames utilizing RL is presented in Chapter 4. This problem aims to minimize the structural cost of steel building frames subjected to seismic loads under multiple constraints. The graph representation of building frames is also utilized for making state data for the DDPG agent. However, different from those in Chapter 3, structural elements and their connectivity become graph nodes and edges, respectively. Utilizing this graph representation, the structural elements and their internal forces are represented through a node feature matrix and an adjacency matrix. The RL agent is made from GCNs and MLPs. The agent can modify the sizes of the structural elements directly.

At the beginning of the optimization problem, a brace-less building frame with small columns and beams is initialized. Note that this initial structure cannot resist the design and seismic loads and all constraints are violated. The DDPG agent observes the building frame and decides to add a brace or increase the size of an existing structural element. After this process, the finite element simulations are made to obtain the objective functions and constraints. The agent receives a reward computed from the change of the structural volume and improvements of the constraints. The optimization ends when all constraints are satisfied.

The agent is trained to optimize small multiple building frames and verified the improvement of its performance to solve this problem in the training phase. After this phase, the trained agent is applied to diverse large building frames and its results are compared to those obtained by a population-based optimization method. The results confirm the superior ability of the proposed method to simultaneous topology and sizing optimize building frames over the population-based method. The limitations the RL agent in this chapter is that it still cannot be applied to 3-dimensional building frame which has larger number of elements (i.e., graph nodes). Note that this large number of structural elements might hinder the agent to obtain good solution because there are too many actions to choose in each MDP.

7.1.3 Geometry optimization of lattice shells

The RL method for geometry optimization of lattice shells is presented in Chapter 5, to verify that this method can solve all main optimization problems of truss and frame, including topology, sizing, and geometry optimizations. In this optimization problem, the geometry of lattice shells is defined utilizing Bézier control points that are design variables while the objective function is to minimize the strain energy of the lattice shells. To represent and combine information from the structure and control points, novel graph representations and neural network architectures are introduced in this research. The structure is represented through a node feature matrix, adjacency matrices, and weight adjacency matrices, similar to those in Chapter 3. The Bézier control net is represented utilizing a node feature matrix of the Bézier control points and an adjacency matrix of the Bézier control net. The DDPG agent is made of GATs and MLPs together with GSP operations that combine graph representation data of the structure and control net. The architecture also enables the agent to determine the heights of all Bézier control points simultaneously. The optimization starts with a flat lattice shell which is observed by the agent. The agent takes the observation data as inputs and modifies the heights of all control points (i.e., does the action). After the action, the structure is changed and structural analysis is performed to obtain the structural response. The agent receives a reward based on the improvement of the structure measured utilizing the strain energy. The optimization keeps on until a predetermined optimization step is reached.

Similarly, to previous chapters, the DDPG agent is trained to optimize small multiple lattice shells whose geometries are defined using one discretization of the Bézier control net. The improvement of the ability of the agent to optimize structures is tracked and measured through the rewards obtained in this phase which indicate the positive improvement. After the training, the agent is applied to diverse structures having different numbers of DoFs, Bézier control nets, upper bounds of design constraints, and types of design variables, in the test phase. The results in this phase are compared to stochastic- and population-based optimization approaches. The results confirm that the proposed RL method outperforms the stochastic- and population-based approaches in both performance and computational cost. The limitations of the proposed method in this chapter are that the RL agent needs to be trained. Therefore, it might not be applicable to geometry optimization with different configurations of control points, such as triangular mesh. Another notable limitation is that the geometry optimization problem is generally has continuous design variables which can be solved to obtain exact optimal solution utilizing gradient-based search approaches which may yield better results than RL approaches when large computational cost is allowed.

7.1.4 Multi-objective optimization of trusses

Chapter 6 presents a novel framework for solving multi-objective optimization problems utilizing the MARL approach. The multi-objective optimization aims to obtain a large Pareto front, measured using the hypervolume. In this proposed framework, multiple MADDPG agents are trained to iteratively modify solutions in each optimization step. Once all the solutions are modified, the new solution pool is made from non-dominated solutions of the modified solutions and the current solution pool. The optimization ends when a predetermined number of the optimization steps is reached. The proposed framework is applied to a mathematical multi-objective optimization problem, a 10-bar truss multi-objective optimization problem, and a multi-objective optimization problem of trade-off truss design, respectively.

The MADDPG agents are made from GCNs and MLPs. They observe the state data including the data of the being modified solution and the data representing the objective space. The being modified solution is represented as a vector in the mathematical problem and as a graph in both truss problems. In the truss problems, structural nodes and elements become graph nodes and edges. A node feature matrix, an adjacency matrix, and weight adjacency matrices are utilized to represent data in the structural nodes, nodal connectivity, and internal forces and sectional area, respectively. A novel graph representation of the non-dominate solution is also introduced in this research, the representation provides information on the current non-dominated solutions on the objective space to the MADDPG agents. A general reward function that is applicable to all multi-objective optimization problems is also proposed. This reward is derived from the increment of the hypervolume size, the improvements of each objective, the current hypervolume size, the spread (i.e., length), and distribution (i.e., crowded distance) of the front. The agents receive this reward after all of them modify a solution.

The proposed framework is applied to the mathematical multi-objective optimization problem to verify its ability to obtain a true Pareto front. In this problem, the results from the training and test phases indicate the ability of the agents to improve their obtained reward and to obtain a true Pareto front. In the 10-bar truss multi-objective optimization problem where the structural volume and nodal deformation are to be minimized, the MADDPG agent can also improve their rewards and obtain a competitive Pareto front compared to other optimization methods from other literature. In the multi-objective optimization problem of trade-off truss design, the problem aims to minimize the structural volume and shape deviation under stress and deformation constraints. Results show that the agents can improve their performance in the training phase, reflected through the increment of the rewards and the hypervolume sizes. Once trained, the MADDPG agents are applied to multiple larger trusses than those utilized during the training. The results from MADDPG are compared to those obtained by NSGA-II utilizing similar computational costs. The comparative study shows that the proposed framework yields better solutions for the multi-objective optimization measured by the size of the hypervolume and the spread and distribution of the front. The results from the MADDPG agents in different trials are more similar compared to NSGA-II, reflected in the standard deviation of the sizes of HVs.

An important limitation of this chapter is that the current method is still applicable to only bi-objective optimization problems because the reward formulation is currently based on the values measured in the bi-objective space.

7.2 Future directions

Aside from the proposed RL approaches, this dissertation initiates problems related to the applications of these approaches to practical optimization problems that require further research.

In Chapter 3, it is desirable for the agent to optimize lattice shells with different patterns, such as triangle or hexagonal, since these patterns are more suitable to the freeform lattice shells compared to rectangle forms. The proposed method should also be able to optimize the topology and sizing simultaneously and be able to utilize structural sections that differ from those used during the training. All we need is a training framework to consider all these extensions. Once the agent is able to optimize topology and sizing simultaneously in different sectional lists and lattice patterns, it is applicable to practical optimization.

The agent in Chapter 4 should be able to optimize 3-dimensional building frames. Note that the size of the graph becomes larger along with the number of possible structural elements in a 3-dimensional structure. Therefore, it might be difficult for the RL agent to learn the task because there are many actions to be chosen from. One possible solution is to utilize a hierarchical embedding method that can reduce the size of the graph. However, there will also be a trade-off between the graph size and the loss of graph information. Apart from the application in a 3-dimensional building frame, the method should also be applicable to sectional indices not utilized during the training.

Similar to the future directions for Chapter 3, the future research for the RL agent in Chapter 5 should focus on training the agent utilizing different lattice and Bézier patterns. The extension of this chapter should also incorporate the topology and sizing, making the optimization become the layout optimization. It is also desirable if the agent can be utilized in widely used software packages such as Rhinoceros [146] or Revit [147].

In Chapter 6, future research should mainly focus on expanding the proposed method to the problem with more than two objectives. This improvement could be made by modifying a component of the reward related to the Euclidean distance into the area measurement in the *n*-dimensional space. Another possible research direction is to improve the efficiency of creating neighborhood solutions. This improvement could be made by allowing the agents to drastically modify the solutions. However, there will be a trade-off between this efficiency and stability and learning performance because the state and the next state could become very different, hindering the agents from learning effectively.

References

[1] Ohsaki M (2010) Optimization of finite dimensional structures (1st ed). CRC Press.

[2] Christensen P W, Klarbring A (2009) An introduction to structural optimization, Solid mechanics and its applications, Vol.153, Springer, Dordrecht.

[3] Ohsaki M, Swan C C (2002) Topology and geometry optimization of trusses and frames. In: *Recent Advances in Optimal Structural Design*. American Society of Civil Engineers. pp.97–123.

[4] Topping B H (1983) Shape optimization of skeletal structures: a review. Journal of Structural Engineering. Vol.109(8), pp.1933–1951.

[5] Wang D, Zhang W H, Jiang J S (2002) Truss shape optimization with multiple displacement constraints. Computer Methods in Applied Mechanics and Engineering. Vol.191, pp.3597–361.

[6] Kociecki M, Adeli H (2015) Shape optimization of free-form steel space-frame roof structures with complex geometries using evolutionary computing. Engineering Applications of Artificial Intelligence. Vol.38, pp.168–182.

[7] Ohsaki M (1997) Optimization of building structural systems using parametric multidisciplinary optimization method. J. Struct. Engng. pp.79–88.

[8] Ohsaki M, Katoh N (2005) Topology optimization of trusses with stress and local constraints on nodal stability and member intersection. Structural and Multidisciplinary Optimization. Vol.29, pp.190–197.

[9] Ohsaki M, Watada R (2008) Linear mixed integer programming for topology optimization of trusses and plates. In: Proceedings of *the 6th Int. Conf. on Computation of Shell and Spatial Structures*, IASS-IACM, Ithaca, NY.

[10] Pareto V (1906) Manual of political economy. Macmillan.

[11] Moradi A, Mirzakhani Nafchi A, Ghanbarzadeh A (2015) Multi-objective optimization of truss structures using Bees Algorithm. Scientia Iranica. Vol.22(5), pp.1789–1800.

[12] Ohsaki M, Yamakawa M, Imazeki S (2018) Multi-objective optimization of a tower-type truss using order statics. In: *Japan-China Workshop on Analysis and Optimization of Large-scale Structures*. pp.32–36.

[13] Do B, Ohsaki M (2022) Sequential sampling approach to energy-based multi-objective design optimization of steel frames with correlated random parameters. Earthquake Engineering & Structural Dynamics. Vol.51(3), pp.588–611.

[14] Cauchy A (1847) Méthode générale pour la résolution des systemes d'équations simultanées. C. R. Acad. Sci. Paris. Vol.25, pp.536–538.

[15] Davidon W C (1959) Variable Metric Method for Minimization. Technical Report ANLC5990 (Revised), Argonne National Laboratory, Argonne.

[16] Boggs P T, Tolle J W (1995) Sequential quadratic programming. Acta numerica. Vol.4, pp.1–51.

[17] Dikin I I (1967) Iterative solution of problems of linear and quadratic programming. In: *Doklady Akademii Nauk*. Russian Academy of Sciences. Vol.174(4), pp.747–748.

[18] Choi K K, Kim N H (2005) In: Structural sensitivity analysis and optimization 1: linear systems. Springer, New York, pp.119–170.

[19] Adelman H, Haftka R (1986) Sensitivity analysis of discrete structural systems. American Institute of Aeronautics and Astronautics Journal. Vol.24, pp.823–832.

[20] Kirsch U (1994) Efficient sensitivity analysis for structural optimization. Computer Methods in Applied Mechanics and Engineering. Vol.117, pp.143–156.

[21] Putresza J, Kolakowski P (1998) Sensitivity analysis of frame structures (virtual distortion method approach). International Journal for Numerical Methods in Engineering. Vol.43, pp.1085–1108.

[22] Robbins H, Monro S (1951) A stochastic approximation method. The Annals of Mathematical Statistics. Vol.22(3), pp.400–407.

[23] Kirkpatrick S, Gelatt Jr C D, Vecchi M P (1983) Optimization by simulated annealing. science. Vol.220(4598), pp.671–680.

[24] Bennage W A, Dhingra A K (1995) Single and multiobjective structural optimization in discrete-continuous variables using simulated annealing. International Journal for Numerical Methods in Engineering. Vol.38(16), pp.2753–2773.

[25] Leite J P B, Topping B H V (1999) Parallel simulated annealing for structural optimization. Computers & Structures. Vol.73(1-5), pp.545–564.

[26] Hasancebi O, Erbatur F (2002) On efficient use of simulated annealing in complex structural optimization problems. Acta mechanical. Vol.157(1-4), pp.27–50.

[27] De S, Hampton J, Maute K, Doostan A (2020) Topology optimization under uncertainty using a stochastic gradient-based approach. Structural and Multidisciplinary Optimization. Vol.62, pp.2255–2278.

[28] Holland J H (1992) Genetic Algorithms. Scientific American. Vol.267, pp.66–72.

[29] Poli R, Kennedy J, Blackwell T (2007) Particle swarm optimization: an overview. Swarm Intelligence Vol.1, pp.33–57.

[30] Deb K, Agrawal S, Pratap A, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. Vol.6, pp.182–197.

[31] Jenkins W M (1991) Towards structural optimization via the genetic algorithm. Computers & Structures. Vol.40(5), pp.1321–1327.

[32] Ohsaki M (1995) Genetic algorithm for topology optimization of trusses. Computers & Structures. Vol.57(2), pp.219–225.

[33] Kaveh A, Zolghadr A (2014) Democratic PSO for truss layout and size optimization with frequency constraints. Computers & Structures. Vol.130, pp.10–21.

[34] Sakai Y, Ohsaki M (2021) Optimization method for shape design of Auxetic Bending-Active Gridshells using discrete differential geometry. Structures. Vol.34, pp.1589–1602

[35] Vanluchene R D, Sun R (1990) Neural networks in structural engineering. Computer-Aided Civil Infrastructure Eng. Vol.5, pp.207–215.

[36] Sun H, Burton H V, Huang H (2021) Machine learning applications for building structural design and performance assessment: State-of-the-art review. Journal of Building Engineering, Vol.33(101816).

[37] Xie Y, Li S, Wu C T, Lyu D, Wang C, Zeng D (2022) A generalized Bayesian regularization network approach on characterization of geometric defects in lattice structures for topology optimization in preliminary design of 3D printing. Computational Mechanics. Vol.69, pp.1191–1212.

[38] Christian M C (2022) Machine learning in structural design: an opinionated review. Frontiers in Built Environment. Vol.8, pp.815717.

[39] Mirra G, Pugnale A (2021) Comparison between human-defined and ai-generated design spaces for the optimisation of shell structures. Structures. Vol.34, pp.2950–2961.

[40] Kingma P D, Welling M (2019) An introduction to variational autoencoders. Foundations and Trends in Machine Learning. Vol.12(4), pp.307–392.

[41] Samaniego E P, Anitescu C, Goswami S, Nguyen-Thanh V M, Guo H, Hamdia K M, Rabczuk T, Zhuang X (2020) An energy approach to the solution of partial differential equations in computational mechanics via machine learning: concepts, implementation and applications. Computer Methods in Applied Mechanics and Engineering. Vol.362(112790).

[42] Zheng H, Moosavi V, Akbarzadeh M (2020) Machine learning assisted evaluations in structural design and construction. Automation in Construction. Vol.119(103346).

[43] Fuhrimann L, Moosavi V, Ohlbrock P O, D'acunto P (2018) Data-driven design: exploring new structural forms using machine learning and graphic statics. In: Proceedings of *the International Association for Shell and Spatial Structures*. pp.1–8.

[44] Xie Y, Li S, Wu C T, Lyu D, Wang C, Zeng D (2022) A generalized Bayesian regularization network approach on characterization of geometric defects in lattice structures for topology optimization in preliminary design of 3D printing. Computational Mechanics. Vol.69, pp.1191–1212.

[45] Tamura T, Ohsaki M, Takagi J (2018) Machine learning for combinatorial optimization of brace placement of steel frames, Jpn. Architect. Rev., Vol.1, pp.419–430.

[46] Sakaguchi K, Ohsaki M, Kimura T (2021) Machine learning for extracting features of approximate optimal brace locations for steel frames, Frontiers in Built Environment. Vol.6(616455).

[47] Kupwiwat C, Yamamoto K (2020) Fundamental study on morphogenesis of shell structure using reinforcement. Journal of Structural Engineering B. Architectural Institute of Japan. Vol.67B, pp.211–218.

[48] Lee S, Kim H, Lieu Q X, Lee J (2020) CNN-based image recognition for topology optimization. Knowledge-Based Systems. Vol.198(105887).

[49] Hoyer S, Sohl-Dickstein J, Greydanus S (2019) Neural reparameterization improves structural optimization. In: *NeurIPS* 2019. Workshop on Solving Inverse Problems with Deep Networks.

[50] Hayashi K, Ohsaki M (2021) Reinforcement learning and graph embedding for binary truss topology optimization under stress and displacement constraints. Frontiers in Built Environment. Vol.6(59).

[51] Goodfellow I, Bengio Y, Courville A (2016) Deep Learning. The MIT Press.

[52] Mai H T, Kang J, Lee J (2021) A machine learning-based surrogate model for optimization of truss structures with geometrically nonlinear behavior. Finite Elements in Analysis and Design. Vol.196(103572).

[53] Li K, Yu Y, He J, Zhao D, Lin Y (2018) Structural optimization of Hatch cover based on bi-directional evolutionary structure optimization and surrogate model method. Journal of Shanghai Jiaotong University (Science). Vol.23, pp.538–549.

[54] Luo D, Huang J, Su G, Tao H (2023) A dynamic Gaussian process surrogate model-assisted particle swarm optimisation algorithm for expensive structural optimisation problems. European Journal of Environmental and Civil Engineering. Vol.27(1), pp.416–436.

[55] Chen M, Shi X, Zhang Y, Wu D, Guizani M (2017) Deep feature learning for medical image analysis with convolutional autoencoder neural network. IEEE Transactions on Big Data. Vol.7(4), pp.750–758.

[56] Palmeri M, Málaga-Chuquitaype C, Kampas G, Memarzadeh M (2021) AI-based Optimization of Off-Earth Habitat Structures. In: *Tech. rep., "Tehcnical Report 21/03", Emerging Structural Technologies*. London, United Kingdom: Imperial College London.

[57] Cheung A, Cabrera C, Sarabandi P, Nair K K, Kiremidjian A, Wenzel H (2008) The application of statistical pattern recognition methods for damage detection to field data. Smart materials and structures. Vol.17(6).

[58] Ng C T (2014) On the selection of advanced signal processing techniques for guided wave damage identification using a statistical approach. Engineering Structures. Vol.67, pp.50–60.

[59] Hwang J H, Joo B C, Yoo Y J, Park K T, Lee C H (2013) Damage detection of a prototype building structure under shaking table testing using outlier analysis. In: *Health Monitoring of Structural and Biological Systems 2013*. Vol.8695, pp.952–957.

[60] Tibaduiza D A, Mujica L E, Rodellar J, Güemes A (2016) Structural damage detection using principal component analysis and damage indices. Journal of Intelligent Material Systems and Structures. Vol.27(2), pp.233–248.

[61] Box G E, Jenkins G M, Reinsel G C, Ljung G M (2015) Time series analysis: forecasting and control. John Wiley & Sons.

[62] Barber D (2011) Bayesian Reasoning and Machine Learning. Cambridge University Press.

[63] Pearson K (1901) On lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin philosophical magazine and journal of science, Vol.2(11), pp.559–572.

[64] Wold S, Esbensen K, Geladi P (1987) Principal component analysis. Chemometrics and intelligent laboratory systems. Vol.2(1-3), pp.37–52.

[65] Sutton R S, Barto A G (2018) Reinforcement learning, an introduction. 2nd ed The MIT Press, Cambridge MA.

[66] Bellman R (1956) A problem in the sequential design of experiments. Sankhya. Vol.16, pp.221–229.

[67] Bellman R, Dreyfus S E (1959) Functional approximations and dynamic programming. Mathematical Tables and Other Aids to Computation. Vol.13, pp.247–251.

[68] Bellman R (1954) The theory of dynamic programming. Bulletin of the American Mathematical Society. Vol.60(6), pp.503–515.

[69] Bellman R (1957) A Markovian decision process. Journal of Mathematics and Mechanics. pp.679–684.

[70] Watkins C J C H, Dayan P (1992) Q-learning. Machine Learning. Vol.8, pp.279–292.

[71] Richard S, David M, Satinder S, Yishay M (1999) Policy Gradient Methods for Reinforcement Learning with Function Approximation. Vol.12, pp.1057–1063.

[72] Lillicrap T P, Hunt J J, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D (2016) Continuous control with deep reinforcement learning. In: Proceedings of *the International Conference on Learning Representations* (Poster).

[73] Konda V, Tsitsiklis J (1999) Actor-critic algorithms. Advances in neural information processing systems. Vol.12, pp.1008-1014.

[74] Haarnoja T, Zhou A, Abbeel P, Levine S (2018) Soft actor-critic: off policy maximum entropy deep reinforcement learning with a stochastic actor. In: Proceedings of *the 35th International Conference on Machine Learning*, pp.1861–1870.

[75] Kiefer J, Wolfowitz J (1952) Stochastic estimation of the maximum of a regression function. The Annals of Mathematical Statistics. Vol.23(3), pp.462–466.

[76] Sebastian R (2016) An overview of gradient descent optimization algorithms. arXiv:1609.04747.

[77] Kingma D, Ba J (2015) Adam: a method for stochastic optimization. In: Proceedings of *the 3rd International Conference on Learning Representations*, pp.1–15

[78] Uhlenbeck G E, Ornstein L S (1930) On the theory of the brownian motion. Physical Review. Vol.36(5), pp.823–841.

[79] Gambardella L M, Dorigo M (1995) Ant-Q: A reinforcement learning approach to the traveling salesman problem. In: *Machine Learning Proceedings 1995*. pp.252–260.

[80] Huynh T N, Do D T, Lee J (2021) Q-Learning-based parameter control in differential evolution for structural optimization. Applied Soft Computing. Vol.107(107464).

[81] Littman M L (1994) Markov Games as a Framework for Multi-Agent Reinforcement Learning. International Conference on Machine Learning. Vol.157, pp.157-163.

[82] Tesauro G (2004) Extending q-learning to general adaptive multi-agent systems. In: *Advances in Neural Information Processing Systems*, pp.871–878.

[83] Foerster J, Farquhar G, Afouras T, Nardelli N, Whiteson S (2018) Counterfactual multi-agent policy gradients. In: Proceedings of *the AAAI Conference on Artificial Intelligence*. Vol.32(1).

[84] Nasir Y S, Guo D (2019) Multi-agent deep reinforcement learning for dynamic power allocation in wireless networks. IEEE Journal on Selected Areas in Communications. Vol.37(10), pp.2239–2250.

[85] Yu L, Sun Y, Xu Z, Shen C, Yue D, Jiang T, Guan X (2020) Multi-agent deep reinforcement learning for HVAC control in commercial buildings. IEEE Transactions on Smart Grid. Vol.12(1), pp.407–419.

[86] Lowe R, Wu Y I, Tamar A, Harb J, Pieter Abbeel O, Mordatch I (2017) Multi-agent actor-critic for mixed cooperative-competitive environments. Advances in neural information processing systems. Vol.30.

[87] Huynh T N, Do D T, Lee J (2021) Q-Learning-based parameter control in differential evolution for structural optimization. Applied Soft Computing. Vol.107(107464).

[88] Hayashi K, Ohsaki M (2021) Reinforcement learning for optimum design of a plane frame under static loads. Engineering with Computers. Vol.37(3), pp.1999–2011.

[89] Sahachaisaree S, Sae-Ma P, Nanakorn P (2020) Two-dimensional truss topology design by reinforcement learning. In: Proceedings of *the International Conference on Sustainable Civil Engineering and Architecture (ICSCEA 2019).* pp.1237–1245.

[90] Kupwiwat C, Yamamoto K (2020) マルチエージェント強化学習を用いた構造形態創生に関する基礎的研. In: *コロキウム構造形態の解析と創生 2020 講演論文集.* pp.65–70. (in Japanese)

[91] Shea K (1997) Essays of discrete structures: purposeful design of grammatical structures by directed stochastic search. Carnegie Mellon University.

[92] Kaveh A, Koohestani K (2008) Graph products for configuration processing of space structures. Computers & structures. Vol.86(11–12), pp.1219–1231.

[93] Hooshmand A, Campbell M I (2016) Truss layout design and optimization using a generative synthesis approach. Computers & Structures. Vol.163, pp.1–28.

[94] Van Mele T, Block P (2014) Algebraic graph statics. Computer-Aided Design. Vol.53, pp.104–116.

[95] D'acunto P, Jasienski J P, Ohlbrock P O, Fivet C, Schwartz J, Zastavni D (2019) Vector-based 3D graphic statics: A framework for the design of spatial structures based on the relation between form and forces. International Journal of Solids and Structures. Vol.167, pp.58–70.

[96] Zanni G, Pennock G R (2009) A unified graphical approach to the static analysis of axially loaded structures. Mechanism and machine theory. Vol.44(12), pp.2187–2203.

[97] Abualdenien J, Borrmann A (2021) PBG: A parametric building graph capturing and transferring detailing patterns of building models. In: Proceedings of *the CIB W78 Conference 2021*.

[98] Lipp M, Wonka P, Wimmer M (2008) Interactive visual editing of grammars for procedural architecture. In: Proceedings of *the ACM SIGGRAPH 2008*. pp.1–10.

[99] Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. Psycological review. Vol.65(6), pp.386–408.

[100] Ivakhnenko A G (1968) The group method of data handling – a rival of the of stochastic approximation, Soviet Automatic Control, Vol.13(3), pp.43–55.

[101] Cybenko G (1989) Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals, and Systems. Vol.2(4), pp.303–314.

[102] Verhulst P F (1845) Mathematical researches into the law of population growth increase. Nouveaux Mémoires de l'Académie Royale des Sciences et Belles-Lettres de Bruxelles. Vol.18, pp.1–42.

[103] Nair V, Hinton G E (2010) Rectified linear units improve restricted boltzmann machines. In: Proceedings of *the International Conference on Machine Learning*, pp.807–814

[104] Maas A L, Hannun A Y, Ng A Y (2013) Rectifier nonlinearities improve neural network acoustic models. In: Proceedings of *the International Conference on Machine Learning*. Vol.30(1).

[105] Osborn G (1902) Mnemonic for hyperbolic formulae. The Mathematical Gazette. Vol.2(34), pp.189–189.

[106] Durbin R, Rumelhart D E (1989) Product units: A computationally powerful and biologically plausible extension to backpropagation networks. Neural computation. Vol.1(1), pp.133–142.

[107] Polyak B T (1964) Some methods of speeding up the convergence of iteration methods, USSR Computational Mathematics and Mathematical Physics. Vol.4(5), pp.1–17.

[108] Hinton G (2012) Neural networks for machine learning. Coursera, video lectures.

[109] LeCun Y, et al. (1995) Comparison of learning algorithms for handwritten digit recognition. In: Proceedings of *the International Conference on Artificial Neural Networks*. Vol.10(1).

[110] Kipf T N, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: Proceedings of *the International Conference on Learning Representations*. pp.1–14.

[111] Veličković P, Cucurull G, Casanova A, Romero A, Lio P, Bengio Y (2017) Graph Attention Networks. In: Proceedings of *the International Conference on Learning Representations*. pp.1–12.

[112] Hu F, Zhu Y, Wu S, Wang L, Tan T (2019) Hierarchical graph convolutional networks for semi-supervised node classification. In: Proceedings of *the 28th International Joint Conference on Artificial Intelligence*. pp.4532–4539.

[113] Li Z, Liu Z, Huang J, Tang G, Duan Y, Zhang Z, Yang Y (2019) MV-GCN: multi-view graph convolutional networks for link prediction. IEEE Access. Vol.7, pp.176317–176328.

[114] Xie Y, Yao C, Gong M, Chen C, Qin A K (2020) Graph convolutional networks with multi-level coarsening for graph classification. Knowledge-Based Systems. Vol.194(105578).

[115] Yu Z, Wang H, Liu Y, Böhm C, Shao J (2020) Community attention network for semi-supervised node classification. In: Proceedings of *the IEEE International Conference on Data Mining*. Vol.17(20), pp.1382–1387.

[116] Huang J, Shen H, Hou L, Cheng X (2019) Signed Graph Attention Networks. In: Proceedings of *the International Conference on Artificial Neural Network*. Vol.28, pp.566–577.

[117] Gao J, Gao J, Ying X, Lu M, Wang J (2021) Higher-order interaction goes neural: a substructure assembling graph attention network for graph classification. IEEE transactions on knowledge and data engineering. Vol.35(2), pp.1594–1608

[118] Aich S, Stavness I (2019) Global sum pooling: a generalization trick for object counting with small datasets of large images. In: Proceedings of *the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp.73–82

[119] Kao C Y, Loh C H (2013) Monitoring of long-term static deformation data of Fei-Tsui arch dam using artificial neural network-based approaches. Structural Control and Health Monitoring. Vol.20(3), pp.282–303.

[120] Karina C N, Chun P J, Okubo K (2017) Tensile strength prediction of corroded steel plates by using machine learning approach. Steel Compos. Struct. Vol.24(5), pp.635–641.

[121] Lee S, Lee C (2014) Prediction of shear strength of FRP-reinforced concrete flexural members without stirrups using artificial neural networks. Engineering structures. Vol.61, pp.99–112.

[122] Chatterjee S, Sarkar S, Hore S, Dey N, Ashour A S, Shi F, Le D N (2017) Structural failure classification for reinforced concrete buildings using trained neural network based multi-objective genetic algorithm. Structural Engineering and Mechanics. Vol.63(4), pp.429–438.

[123] Li Z, Burgueño R (2010) Using soft computing to analyze inspection results for bridge evaluation and management. Journal of Bridge Engineering. Vol.15(4), pp.430–438.

[124] Radhika S, Tamura Y, Matsui M (2015) Cyclone damage detection on building structures from pre-and post-satellite images using wavelet based pattern recognition. Journal of Wind Engineering and Industrial Aerodynamics. Vol.136, pp.23–33.

[125] García-Segura T, Yepes V, Frangopol D M (2017) Multi-objective design of post-tensioned concrete road bridges using artificial neural networks. Structural and Multidisciplinary Optimization. Vol.56, pp.139–150.

[126] Papadrakakis M, Lagaros N D (2002) Reliability-based structural optimization using neural networks and Monte Carlo simulation. Computer methods in applied mechanics and engineering. Vol.191(32), pp.3491–3507.

[127] Lagaros N D, Papadrakakis M, Plevris V (2005) Multiobjective optimization of space structures under static and seismic loading conditions. In: Evolutionary Multiobjective Optimization: Theoretical Advances and Applications. London: Springer London. pp.273–300.

[128] Fortin F A, De Rainville F M, Gardner M A, Parizeau M, Gagné C (2012) DEAP: Evolutionary algorithms made easy. Journal of Machine Learning Research. Vol.13(1), pp.2171–2175.

[129] Virtanen P, Gommers R, Oliphant T E, et al. (2020) SciPy 1.0: fundamental algorithms for scientific computing in Python. Nature Methods. Vol.17, pp.261–272.

[130] Kimura T, Ohmori H (2008) Computational morphogenesis of free form shells. Journal of the International Association for Shell and Spatial Structures. Vol.49(3), pp.175–180.

[131] Vu-Bac N, Duong T X, Lahmer T, Zhuang X, Sauer R A, Parke H S, Rabczuk T (2018) A NURBS-based inverse analysis for reconstruction of nonlinear deformations of thin shell structures. Computer Methods in Applied Mechanics and Engineering. Vol.331, pp.427–455.

[132] Bernstein S (1912) Démonstration du théorème de weierstrass fondée sur le calcul des probabilités. Comm Soc Math Kharkow. Vol.13, pp.1–2.

[133] Ramm E, Bletzinger K U, Reitinger R (1993) Shape optimization of shell structures. Revue Européenne des Éléments Finis. Vol.2(3), pp.377–398.

[134] Roulier J A, Rondo T (1994) Measures of fairness for curves and surfaces. In: *N S Spadis (Ed) Designing fair curves and surfaces*. SIAM, pp.75–122.

[135] Ohsaki M, Nakamura T, Kohiyama M (1997) Shape optimization of a double-layer space truss described by a parametric surface. International Journal of Space Structures. Vol.12, pp.109–119.

[136] Ohsaki M, Hayashi M (2000) Fairness metrics for shape optimization of ribbed shells. Journal of the International Association for Shell and Spatial Structures. Vol.41(1), pp.31–39.

[137] Deb K (2001) Multi-objective optimization using evolutionary algorithms. Wiley-Interscience series in systems and optimization.

[138] Wolpert D H, Wheeler K R, Tumer K (2000) Collective intelligence for control of distributed dynamical systems. Europhysics Letters. Vol.49(6), pp.708.

[139] Mannion P, Devlin S, Duggan J, Howley E (2016) Avoiding the tragedy of the commons using reward shaping. In: Proceedings of *the Adaptive and Learning Agents Workshop* (at AAMAS 2016).

[140] Tumer K, Agogino A (2007) Distributed agent-based air traffic flow management. In: Proceedings of the *6th International Joint Conference on Autonomous Agents and Multiagent Systems*. pp.1–8.

[141] Wolpert D H, Tumer K (2002) Collective intelligence, data routing and Braess' paradox. Journal of Artificial Intelligence Research. Vol.16, pp.359–387.

[142] Zitzler E, Deb K, Thiele L (2000) Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. Evolutionary Computation. Vol.8, pp.173–195.

[143] Kumar S, Tejani G G, Pholdee N, et al. (2021) Multi-objective modified heat transfer search for truss optimization. Engineering with Computers. Vol.37, pp.3439–3454.

[144] Langenhan C, Weber M, Liwicki M, Petzold F, Dengel A (2013) Graph-based retrieval of building information models for supporting the early design stages. Advanced Engineering Informatics. Vol.27(4), pp.413–426.

[145] Vestartas P (2021) Design-to-fabrication workflow for raw-sawn-timber using joinery solver. Lausanne Switzerland: EPFL. Doctoral dissertation, Ph. D. thesis. doi: 10.5075/EPFLTHESIS-8928.

[146] McNeel R, et al. (2010) Rhinoceros 3D (Version 6.0). Available at: https://www.rhino3d.com (Accessed: November 29, 2023)

[147] Autodesk (2020) Autodesk Revit. Available at: https://www.autodesk.com (Accessed: November 29, 2023)

Publications

Peer-reviewed articles associated with this dissertation

[1] Kupwiwat C, Hayashi K, Ohsaki M (2022) Deep deterministic policy gradient and graph convolutional network for bracing direction optimization of grid shells. Frontiers in Built Environment, Vol.8(899072).

[2] Kupwiwat C, Iwagoe Y, Hayashi K, Ohsaki M (2023) Deep deterministic policy gradient and graph convolutional network for topology optimization of braced steel frames. Journal of Structural Engineering B. Architectural Institute of Japan. Vol.69B, pp.129–139.

[3] Kupwiwat C, Hayashi K, Ohsaki M (2023) Deep deterministic policy gradient and graph attention network for geometry optimization of latticed shells. Applied Intelligence. Vol.53, pp.19809–19826.

[4] Kupwiwat C, Hayashi K, Ohsaki M (2024) Multi-objective optimization of truss structure using multi-agent reinforcement learning and graph representation. Engineering Applications of Artificial Intelligence. Vol.129(107594), no pages.

Oral presentations associated with this dissertation

[1] Kupwiwat C, Hayashi K, Ohsaki M (2023) A multi-agent reinforcement learning framework for bi-objective optimization. In. the 33th Design and Systems Conference (D&S2023), The Japan Society of Mechanical Engineers. No.C000074.

[2] Kupwiwat C, Hayashi K, Ohsaki M (2023) Multi-objective optimization of 10-bar truss using multi-agent reinforcement learning. In. Annual Convention of Architectural Institute of Japan (AIJ), Architectural Institute of Japan. No.20134.

[3] Kupwiwat C, Hayashi K, Ohsaki M (2023) Sizing optimization of free-form lattice shells using deep deterministic policy gradient and graph convolutional networks. In. Proceeding of International Association for Shell and Spatial Structures 2023. pp. 1458–1468.

[4] Kupwiwat C, Hayashi K, Ohsaki M (2022) Geometry optimization of lattice shells using GAT-DDPG with Bézier surface. In. Annual Convention of Architectural Institute of Japan (AIJ), Architectural Institute of Japan. No.20412.

[5] Kupwiwat C, Hayashi K, Ohsaki M (2022) Deep deterministic policy gradient and graph convolutional network for geometry and topology optimization of braced latticed shells. In. Annual Convention of Architectural Institute of Japan (AIJ) Kinki Branch, Architectural Institute of Japan (AIJ). No.10096.

[6] Kupwiwat C, Hayashi K, Ohsaki M (2022) Topology optimization of braced latticed shells using deep deterministic policy gradient and graph convolutional network. In. Asian Congress of Structural and Multidisciplinary Optimization (ACSMO), Asian Society for Structural and Multidisciplinary Optimization (ASSMO). No.acsmo2022-p0005.

Acknowledgments

This dissertation was done when I was a doctoral student at Ohsaki & Zhang Laboratory, the Department of Architecture and Architectural Engineering, Kyoto University. I gratefully acknowledge the financial support over the years from the Japanese Ministry of Education, Culture, Sports, Science and Technology (MEXT) for the Monbukagakusho Scholarship (文部科学省奨学金).

Above all, I would like to express my deepest gratitude to my academic advisor, Professor Makoto Ohsaki, for his guidance, support, and encouragement. His expertise in structural optimization and support encourages me to endeavor knowledge to shape my understanding and give me academic freedom to find my inclinations. I also would like to express my gratitude to Assistant Professor Kazuki Hayashi, for his guidance. His expertise in machine learning gives me many inspirations for solving problems. Both of them also taught me how to carefully prepare for an academic career, starting by constructing a solid foundation of knowledge.

I would like to thank the committee members, Professor Minehiro Nishiyama and Associate Professor Kohei Fujita, for spending their precious time reading, discussing, and providing constructive feedback on the contents of my dissertation and my presentation.

I would like to thank Associate Professor Jingyao Zhang and Assistant Professor Kentaro Hayakawa for their discussions on essential aspects of structural optimization. I am grateful to my Master's advisor, Professor Yamamoto Kenji at Tokai University, for guiding me through the very first steps of how to conduct a research project properly.

I also want to thank the former and current members of the Ohsaki & Zhang Laboratory, especially Yuichi Iwagoe for providing valuable research material in Chapter 4, Dr. Yusuke Sakai, Dr. Wei Shen, and Dr. Shaojun Zhu for their discussion and feedback about the approaches I proposed in this dissertation, to Dr.Do Kim Bach for his suggestion about the dissertation submission process. I thank Safumi Saiki for her administrative assistance, which has helped me to focus on my research over the years.

Three years of my life in Kyoto are full of great experiences and enjoyable dinners, thanks to Chitipat Sornsukolrat, Satanan Phattanaprayoonvong, Anan Nilrattanakhun, Soulida Louangphachaleun, Jakkraphan Phetharn, and Chanya Wisalrakij.

Finally, my heartfelt thanks go to my parents, my younger sisters, my grandma, and my dear girlfriend Ngamnij Prakasitanonda for their supportive love, understanding, and calls over the years. I dedicate this dissertation to them.