Submitted to Applied Intelligence

Deep deterministic policy gradient and graph attention network for geometry optimization of latticed shells

Chi-tathon KUPWIWAT [1, *], **Kazuki HAYASHI** [1] **and Makoto OHSAKI** [1]

[1] *Department of Architecture and Architectural Engineering, Graduate School of Engineering, Kyoto University, Kyoto, Japan*
[*] *Corresponding author. Email address*: kupwiwat.chitathon.73c@st.kyoto-u.ac.jp

Abstract
This paper proposes a combined approach of deep deterministic policy gradient (DDPG) and graph attention network (GAT) to the geometry optimization of latticed shells with surface shapes defined by a Bézier control net. The optimization problem is formulated to minimize the strain energy of the latticed structures with heights of the Bézier control points as design variables. The information of the latticed shells, including nodal configurations, element properties and internal forces, and the Bézier control net, consisting of control points and control net, are represented as graphs using node feature matrices, adjacency matrices, and weighted adjacency matrices. A specifically designed DDPG agent utilizes GAT and matrix manipulations to observe the state of the structure through the graphs, and decides which and how Bézier control points to move. The agent is trained to excel in the task through a reward signal computed from changes in the strain energy in each optimization step. As shown in numerical examples, the trained agent can effectively optimize structures of different sizes, control nets, configurations, and initial geometries from those used during the training. The performance of the trained agent is competitive compared to particle swarm optimization and simulated annealing despite using a lower computational cost.

Article highlights
- A method using a reinforcement learning agent is proposed to optimize the geometry of latticed structures.
- The agent is designed to observe the structure and Bézier control net and modify the Bézier control net.
- The method yields good results using fewer computations when compared to other conventional methods.

Keywords: Bézier surface, Deep deterministic policy gradient, Geometry optimization, Graph attention network, Reinforcement learning

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

Acknowledgments
Japan ministry of education, culture, sports, science, and technology (MEXT) scholarship [grant number 180136] and Japan society for the promotion of science (JSPS) KAKENHI [grant numbers JP 20H04467, JP 21K20461] helped fund this research.

1. Introduction

Structural optimization is a branch of optimization that aims to find the best design variables that minimize/maximize the objective function, which is related to structural cost and/or performance under design constraints [1]. For discrete structures, such as trusses and frames, structural optimization can be classified as geometry and topology optimizations where nodal locations and nodal connectivity, respectively, are design variables [2]. Structural optimization is important for the initial design process of latticed shells, which cover a large space with a number of truss and frame elements, because their structural performances are closely related to their shape and topology [3]. An optimization problem of latticed shells is generally formulated to maximize the stiffness against static loads within a given structural volume, and the compliance or the strain energy is often selected as the objective function to be minimized. Examples of such formulations can be found in Refs. [4–6].

Availability of the method for geometry optimization of latticed shells depends on whether the design variables are continuous or discrete. If the nodal locations are continuous variables, the optimization problem is classified as a nonlinear programming problem that can be solved using a gradient-based approach together with shape sensitivity analysis [7–10]. If the nodal locations are selected from a list of pre-defined discrete values, the optimization problem is considered an integer programming or combinatorial problem, where gradient information is not available, and the

solutions can be obtained using heuristic methods such as genetic algorithm (GA) and simulated annealing (SA) [11–13].

Parametric surfaces are effective for representing the structural nodal height to obtain a smooth shape of latticed shells. The surfaces are utilized to reduce the number of design variables in the optimization problem [14] and analyze the shell structures [15]. The type of parametric surface that has most attracted researchers is the Bézier surface which defines the shape with a tensor product of the Bernstein polynomials [16] and their control points. Pioneering research on shape optimization of shells using Bézier surface can be found in Ref. [17] and is further refined in later years [18–20].

In the past decade, machine learning (ML) approaches have been extensively studied for application to engineering problems. With the advent of computational models such as neural network (NN) [21-23], a number of applications of ML can be found in engineering fields [24-26]. Particularly, ML for optimization of spatial structures has attracted many researchers [27]. Mirra and Pugnele [28] used variational autoencoder (VAE) [29] for designing spatial structures. Samaniego et al. [30] proposed an ML method to approximate the mechanical response of plate and shell structures. Zheng et al. [31] and Fuhrimann et al. [32] used ML models to explore design spaces of spatial structures. Xie et al. [33] proposed a Bayesian Network to assess the deviation between constructed shapes and designed shapes of the 3D printed lattice structures.

Reinforcement learning (RL) is a type of ML where a model or agent learns to decide its interactions with an environment. The agent improves its policy which dictates how it makes decisions according to the given reward signal determined by how the environment changes. Deep deterministic policy gradient (DDPG) [34] is a type of RL algorithm that can handle multiple actions at the same time. When applied to optimization problems, DDPG could reduce the number of optimization steps, and accordingly, save the computational cost. Kupwiwat and Yamamoto [35] demonstrated the applicability of DDPG to geometry optimization of latticed shell structures using NNs as agents. However, their method of having an agent observe and do action by moving through nodes of the entire structure still requires large numbers of optimization steps for large latticed shells.

Hayashi and Ohsaki [36] proposed a method for the binary topology optimization problem of planar trusses using RL with a graph representation of the structure. Their implementation allows the agent to observe the entire truss through transmitted graph signals of structural nodes and elements and modify any truss element in the structure. In later years, Zhu et al. [37] applied this method to the stochastic generation of truss topology. Kupwiwat et al. [38] presented a combined approach of DDPG and graph representation for optimizing topologies of grid shells that can be applied to large structures and obtain competitive results compared to GA. It is worth noting that the graph representation is utilized only for the structural domain in all research mentioned above.

NN architectures that are designed for processing graph data are called graph neural network (GNN). Recently, GNNs have gained attention in the research field of ML because various data, such as protein structure, texts, images, and social network data, can be expressed in the form of graphs. To effectively process graph data, some variances of GNNs have been proposed such as message passing NN [39] and graph convolutional network (GCN) [40]. Graph attention network (GAT) [41] is also a type of GNNs that utilizes an attention mechanism used in the transformer architecture [42], which can indicate importance of the adjacent nodes in the graph. GAT is applicable in various domains including node classification [43], link prediction [44], and graph classification [45].

This paper proposes a geometry optimization method for latticed shell structures using DDPG and GAT together with a Bézier surface for minimizing the strain energy. The proposed RL agent is trained to observe the structure through graph representations of both latticed structure and Bézier control net, and adjust the heights of Bézier control points from a flat configuration. The remainder of this paper is structured as follows. Section 2 formulates the geometry optimization problem. In Section 3, RL and DDPG are briefly explained. GAT and the definitions of vectors and matrices in the proposed GAT-DDPG method are introduced in Section 4. Section 5 presents operational details of GAT-DDPG to integrate the latticed structure and its Bézier control net, and train the agent for solving the optimization problem. In Section 6, numerical examples are presented, where the results and computational cost of the proposed method are compared with those by particle swarm optimization (PSO) and SA. The robustness of the agent, when applied to latticed shells with different sizes, boundary conditions, and optimization settings, is also investigated in this section.

The main contributions of this paper are combining GAT and DDPG for optimizing the geometry of latticed shells and combining topological data from Bézier and structural domains to be observed by the RL agent. The proposed method is applicable to the geometry optimization to minimize the strain energy of latticed shells with various numbers of grids. The design variable is the heights of Bézier control points which can be either discrete or continuous values. The numbers of grids in the Bézier control net and the structure can be separately specified and fix-supports can be arbitrarily positioned at the rims of the structures. The method generally yields competitive results while requiring lower computational costs than other methods.

2. Optimization problems of latticed shells

2.1 Objective function

Consider a problem of minimizing the strain energy of a latticed shell stiffened by diagonal braces subjected to static loads where the boundary conditions are predetermined for each structural node. Three-dimensional beam elements with 12 degrees of freedom (DoFs) and three-dimensional truss elements with 6 DoFs are used to represent the lattice frame and brace, respectively. In this paper, structural stiffness matrix analysis is utilized to compute the total strain energy where the stiffness matrices of the frame element and the brace element in the local coordinate system are denoted as $\mathbf{k}_\mathrm{f} \in \mathbb{R}^{12\times12}$ and $\mathbf{k}_\mathrm{e} \in \mathbb{R}^{6\times6}$, respectively. Modified into the global coordinate system, these matrices are congregated as the global stiffness matrix $\mathbf{K} \in \mathbb{R}^{n_\mathrm{D}\times n_\mathrm{D}}$, where n_D denotes the total number of DoFs, after specifying the boundary conditions, of the structure. Let $\mathbf{p} \in \mathbb{R}^{n_\mathrm{D}}$ be the nodal load vector in the global coordinate system obtained from aggregating point loads and weights applied on every node and element, respectively. The nodal displacement vector $\mathbf{d} \in \mathbb{R}^{n_\mathrm{D}}$ is computed by solving the stiffness equation as follows:

$$\mathbf{K}\mathbf{d} = \mathbf{p} \tag{1}$$

The transpose of a vector or a matrix is denoted by the superscript T. The total strain energy E is derived from

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

2.2 Bézier surface

Geometry of the latticed shell as shown in Fig. 1 can be defined using a Bézier control net and Bernstein basis functions. The Bernstein basis function of order n is given as

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}, (i = 0,1,\dots,n) \tag{3a}$$

$$\text{where } \binom{n}{i} = \begin{cases} \frac{n!}{i!(n-i)!} & \text{for } 0 \le i \le n \\ 0 & \text{for } i < 0 \text{ or } i > n \end{cases} \tag{3b}$$

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

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

$$\mathbf{B}_\mathrm{b}^{c_n,c_m}(u,v) = \sum_{i=0}^{c_n}\sum_{j=0}^{c_m} \mathbf{R}_{i,j}\, B_i^{c_n}(u) B_j^{c_m}(v) \tag{4}$$

where $\mathbf{R}_{i,j} (i = 1,..,c_n; j = 1,..,c_m)$ are the coordinates of control points, i.e., the coordinates of the vertices in the control net, and c_n and c_m are the orders of basis functions in u and v directions, respectively.

Consider a latticed shell with a size of $a \times b$ located in the region bounded by $0 \le x \le a$ and $0 \le y \le b$ on the xy-plane as shown in Fig. 1A. The horizontal coordinates of $\mathbf{R}_{i,j}$ are assigned at a uniform spacing to construct a $(c_n - 1) \times (c_m - 1)$-grid control net as shown in Fig. 1B. Thus, by further assigning u and v values of the nodes at the uniformly spaced grid points on the (u, v) parameter plane in the range $[0,1] \times [0,1]$, the structural nodes are uniformly spaced in the $a \times b$ region on the xy-plane, as shown in Fig. 1C. The nodal location vector of the ith node corresponding to the parameters (u_i, v_i) is computed as $\mathbf{B}_\mathrm{b}^{c_n,c_m}(u_i, v_i)$ in Eq. (4) as shown in Fig. 1D.

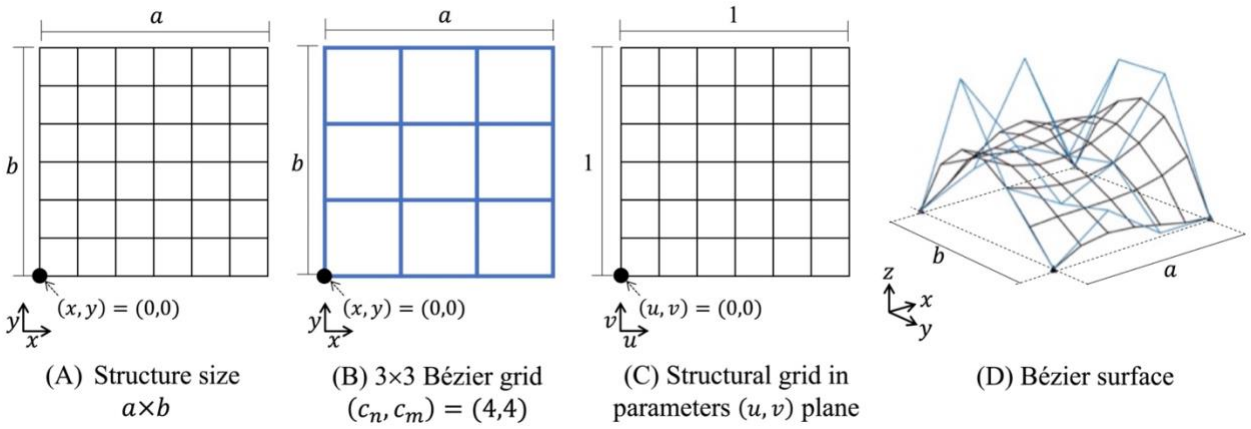

Fig. 1 Bézier surface of the latticed shell defined by Bézier control net.

By fixing the (x, y) coordinates (1st and 2nd components of $\mathbf{R}_{i,j}$) of the control points, the z-coordinates (3rd components of $\mathbf{R}_{i,j}$) of Bézier control net determines the nodal heights of the latticed shell which consists of frame elements with 12 DoFs in the main axes (i.e., x and y axes in Fig.1) 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 that of the latticed structure are independent of each other and can be determined separately based on the preferences of designers and/or engineers. The examples of the proposed method with a different discretization of the Bézier control nets and structural grids are shown in Section 6.3.1.

2.3 Geometry optimization problem

Geometry of a latticed shell with n nodes and n_{b} control points shall be optimized by changing the heights of Bézier control points. Let $\mathbf{z} = (z_1, z_2, \dots, z_{n^{\mathrm{free}}})$ and $\mathbf{z}^{\mathrm{b}} = (z_1^{\mathrm{b}}, z_2^{\mathrm{b}}, \dots, z_{n_{\mathrm{b}}^{\mathrm{free}}}^{\mathrm{b}})$ denote the vector of nodal heights and the vector of movable heights of Bézier control points, while the heights of supports are fixed. When the values of parameters (u, v) are assigned to each node, the vector of nodal heights is computed from the heights of control points as $\mathbf{z}(\mathbf{z}^{\mathrm{b}})$. Note that $\mathbf{z}$ is derived from $\mathbf{z}^{\mathrm{b}}$ and this paper considers only $\mathbf{z}^{\mathrm{b}}$ as the design variable of the optimization problem. Let $z_{\max}$, $z_{\min}$, $z_{\min}^{\mathrm{b}}$, and $z_{\max}^{\mathrm{b}}$ 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 deformation are regarded as functions of $\mathbf{z}^{\mathrm{b}}$ as $\mathbf{K}(\mathbf{z}^{\mathrm{b}})$ and $\mathbf{d}(\mathbf{z}^{\mathrm{b}})$, respectively. Then, the geometry optimization problem to maximize the stiffness of the structure, or to minimize the total strain energy, is as follows:

$$\text{minimize} \quad E(\mathbf{z}^{\mathrm{b}}) = \frac{1}{2}\mathbf{d}(\mathbf{z}^{\mathrm{b}})^{\mathrm{T}}\mathbf{K}(\mathbf{z}^{\mathrm{b}})\mathbf{d}(\mathbf{z}^{\mathrm{b}}) \tag{5a}$$

$$\text{subject to} \quad z_{\min} \le z_i(\mathbf{z}^{\mathrm{b}}) \le z_{\max} \quad (i = 1, 2, \dots, n^{\mathrm{free}}) \tag{5b}$$

$$z_{\min}^{\mathrm{b}} \le z_j^{\mathrm{b}} \le z_{\max}^{\mathrm{b}} \quad (j = 1, 2, \dots, n_{\mathrm{b}}^{\mathrm{free}}) \tag{5c}$$

3. Reinforcement learning and deep deterministic policy gradient

This research proposes a method to train an RL agent for geometry optimization of latticed shells. RL consists of a *policy function* to determine how the agent makes actions, a *reward signal* to quantitatively evaluate the consequence of the actions, and a *value function* to predict the accumulated reward signal, respectively [46]. The *environment* refers to the system that the RL agent is taught in. Markov decision process (MDP) is used to model the interaction between an agent and its environment [47,48].

Fig. 2 illustrates the MDP used in this research. In a discrete step t, the agent observes a state S_t as graphs representing the structure and the Bézier control net (Fig. 2A), and takes an action A_t to adjust the Bézier control points (Fig. 2B). The Bézier control net is changed by actions of the agent and the structural geometry is changed accordingly (Fig. 2C). The agent receives a reward R_{t+1} which is computed from the strain energy of the current geometry and the geometry at the previous step. The agent observes the next state S_{t+1}. Note that the RL agent keeps interacting with the environment until the termination criterion is satisfied. This paper uses a predefined number of optimization steps as a termination criterion.

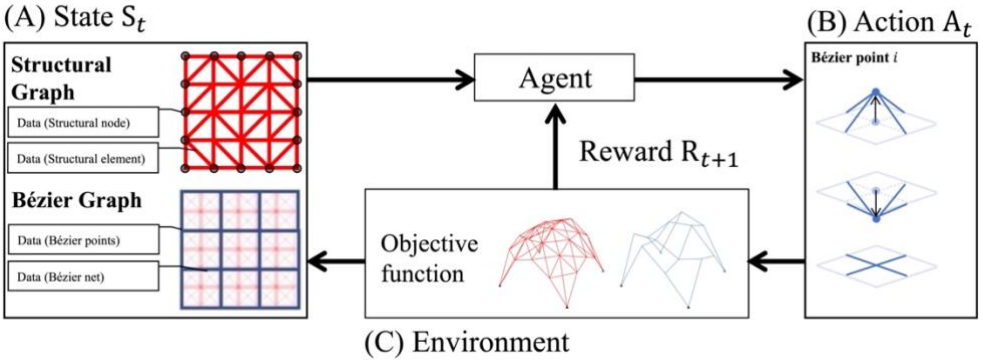

Fig. 2 Proposed MDP for geometry optimization of latticed shells with Bézier surface

DDPG is a type of RL algorithm, characterized by ability of the agent to do multiple actions in an MDP which can reduce the number of optimization steps and computational analyses 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. The objective of the policy function is to obtain high reward by determining the probability $P(A_t^i|S_t)$ of taking an action i in a state S_t

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

$$Q_{\theta_2}\left(S_t, \boldsymbol{\pi}_{\theta_1}(S_t)\right) = \sum_{v=1}^{\infty} \gamma^{v-1} R_{t+v} \tag{6b}$$

where $\gamma \in [0,1)$ is a discount factor for future rewards to balance the importance of instant and future rewards; instant and future rewards are regarded as equally important when γ is close to 1, and future rewards are neglected as γ becomes smaller.

The agent interacts with the environment during the training phase to gather training data $\{S_t, A_t, R_{t+1}, S_{t+1}\}$ and store them in a *replay buffer*. These training data is necessary for training both policy and value functions to improve behavior of the agent, in order to obtain high reward, and increase the accuracy of accumulated reward prediction, respectively. However, simultaneously collecting training data and training functions makes learning unstable because the learning could become diverged [34]. A tau updating technique was developed by Haarnoja et al. [49] that 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 updated into those of online policy and value functions (i.e., $\boldsymbol{\pi}_{\theta_1}$ and Q_{θ_2}) that collect training data. The trainable parameters are updated at a constant interval utilizing a small amount of updating weight determined by τ ($\tau \ll 1$) to stabilize the learning.

In the following DDPG algorithm, $\mathcal{L}(y, \hat{y})$ signifies a loss function between the correct value y and a predicted value $\hat{y}$, and the details of the DDPG algorithm are explained.

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:

$\boldsymbol{\pi}'_{\theta'_1}(\mathbf{S}_t) = \widehat{\mathbf{A}}_t$

$\boldsymbol{\pi}_{\theta_1}(\mathbf{S}_{t+1}) = \widehat{\mathbf{A}}_{t+1}$

$Q'_{\theta'_2}(\mathbf{S}_t, \mathbf{A}_t) = \widehat{\mathbf{Q}}_t$

$Q_{\theta_2}\left(\mathbf{S}_{t+1}, \widehat{\mathbf{A}}_{t+1}\right) = \mathbf{Q}_{t+1}$

$\nabla Q'_{\theta'_2} = \nabla_{\theta'_2} Q'_{\theta'_2}(\mathbf{S}_t, \mathbf{A}_t) \nabla_{\widehat{\mathbf{Q}}_t} \mathcal{L}\left(\mathbf{R}_{t+1} + \mathbf{Q}_{t+1}, \widehat{\mathbf{Q}}_t\right)$

$\nabla J'_{\theta'_1} = -\mathbb{E}\left[\nabla_{\theta'_1} \boldsymbol{\pi}'_{\theta'_1}(\mathbf{S}_t) \nabla_{\widehat{\mathbf{A}}_t} Q'_{\theta'_2}\left(\mathbf{S}_t, \widehat{\mathbf{A}}_t\right)|_{\widehat{\mathbf{A}}_t = \boldsymbol{\pi}'_{\theta'_1}(\mathbf{S}_t)}\right]$

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 stochastic gradient descent (SGD) [50,51] and Adam [52], Adam is utilized for the parameter updating of $\boldsymbol{\theta}'_1$ and $\boldsymbol{\theta}'_2$. By directly introducing Ornstein-Uhlenbeck noise [53] into the output value of the policy function, the exploration of the agent is triggered.

4. Graph attention network

This paper builds the agent using GAT which can process graph representations so that the agent can compute actions from states represented as graphs. The operations of GAT are briefly explained in this section. A graph consisting of n nodes with g features per node is represented as a node feature matrix $\mathbf{N} \in \mathbb{R}^{n \times g}$, an adjacency matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$, and a weighted adjacency matrix $\mathbf{P} \in \mathbb{R}^{n \times n}$, indicating features of each node, nodal connections, and weights of nodal connections, respectively. Updated node signal $\mathbf{N}' \in \mathbb{R}^{n \times h}$ can be computed from the aggregation of the features of nodes and their adjacent nodes where h denotes the dimension of embedding space.

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

$$\mathbf{N}' = \sigma(\boldsymbol{\alpha}\mathbf{M}\mathbf{N}\mathbf{w}) \tag{7}$$

where σ(·) is a non-linear activation function. A GAT computing unit can be considered as a *layer*, 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.

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

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^{\text{T}}[\mathbf{w}h_i \parallel \mathbf{w}h_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}(\mathbf{a}^{\text{T}}[\mathbf{w}h_i \parallel \mathbf{w}h_k])\right)} \tag{8}$$

where $\mathbf{a}$ is the trainable attention weight. h_i is the nodal features or embedding feature of node i. exp (x), $\parallel$, and LeakyReLU denote the exponential function of x, the concatenation operation which horizontally joins one or more matrices to make a new matrix, and the rectified linear activation function [54,55], respectively. The subscript ij indicates (i, j) component of a matrix.

5. DDPG and GAT for optimization of latticed shells

In this section, a geometry optimization method for latticed shells using DDPG and GAT is explained. In Section 5.1, the state of the structure, including configurations of nodes and elements as well as internal forces, is explained. Information of the Bézier control net, including control points and configuration of the control net represented by graphs, is also explained. Section 5.2 explains how the GAT-DDPG agent computes output from the state. Section 5.3 explains how the output of the agent is interpreted to modify the structure. Section 5.4 describes how the reward is computed after the structure is modified.

5.1 State

Graphs are used for describing data of the structure, such as nodal coordinates and internal forces, at each optimization phase, comparable to the step in the MDP formulation. In the node feature matrix $\mathbf{N} \in \mathbb{R}^{n\times 5}$ of this study, each node has five features, and the ith row $\boldsymbol{n}_i$ of is written as

$$\boldsymbol{n}_i = \left\{\frac{1}{D_x}\left(x_i - \min_{p\in\{1,\dots,n\}} x_p\right) \quad \frac{1}{D_y}\left(y_i - \min_{p\in\{1,\dots,n\}} y_p\right) \quad z_i/z_{\max} \quad k^i_{\text{free}} \quad k^i_{\text{fix}}\right\}$$
$$D_x = \max_{p\in\{1,\dots,n\}} x_p - \min_{p\in\{1,\dots,n\}} x_p,\ D_y = \max_{p\in\{1,\dots,n\}} y_p - \min_{p\in\{1,\dots,n\}} y_p$$

where x_i, y_i, and z_i are the coordinates of node i, k^i_{free} and k^i_{fix} are specified based on the boundary condition; $\left(k^i_{\text{free}}, k^i_{\text{fix}}\right) = (0, 1)$ if there is a fixed support at node i, and $\left(k^i_{\text{free}}, k^i_{\text{fix}}\right) = (1, 0)$ otherwise.

Similar to Kupwiwat et al. [38], structural elements and their internal forces are represented using adjacency matrices and weighted adjacency matrices, respectively. In the adjacency matrix of the frame element $\mathbf{M}_1$, each entry is $m_{1_{ij}} = m_{1_{ji}} = k^e_{\text{frame}}$, where k^e_{frame} denotes the presence and absence of a 12-DoF frame e linking nodes i to j by $k^e_{\text{frame}} = 1$ and 0, respectively. In the adjacency matrix of the truss $\mathbf{M}_2$, each entry is $m_{2_{ij}} = m_{2_{ji}} = k^e_{\text{truss}}$, where k^e_{truss} indicates the presence and absence of a 6-DoFs truss e connecting nodes i and j by $k^e_{\text{truss}} = 1$ and 0, respectively. The adjacency matrix of both frame and truss $\mathbf{M}_3$ is obtained by $\mathbf{M}_3 = \mathbf{M}_1 + \mathbf{M}_2$.

Weighted adjacency matrices are utilized to express the internal forces showing the effectiveness of structural elements. The weighted adjacency matrix of the frame element $\mathbf{P}_1$ is determined based on the proportion of the bending moment to the axial force, which is a useful indicator to minimize the strain energy for this type of structure. The entry $p_{1_{ij}}$ in $\mathbf{P}_1$ corresponding to internal forces of the frame e which links nodes i and j is defined as

$$p_{1_{ij}} = \frac{k^e_{\text{frame}} b'_{ei}}{(a'_e + 1)} \tag{9a}$$

$$b'_{ei} = \frac{\left(|b_{ei}| - {b_{\text{f}}}^{\min}\right)}{\left({b_{\text{f}}}^{\max} - {b_{\text{f}}}^{\min}\right)} \tag{9b}$$

$$a'_e = \frac{(|a_e| - {a_{\text{f}}}^{\min})}{({a_{\text{f}}}^{\max} - {a_{\text{f}}}^{\min})} \tag{9c}$$

in which b_{ei} and a_e are the bending moment on the section at node i around the horizontal axis and the axial force of frame e, respectively. ${b_{\text{f}}}^{\min}$ and ${b_{\text{f}}}^{\max}$ are the minimum and maximum absolute values of bending moments at frames ends, respectively. ${a_{\text{f}}}^{\min}$ and ${a_{\text{f}}}^{\max}$ are the minimum and maximum absolute values of axial forces of frames.

Given a truss e that links nodes i and j, the entry $p_{2_{ij}}$ in $\mathbf{P}_2$ is defined as

$$p_{2_{ij}} = k^e_{\text{truss}} a'_e \tag{10a}$$

$$a'_e = \frac{(|a_e| - {a_{\text{q}}}^{\min})}{({a_{\text{q}}}^{\max} - {a_{\text{q}}}^{\min})} \tag{10b}$$

where a_e, ${a_\mathrm{q}}^{\min}$, and ${a_\mathrm{q}}^{\max}$ are the axial forces in the truss, the minimum absolute values, and the maximum absolute values of axial forces of the trusses, respectively.

Additional to the structural nodes and elements, the attributes of Bézier control points are incorporated into the graph of the Bézier control net. The control net feature matrix and the control net adjacency matrix are represented by $\mathbf{N}_\mathrm{b} \in \mathbb{R}^{n_\mathrm{b}\times 5}$ and $\mathbf{M}_\mathrm{b} \in \mathbb{R}^{n_\mathrm{b}\times n_\mathrm{b}}$, respectively, as

$$\boldsymbol{n}_{\mathrm{b}i} = \left\{ x_i^\mathrm{b} \Big/ \max_{p\in\{1,\dots,n_\mathrm{b}\}} x_p^\mathrm{b} \quad y_i^\mathrm{b} \Big/ \max_{p\in\{1,\dots,n_\mathrm{b}\}} y_p^\mathrm{b} \quad z_i^\mathrm{b}/z_\mathrm{max}^\mathrm{b} \quad {k_\mathrm{b}}_\mathrm{free}^{\ i} \quad {k_\mathrm{b}}_\mathrm{fix}^{\ i} \right\}$$

where x_i^b, y_i^b, and z_i^b are the coordinates of control point i. ${k_\mathrm{b}}_\mathrm{free}^{\ i}$ and ${k_\mathrm{b}}_\mathrm{fix}^{\ i}$ are determined depending on the boundary condition of the structural nodes related to the control point; $\left({k_\mathrm{b}}_\mathrm{free}^{\ i}, {k_\mathrm{b}}_\mathrm{fix}^{\ i}\right) = (0,1)$ if control point i is fixed, and $\left({k_\mathrm{b}}_\mathrm{free}^{\ i}, {k_\mathrm{b}}_\mathrm{fix}^{\ i}\right) = (1,0)$ if control point i can be moved. In the Bézier control net adjacency matrix $\mathbf{M}_\mathrm{b}$, the presence and absence of a grid connecting control points i and j is represented as $m_{\mathrm{b}ij} = m_{\mathrm{b}ji} = k_\mathrm{Bézier}^e$, where $k_\mathrm{Bézier}^e$ indicates the presence and absence of a grid e that connects control point i and j by $k_\mathrm{Bézier}^e = 1$ and 0, 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.

5.2 GAT-DDPG agent

This research constructs a DDPG agent from GAT. However, the original GAT method cannot handle different types of graphs like the latticed structural graph and the Bézier control net graph. This section modifies the GAT-DDPG agent so that it can be used for solving structural optimization consisting of multiple graphs with different numbers of nodes as inputs.

Policy and value functions of the agent are made up of several GAT layers. The state information presented in Section 5.1 is used as input by the policy function to produce the output for changing Bézier control points. Rectified linear unit (ReLU) activation function, which is partially linear and effective for gradient-based optimization, such as SGD or Adam, transforms the output of each GAT layer. Having the representation as $\mathrm{ReLU}(\cdot) = \max(0,\cdot)$, it is utilized in every output of GAT layer besides the last layer of the policy function. The last GAT layer of the policy function utilizes the Sigmoid activation function, denoted as $\mathrm{Sigmoid}(\cdot) = 1/(1+\mathrm{e}^{-(\cdot)})$, to compute probability-based output for predicting the probability of doing an action A_t^i in a state S_t. GAT layers with ReLU and Sigmoid activation functions are represented in Eqs. (11a) and (11b), respectively.

$$\mu(\mathbf{N}', \mathbf{M}') = \mathrm{ReLU}\left(\mathbf{M}'\mathbf{N}'\mathbf{w}_\mu\right) \tag{11a}$$

$$\sigma(\mathbf{N}', \mathbf{M}') = \mathrm{Sigmoid}(\mathbf{M}'\mathbf{N}'\mathbf{w}_\sigma) \tag{11b}$$

where $\mathbf{N}'$ indicates a node feature matrix or a prior GAT layer output. $\mathbf{M}'$ indicates an adjacency matrix or weighted adjacency matrix used in the layer.

Embedded node signals of the graph representations of the latticed structure and the Bézier control points are obtained using different GAT layers. Since the output of the policy function should have the number of rows equal to the number of Bézier control points n_b, the embedded node signal of structural nodes is firstly modified using the global sum pooling operation (GSP) [56] which sums up all entries in each column in the matrix. A vector obtained from GSP is then reshaped into a matrix with n_b rows by the stack operation. The GSP operation and stack operation that transforms a matrix $\mathbf{V} \in \mathbb{R}^{n\times g}$ into a vector $\mathbf{a} \in \mathbb{R}^{1\times g}$ and further into a matrix $\mathbf{A} \in \mathbb{R}^{n_\mathrm{b}\times g}$, respectively, are represented in Eqs. (12) and (13).

$$\mathrm{GSP}(\mathbf{V}) = \left[\sum_{i=1}^{n} \mathrm{v}_{i,1} \quad \cdots \quad \sum_{i=1}^{n} \mathrm{v}_{i,g}\right] \in \mathbb{R}^{1\times g} = \mathbf{a} \tag{12}$$

$$\mathrm{Stack}^{n_\mathrm{b}}(\mathbf{a}) = \begin{bmatrix} \mathbf{a} \\ \vdots \\ \mathbf{a} \end{bmatrix} \} n_\mathrm{b} \text{ times} = \mathbf{A} \tag{13}$$

Utilizing these two operations, the output of the GAT layer with n rows will be transformed using GSP into a vector and then modified into a matrix with n_b rows, which allows policy and value functions to process and combine signals from both structural graph and Bézier control net graph.

Value function takes the output of policy function and graph representations of latticed structure and Bézier control points as inputs for computing a scalar Q-value $\in \mathbb{R}^{1\times 1}$. In this function, embedded node signals of the output of the policy function and the graph representations of structure and control points are also obtained using different GAT layers. GSP operation modifies these embedded node signals into 3 vectors, and they are concatenated into one vector. The joined vector becomes an input for an NN. The NN denoted as $\mathrm{f}_\mathrm{NN}(\cdot)$ for computing the Q-value is represented as follows:

$$\mathrm{f}_\mathrm{NN}(\mathbf{H}) = \boldsymbol{\mathcal{W}}_3\left(\mathrm{ReLU}(\boldsymbol{\mathcal{W}}_2(\mathrm{ReLU}(\boldsymbol{\mathcal{W}}_1\mathbf{H}^\mathrm{T} + \boldsymbol{\mathcal{B}}_1)) + \boldsymbol{\mathcal{B}}_2)\right) + \boldsymbol{\mathcal{B}}_3 \tag{14}$$

where $\mathbf{H}$, $\boldsymbol{\mathcal{W}}_{i\in\{1,2,3\}}$, and $\boldsymbol{\mathcal{B}}_{\in\{1,2,3\}}$ denote the input vector of NN, adjustable internal weight matrix, and adjustable internal bias vector, respectively.

The computations for the policy and value functions are displayed in Table 1. On the left column, the policy function separately processes inputs from state data of frames and trusses, and merges them in step 1. The output matrix from step 1 is manipulated using GSP and stack operations to create a matrix that has the number of rows equal to n_b, and then processed by a GAT layer in step 2. Inputs from the Bézier control net are processed in step 3. Lastly, step 4 computes the probability of adjusting each Bézier control point by aggregating the outputs from steps 2 and 3.

On the right column of the table, the value function also separately processes inputs from state data of frames and trusses, merges them, and transforms the merged matrix into a vector using the GSP operation in step 1. In step 2, a computation similar to those in the policy function is performed, except that the final result is transformed into a vector. In step 3, the output $\boldsymbol{\pi}$ of the policy function is processed and converted into a vector, also using the GSP. The outputs from steps 1-3 are converted into a single vector by a concatenation operation in step 4. The vector is finally used in step 5 for computing the Q-value by the NN.

Table 1 Computations of the policy and value functions

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

5.3 Action

This section explains how the output of the policy function $\boldsymbol{\pi}$ is interpreted to modify the Bézier control points. $\boldsymbol{\pi}$ has n_b rows and three columns which respectively represent each Bézier control point to be adjusted and how it should be adjusted, including moving the control point upward, moving it downward, or keeping it at the same height. The adjustment of a Bézier control point i, i.e., the action, at step t is interpreted as follows:

$$\mathrm{A}_t^i = \begin{cases} z_i^\mathrm{b} = z_i^\mathrm{b} + \Delta z^\mathrm{b} & \text{if } \max\big(\boldsymbol{\pi}_{i,1}, \boldsymbol{\pi}_{i,2}, \boldsymbol{\pi}_{i,3}\big) = \boldsymbol{\pi}_{i,1} \\ z_i^\mathrm{b} = z_i^\mathrm{b} - \Delta z^\mathrm{b} & \text{if } \max\big(\boldsymbol{\pi}_{i,1}, \boldsymbol{\pi}_{i,2}, \boldsymbol{\pi}_{i,3}\big) = \boldsymbol{\pi}_{i,2} \\ z_i^\mathrm{b} = z_i^\mathrm{b} & \text{if } \max\big(\boldsymbol{\pi}_{i,1}, \boldsymbol{\pi}_{i,2}, \boldsymbol{\pi}_{i,3}\big) = \boldsymbol{\pi}_{i,3} \end{cases} \tag{15}$$

At each step, all the control points are subjected to their associated actions.

5.4 Reward

In RL, an agent is trained using a reward signal, given to the agent after it executes action A_t in a state S_t. Rewards need to be formulated so that the agent takes actions that lead to preferable states. Let E_0 and E_t denote the initial strain energy and the strain energy at step t of the structure, respectively. Considering that small strain energy is preferred in this study, reward signal R_{t+1} is computed from the variation of the strain energy as

$$\mathrm{R}_{t+1} = (E_t - E_{t+1})/E_0 \tag{16}$$

6. Numerical examples

6.1 General settings for experiments and structural model

The agent is trained to optimize the structure during the *training phase*, which measures the abilities of an agent to improve (i.e., obtain a greater reward) and stabilize its performance. After the training, the performance of the agent

is evaluated in the *test phase* using larger structures. Algorithms of the proposed method in both phases are shown in Fig. 3.

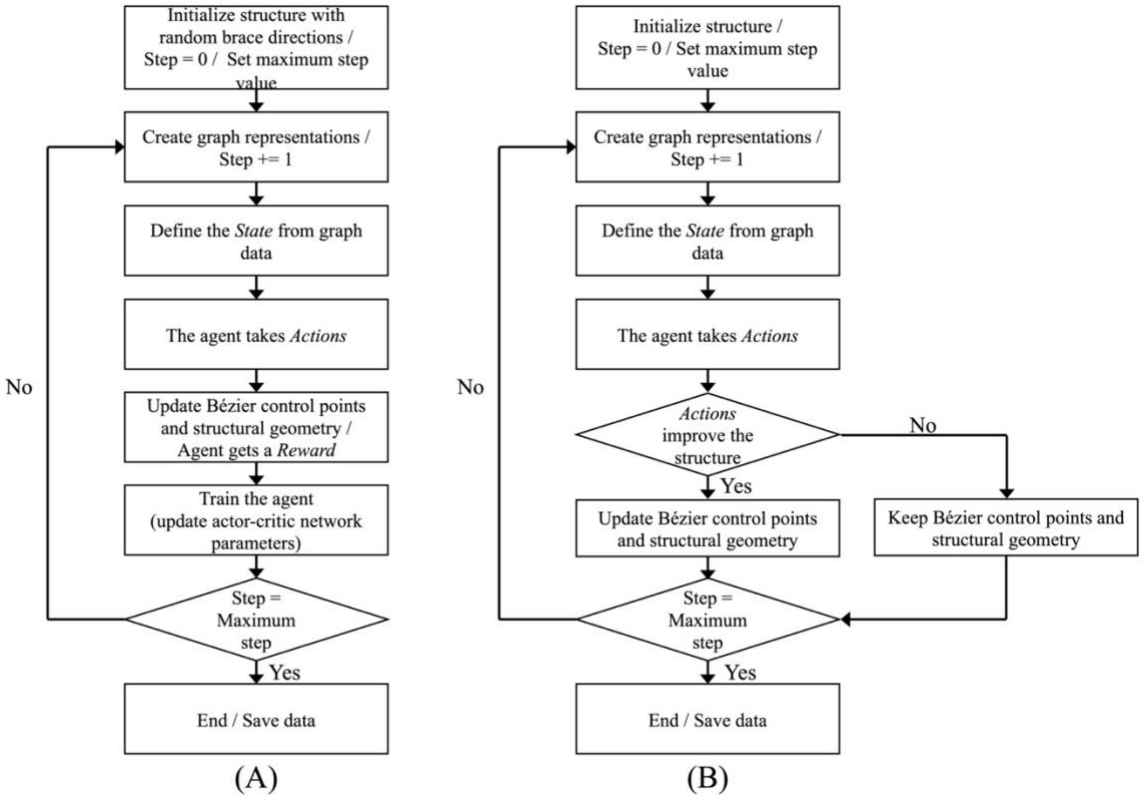

Fig. 3 The algorithms; (A) Training phase, (B) Test phase

In this paper, the structures to be optimized are the latticed shells with a 1.0 m by 1.0 m grid. The 12-DoF frame element has a hollow cylindrical section with an external diameter of 34 mm and a thickness of 2.3 mm with a mass of 1.8 kg/m. The 6-DoF truss element has a solid circular section with a diameter of 12 mm and a mass of 0.9 kg/m. The 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. Fig. 3 shows the flowcharts of algorithms for the training and test phases. The optimization or a *game* ends when the number of steps reaches the predetermined maximum step. The positions of Bézier control points and data of the structural configurations including nodal coordinates, connections, properties of the elements, and loads are saved as an output of the program. Note again that the agent minimizes the strain energy of the structure under these specific load and predetermined boundary constraint conditions. However, different numbers of structural nodes, elements, Bézier control points, and various boundary constraint conditions can be assigned in each optimization.

6.2 Training phase

In each training game, a 4×4-grid latticed shell structural model is initialized randomly, chosen from structural models 1-6 as shown in Fig. 4 where the bracing direction in each grid is also randomly initialized. Every node in the structure is initialized on a plane with a height of 0 and the heights of supports are fixed.

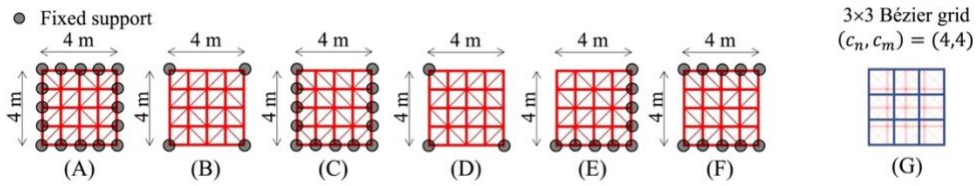

Fig. 4 Structural models 1–6 for the training phase with $c_n = c_m = 4$ for the Bernstein basis functions; (A) Model 1, (B) Model 2, (C) Model 3, (D) Model 4, (E) Model 5, (F) Model 6, (G) Bézier control net

Each game has a maximum number of steps of 20. 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. $z_{\min}$, $z_{\max}$, $z^{\mathrm{b}}_{\min}$, and $z^{\mathrm{b}}_{\max}$ are 0, 1 m, 0, and 2 m, respectively. Replay buffer with the mini-batch size of 32 is utilized for training surrogate functions. Trainable parameters of surrogate policy and value functions were adjusted utilizing Adam optimizer with the learning rates of 10^{-6} and 10^{-5}, respectively. In the value function where the mean square error is used as the loss function, each layer in the NN in Eq. (14) has 200 cells. Each GAT layer in both policy and value function has the dimension of embedding space of 200. Every 100 steps, trainable parameters of surrogate functions are updated to online functions with $\tau = 0.05$.

The agent has been taught to play 1000 games. In Fig. 5, the vertical and horizontal axes represent the accumulated reward obtained in the game, and the game number, respectively. From Fig. 5, the agent improves to achieve the higher reward; the cumulative reward has increased with a fluctuation period during the first 400 games of training and has increased, then maintained to relatively high values afterward. Since the structural models are changed every training game, 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.

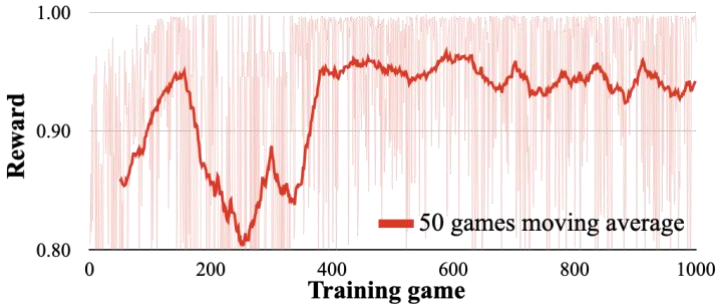

Fig. 5 Variation of moving average of reward during the training

6.3 Test phase

The trained agent is applied to structures that have not been used in the training phase to verify its applicability for geometry optimization problems in this phase. In this paper, three experiments in Section 6.3.1, 6.3.2, and 6.3.3 are conducted, respectively, to verify the ability of the agent compared to other methods, the usefulness of the proposed method for geometry optimization, and the robustness of the agent when the type of design variables is changed. In this phase, only an action that improves the value of the objective function is accepted at each step.

6.3.1 Verifying ability of the agent for geometry optimization compared to PSO and SA

In this experiment, the ability of the agent to optimize the geometry of latticed shells using Bézier surface is verified. Structural models used in this experiment are fully braced 6×6-grid latticed shells with structural models 1-6 and three cases of c_n and c_m of Bernstein basis functions as shown in Fig. 6. Fully braced 10×10-grid latticed shells with structural models 1-2 and one case of c_n and c_m are also considered as shown in Fig. 7. The number of steps for the optimization is 500 for 6×6-grid and 1000 for 10×10-grid shells. $z_{\min}$ and $z^{\mathrm{b}}_{\min}$ are 0 for every structure. $z_{\max}$ and $z^{\mathrm{b}}_{\max}$ are 1 m and 2 m, respectively, for 6×6-grid shells, and are 2 m and 4 m, respectively, for 10×10-grid shells. The increment Δz^{b} is 0.1 m for all cases.

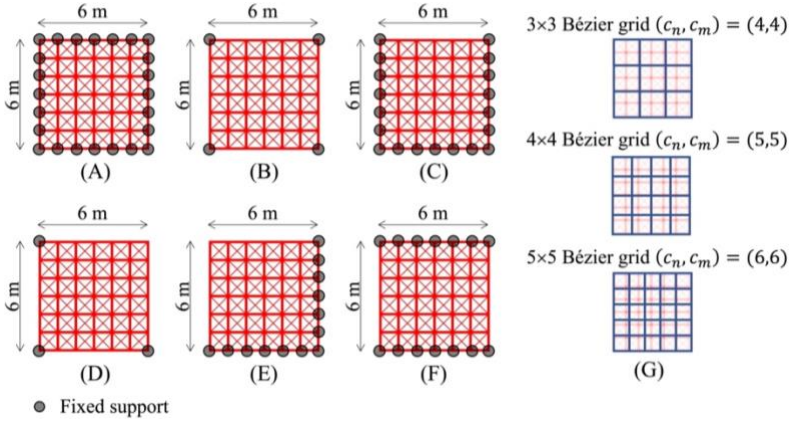

Fig. 6 6×6-grid latticed shells (fully braced) with $(c_n, c_m) = (4,4)$, $(5,5)$, and $(6,6)$; (A) Model 1, (B) Model 2, (C) Model 3, (D) Model 4, (E) Model 5, (F) Model 6, (G) Bézier control nets

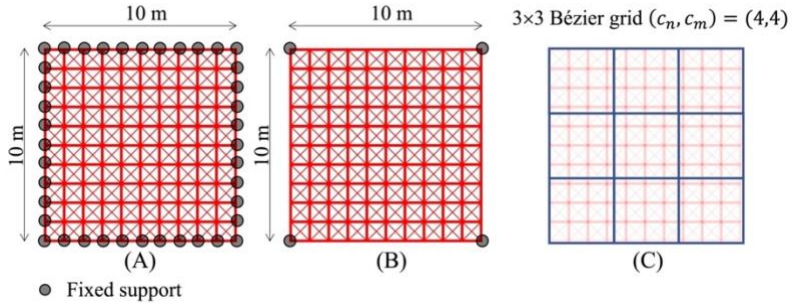

Fig. 7 10×10-grid latticed shells (fully braced) with $(c_n, c_m) = (4,4)$; (A) Model 1, (B) Model 2, (C) Bézier control net

Results obtained from GAT-DDPG, optimizing each structural model ten times, are compared to those obtained from PSO and SA. PSO is an optimization algorithm that iteratively improves the positions of candidate particles called the swarm in the search space. The position of a particle, which is the heights of Bézier control points, is improved by computing the velocity at the current particle position, the best-known position of the particle, and the best particle position among the swarm. The objective function, which is the total strain energy, is then evaluated at the particle position.

Numbers of populations and generations of PSO are set based on the number of structural analyses or the computational cost of GAT-DDPG to benchmark the performance of the agent. In the following examples, the number of particles in PSO is 20. PSO algorithm is terminated after 10 and 20 steps of optimization for 6×6-grid and 10×10-grid structures, respectively. This paper utilized the PSO algorithm from distributed evolutionary algorithms in python (DEAP) [57] which implements the original PSO algorithm proposed by Poli et al. [58]. Optimal solutions are found for five schemes of weight coefficients for the best-known position of the particle and the best particle position among the swarm $(\phi_1, \phi_2) = (0.5,0.5)$, (0.4,0.6), (0.6,0.4), (0.2,0.8), and (0.8,0.2), respectively.

SA is a method of stochastic hill climbing where a candidate solution is modified randomly and the modified solution is accepted to replace the current solution with a probability that decreases with each iteration, controlled by the temperature parameter. This paper utilized the SA algorithm called dual annealing from the python library named SciPy [59]. 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. In this example, the maximum numbers of iterations are 1000 and 2000 for 6×6-grid and 10×10-grid structures, respectively.

The computational costs of GAT-DDPG in this phase (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 are shown in Table 2. Note that the number of structural analyses of GAT-DDPG in the training phase is 20000.

Table 2 Total computational cost of each method

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

The best results of GAT-DDPG are compared to those of PSO and SA among different random seeds in Table 3 which indicates that results from the GAT-DDPG agent are competitive compared to those from PSO and SA. Therefore, the quality of the solutions of the GAT-DDPG agent has been verified. Note that the GAT-DDPG agent can obtain good results even though the structural grid size and Bézier grid size differ from those in the training phase.

Table 3 Strain energy (N·m) of the best results obtained from GAT-DDPG , PSO, and SA

structural model	grid size	(c_n, c_m)	GAT-DDPG (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

Figs. 8-9 show the best results, including the final geometry and the variation of strain energy at each step, of GAT-DDPG using $(c_n, c_m) = (4,4)$ for 10×10-grid shells with structural model 1 and 2, respectively. Even though structural configuration differs from those used in the training phase, the trained agent can minimize the strain energy by adjusting structural geometry through Bézier control points and obtain reasonable geometries. Hence, the agent can be trained using small structural models and deployed into larger ones. Note that, in this experiment, the strain energy reduction in the early steps is large because the initial geometry is flat while the strain energy reduction is small in the later steps of optimization as shown in (E) of Figs. 8 and 9.

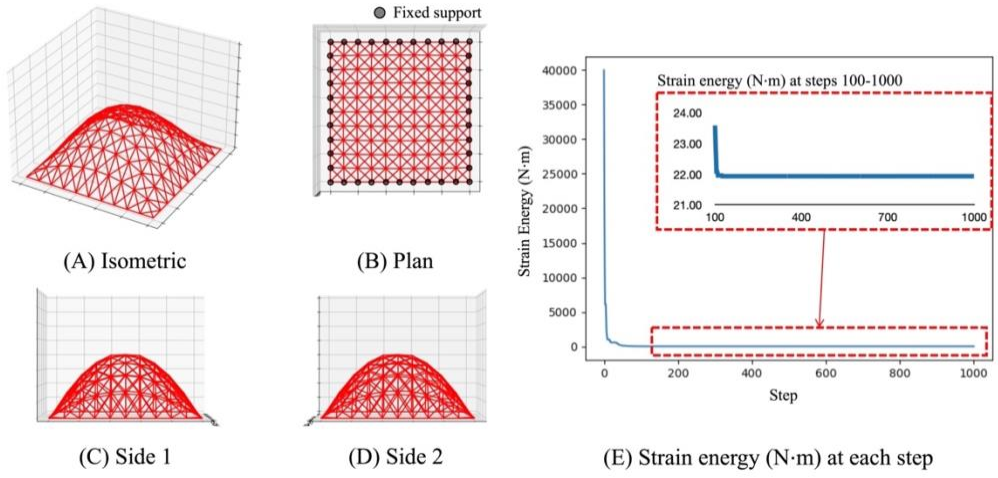

Fig. 8 10×10-grid shell: Structural model 1

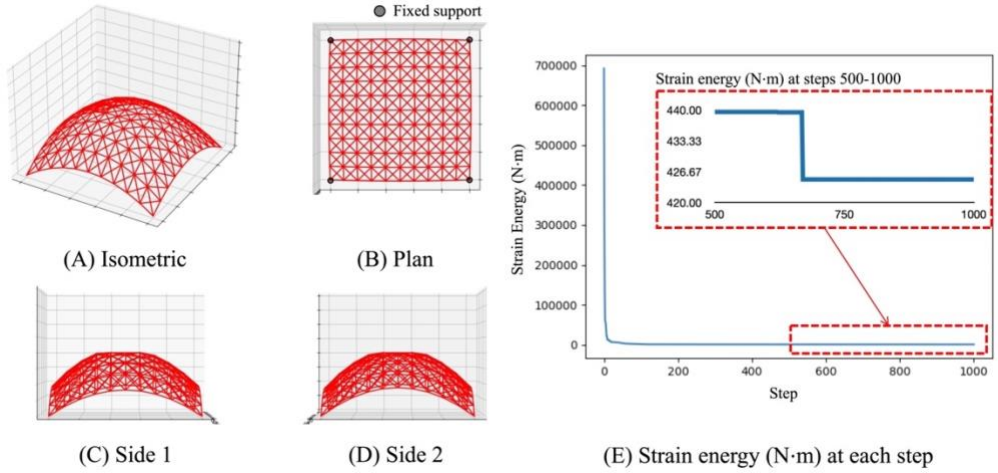

Fig. 9 10×10-grid shell: Structural model 2

6.3.2 Verifying the usefulness of the agent for geometry optimization

This experiment aims to verify the usefulness of the agent for the geometry optimization of latticed shells. The trained agent is applied to a 20×20-grid latticed shell using the structural model 1 with one case of Bernstein basis functions as shown in Fig. 10. The number of steps, $z_{\min}$, $z_{\max}$, $z^{\mathrm{b}}_{\min}$, $z^{\mathrm{b}}_{\max}$, and Δz^{b} are 400, 0, 4 m, 0, 8 m, and 0.1 m, respectively.

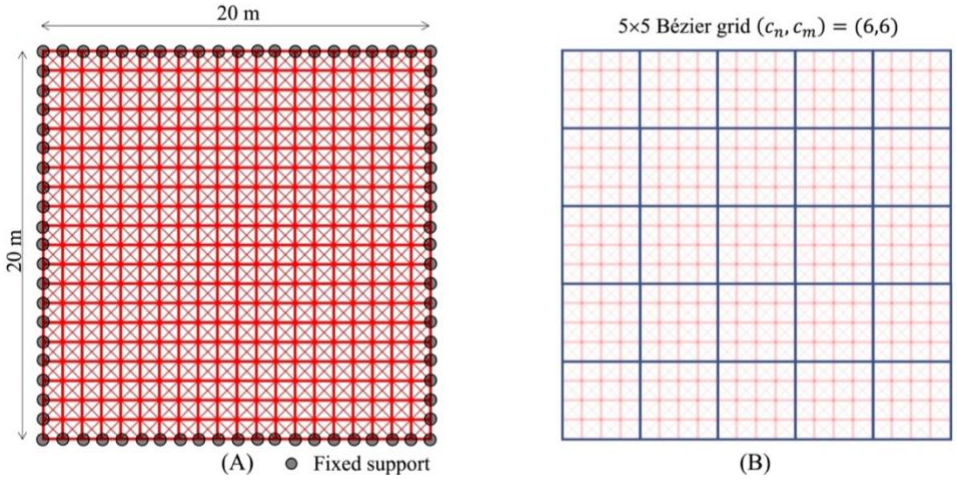

Fig. 10 20×20-grid latticed shell (fully braced) with $(c_n, c_m) = (6,6)$; (A) Structural model 1, (B) Bézier control net

Results of GAT-DDPG obtained from 10 tests are compared with those of PSO and SA utilizing similar parameters, schemes, and trials as in Section 6.3.1. In this experiment, the number of particles in PSO is 20 and the algorithm is terminated after 40 steps of optimization. The maximum number of SA iterations is 4000. Note that the total numbers of structural analysis of the proposed method, PSO, and SA are 4000, 20000, and 20000, respectively. Computational costs of PSO and SA are larger than that of GAT-DDPG , which verifies the usefulness of the agent for geometry optimization using small computational cost.

Table 4 shows the comparison between the best result of GAT-DDPG and the best results obtained by PSO and SA among different random seeds. The GAT-DDPG agent can obtain a better result than PSO and SA while utilizing small computational cost. Therefore, the proposed method can be effective when there are many structural models of latticed shells to be optimized; e.g., in the preliminary design process. The final geometry of the best result obtained by GAT-DDPG is shown in Fig.11.

Table 4 Strain energy (N·m) of the best results obtained from GAT-DDPG , PSO, and SA

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

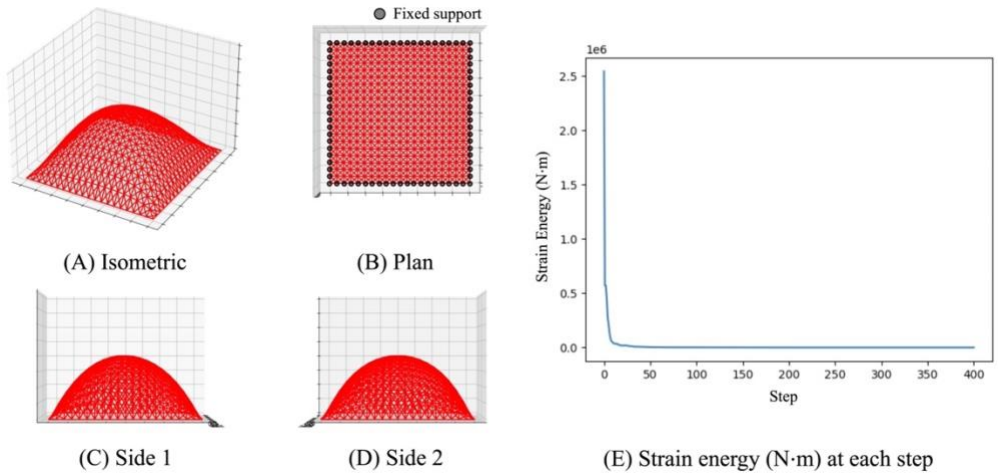

Fig. 11 20×20-grid shell: Structural model 1

6.3.3 Verifying ability of the agent when the type of design variables and initial geometry are changed

In this section, the applicability of the trained agent for solving geometry optimization of fully braced structures that have initial irregular geometries using different types of design variables is presented. Structural models in this experiment are 10×10-grid latticed shells with structural models 1 and 2 and $c_n = c_m = 4$ for the Bernstein basis functions, similar to Fig. 7, but the heights of Bézier control points are randomly initialized in the range [0,2] m.

The number of steps, $z_{\min}$, $z_{\max}$, $z^{\mathrm{b}}_{\min}$, and $z^{\mathrm{b}}_{\max}$ are 200 steps, 0, 2 m, 0, and 4 m respectively. In this experiment, the action is modified so that the agent at step t adjusts the Bézier control point i using the value directly obtained from the output of the policy function as follows:

$$\mathrm{A}^i_t = \begin{cases} z^{\mathrm{b}}_i = z^{\mathrm{b}}_i + \left(\boldsymbol{\pi}_{i,1} \times \Delta z^{\mathrm{b}}\right) & \left(\text{if } \max\left(\boldsymbol{\pi}_{i,1}, \boldsymbol{\pi}_{i,2}, \boldsymbol{\pi}_{i,3}\right) = \boldsymbol{\pi}_{i,1}\right) \\ z^{\mathrm{b}}_i = z^{\mathrm{b}}_i - \left(\boldsymbol{\pi}_{i,2} \times \Delta z^{\mathrm{b}}\right) & \left(\text{if } \max\left(\boldsymbol{\pi}_{i,1}, \boldsymbol{\pi}_{i,2}, \boldsymbol{\pi}_{i,3}\right) = \boldsymbol{\pi}_{i,2}\right) \\ z^{\mathrm{b}}_i = z^{\mathrm{b}}_i & \left(\text{if } \max\left(\boldsymbol{\pi}_{i,1}, \boldsymbol{\pi}_{i,2}, \boldsymbol{\pi}_{i,3}\right) = \boldsymbol{\pi}_{i,3}\right) \end{cases} \tag{17}$$

where Δz^{b} is 0.1 m. Note that the design variables in this Section are continuous whereas those of Section 6.2, Section 6.3.1 and Section 6.3.2 are discrete.

Ten trials of each structural model optimized by the agent are made to obtain the minimum (min.), mean, and standard deviation (std.) of the strain energy for each structural model shown in Table 5. In structural model 1, the best result has strain energy similar to those of Section 6.3.1 while in structural model 2, the best result has strain energy larger than those of Section 6.3.1 by around 10%. However, it should be noted that the computational cost of this section is 50% of those in Section 6.3.1.

Table 5 Results from initial irregular geometries

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

Figs. 12 and 13 show the initial geometry, the final geometry, and the history of strain energy from best results of 10×10-grid latticed shells with structural models 1 and 2, respectively. The agent can obtain reasonably optimized geometries despite using different initial geometries and types of design variables.

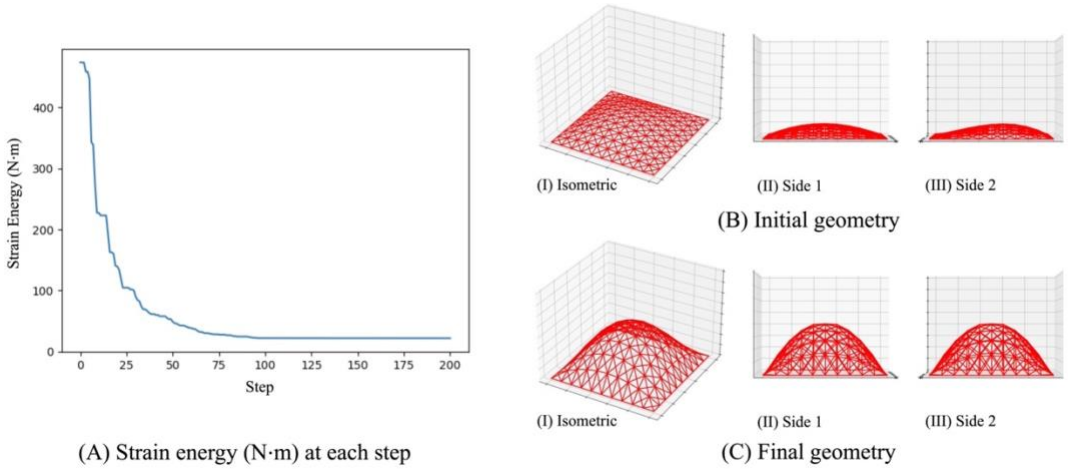

Fig. 12 10×10-grid latticed shell with initial irregular geometry: Structural model 1

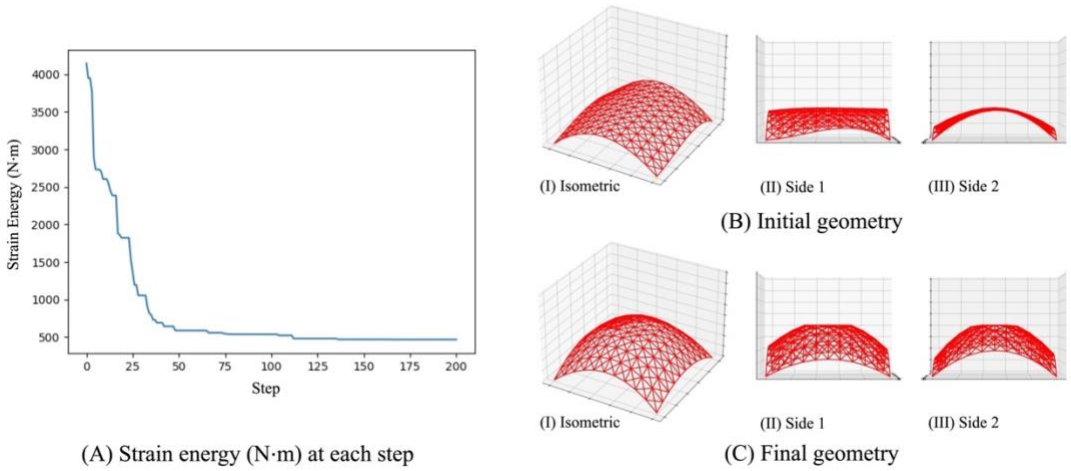

Fig. 13 10×10-grid latticed shell with initial irregular geometry: Structural model 2

7. Conclusion

In this paper, a combined method has been proposed for geometry optimization of latticed shells to minimize the strain energy utilizing DDPG and GAT. The proposed DDPG agent with a GAT framework allows the agent to process the whole structural configuration and response together with the Bézier control net using graph manipulations inside the GAT architecture. The structure is formulated as graphs where node feature matrices, adjacency matrices, and weighted adjacency matrices are utilized to represent structural configuration, internal forces, and Bézier control net configuration. The agent is trained for the geometry optimization task by adjusting Bézier control points in an MDP environment where the agent collects its training data. During the training phase, the value function updates its internal trainable parameters to minimize the prediction loss of obtained reward while the policy function updates its trainable parameters to do actions that maximize the reward defined using the strain energy of the structure.

Numerical examples show that, even though the structural size, Bézier grid size, and bracing orientations change from those in the training phase, the trained agent can optimize structural geometry to reduce the strain energy and obtain competitive results with PSO and SA but requires less computational cost. Therefore, the proposed method can become useful when the optimization problem should be solved many times; e.g., in the preliminary design process of a latticed shell. The proposed method is also versatile for optimizing structures when the type of design variables and initial geometries differ from the training phase.

It should be noted that the agent requires a considerable computational cost during the training phase. However, once the agent is trained, it can be deployed into other compatible computational tools and used for larger structures without retraining. Therefore, the efficiency of the proposed method depends relatively on the number of problems it is applied to after the training.

The proposed method is expected to be a component for solving more complex problems such as simultaneous geometry optimization and topology optimization, also called layout optimization which are our future research interests.

Data statement

Experiment data from this study are accessible to corresponding authors upon request.

References

[1] Christensen P W, Klarbring, A (2010) In: An introduction to structural optimization. Springer, Netherlands, pp 1-8

[2] 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

[3] Ohsaki M (2010) In: Optimization of finite dimensional structures. CRC Press, pp 259-313. http://doi.org/10.1201/EBK1439820032

[4] Topping B H (1983) Shape optimization of skeletal structures: a review. Journal of Structural Engineering 109(8):1933-1951. http://doi.org/10.1061/(ASCE)0733-9445(1983)109:8(1933)

[5] Wang D, Zhang W H, Jiang J S (2002) Truss shape optimization with multiple displacement constraints. Computer Methods in Applied Mechanics and Engineering 191:3597-361. http://doi.org/10.1016/S0045-7825(02)00297-9
[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 38:168-182. http://doi.org/10.1016/j.engappai.2014.10.012
[7] Choi K K, Kim N H (2005) In: Structural sensitivity analysis and optimization 1: linear systems. Springer, New York, pp 119-170.
[8] Adelman H, Haftka R (1986) Sensitivity analysis of discrete structural systems. American Institute of Aeronautics and Astronautics Journal 24:823-832. http://doi.org/10.2514/3.48671
[9] Kirsch U (1994) Efficient sensitivity analysis for structural optimization. Computer Methods in Applied Mechanics and Engineering 117:143-156. http://doi.org/10.1016/0045-7825(94)90080-9
[10] Putresza J, Kolakowski P (1998) Sensitivity analysis of frame structures (virtual distortion method approach). International Journal for Numerical Methods in Engineering 43:1085-1108. http://doi.org/10.1002/1097-0207(20010228)50:6%3C1307::AID-NME38%3E3.0.CO;2-Q
[11] Jenkins W M (1992) Plane frame optimum design environment based on genetic algorithm. Journal of Structural Engineering 118:3103-3112. http://doi.org/10.1061/(ASCE)0733-9445(1992)118:11(3103)
[12] Raich A, Ghaboussi J (2000) Evolving structural design solutions using an implicit redundant genetic algorithm. Structural and Multidisciplinary Optimization 20:222-231. http://doi.org/10.1007/s001580050150
[13] Kaveh A, Rahami H (2006) Analysis, design and optimization of structures using force method and genetic algorithm. International Journal for Numerical Methods in Engineering 65(10):1570-1584. http://doi.org/10.1002/nme.1506
[14] Kimura T, Ohmori H (2008) Computational morphogenesis of free form shells. Journal of the International Association for Shell and Spatial Structures 49(3):175-180
[15] 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 331:427-455. http://doi.org/10.1016/j.cma.2017.09.034
[16] Bernstein S (1912) Démonstration du théorème de weierstrass fondée sur le calcul des probabilités. Comm Soc Math Kharkow 13:1-2
[17] Ramm E, Bletzinger K U, Reitinger R (1993) Shape optimization of shell structures. Revue Européenne des Éléments Finis 2(3):377-398. http://doi.org/10.1080/12506559.1993.10511083
[18] 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. http://doi.org/10.1137/1.9781611971521.ch5
[19] 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 12:109-119. http://doi.org/10.1177/026635119701200205
[20] Ohsaki M, Hayashi M (2000) Fairness metrics for shape optimization of ribbed shells. Journal of the International Association for Shell and Spatial Structures 41(1):31-39
[21] Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. Psycological Review 65(6):386-408. http://doi.org/10.1037/h0042519
[22] Ivakhnenko A G (1968) The group method of data handling – A rival of the of stochastic approximation. Soviet Automatic Control 13(3):43-55
[23] Goodfellow I, Bengio Y, Courville A (2016) In: Deep learning. MIT Press, Massachusetts, pp 163-220
[24] Vanluchene R D, Sun R (1990) Neural networks in structural engineering. Computer-Aided Civil and Infrastructure Engineering 5(3):207-215. http://doi.org/10.1111/j.1467-8667.1990.tb00377.x
[25] Berke L, Hajela P (1993) Application of neural nets in structural optimization. In: Rozvany, G.I.N. (eds) Optimization of large structural systems. NATO ASI series vol 231, pp 731-745. http://doi.org/10.1007/978-94-010-9577-8_36
[26] Mai T H, 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 196:103572. http://doi.org/10.1016/j.finel.2021.103572
[27] Christian M C (2022) Machine learning in structural design: an opinionated review. Frontiers in Built Environment 8:815717. http://doi.org/10.3389/fbuil.2022.815717
[28] Mirra G, Pugnale A (2021) Comparison between human-defined and ai-generated design spaces for the optimisation of shell structures. Structures 34:2950-2961. http://doi.org/10.1016/j.istruc.2021.09.058

[29] Kingma P D, Welling M (2019) An introduction to variational autoencoders. Foundations and Trends in Machine Learning 12(4):307-392. http://doi.org/10.1561/2200000056
[30] 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 362:112790. https://doi.org/10.1016/j.cma.2019.112790
[31] Zheng H, Moosavi V, Akbarzadeh M (2020) Machine learning assisted evaluations in structural design and construction. Automation in Construction 119:103346. http://doi.org/10.1016/j.autcon.2020.103346
[32] 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 international association for shell and spatial structures, pp 1-8
[33] 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 69:1191-1212. http://doi.org/10.1007/s00466-021-02137-8
[34] 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: International conference on learning representations (Poster)
[35] Kupwiwat C, Yamamoto K (2021) Fundamental study on morphogenesis of shell structure using reinforcement. Journal of Structural Engineering B. Architectural Institute of Japan, 67B:211-218
[36] Hayashi K, Ohsaki M (2021) Reinforcement learning and graph embedding for binary truss topology optimization under stress and displacement constraints. Frontiers in Built Environment 6:59. http://doi.org/10.3389/fbuil.2020.00059
[37] Zhu S, Ohsaki M, Hayashi K, Guo X (2021) Machine-specified ground structures for topology optimization of binary trusses using graph embedding policy network. Advances in Engineering Software 159:103032. http://doi.org/10.1016/j.advengsoft.2021.103032
[38] 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 8:899072. http://doi.org/10.3389/fbuil.2022.899072
[39] Gilmer J, Schoenholz S, Riley P, Vinyals O, Dahl G (2017) Neural message passing for quantum chemistry. In: Proceedings of international conference on machine learning, vol 70, pp 1263-1272
[40] Kipf T N, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: Proceedings of international conference on learning representations, pp 1-14
[41] Veličković P, Cucurull G, Casanova A, Romero A, Lio P, Bengio Y (2017) Graph Attention Networks. In: Proceedings of international conference on learning representations, pp 1-12
[42] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez A N, Kaiser L, Polosukhin I (2017) Attention is all you need. In: Proceedings of advances in neural information processing systems, pp 5998–6008
[43] 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. http://doi.org/10.1109/ICDM50108.2020.00181
[44] Huang J, Shen H, Hou L, Cheng X (2019) Signed Graph Attention Networks. In: Proceedings of international conference on artificial neural network, vol 28, pp 566-577. https://doi.org/10.1007/978-3-030-30493-5_53
[45] 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 35(2):1594-1608. http://doi.org/10.1109/TKDE.2021.3105544
[46] Sutton R S, Andrew G B (2018) In: Reinforcement learning, an introduction. MIT Press, Massachusetts, pp 1-22. http://doi.org/10.1109/TNN.1998.712192
[47] Bellman R (1957) A markovian decision process. Indiana University Mathematics Journal 6(4):679-684. http://doi.org/10.1512/iumj.1957.6.56038
[48] Bellman R (1954) The theory of dynamic programming. Bulletin of the American Mathematical Society 60:503-515. http://doi.org/10.1090/S0002-9904-1954-09848-8
[49] 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 international conference on machine learning, pp 1861-1870
[50] Robbins H, Monro S (1951) A stochastic approximation method. The Annals of Mathematical Statistics 22(3):400-407. http://doi.org/10.1214/aoms/1177729586
[51] Kiefer J, Wolfowitz J (1952) Stochastic estimation of the maximum of a regression function. The Annals of Mathematical Statistics 23(3):462-466. http://doi.org/10.1214/aoms/1177729392

[52] Kingma D, Ba J (2015) Adam: a method for stochastic optimization. In: Proceedings of international conference on learning representations, pp 1-15
[53] Uhlenbeck G E, Ornstein L S (1930) On the theory of the brownian motion. Physical Review. 36(5):823-841. http://doi.org/10.1103/PhysRev.36.823
[54] Maas A, Hannun A, Ng A (2013) Rectifier nonlinearities improve neural network acoustic models. In: Proceedings of international conference on machine learning, vol 30(1), pp 3
[55] Nair V, Hinton G E (2010) Rectified linear units improve restricted boltzmann machines. In: Proceedings of international conference on machine learning, pp 807-814
[56] 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
[57] 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 13(1):2171-2175. http://doi.org/10.5555/2503308.2503311
[58] Poli R, Kennedy J, Blackwell T (2007) Particle swarm optimization: an overview. Swarm Intelligence 1:33-57. http://doi.org/10.1007/s11721-007-0002-0
[59] Virtanen P, Gommers R, Oliphant T E et al (2020) SciPy 1.0: fundamental algorithms for scientific computing in Python. Nature Methods 17:261-272. http://doi.org/10.1038/s41592-019-0686-2