Gravity-GNN: Deep Reinforcement Learning Guided Space Gravity-based Graph Neural Network

Huaming Wu
Center for Applied Mathematics, Tianjin University
Tianjin, China
whming@tju.edu.cn

Lei Tian
Tianjin University
Tianjin, China
tianlei@tju.edu.cn

Chaogang Tang
China University of Mining and Technology
Xuzhou, China
cgtang@cumt.edu.cn

Pengfei Jiao
Hangzhou Dianzi University
Hangzhou, China
pjiao@hdu.edu.cn

Minxian Xu
Chinese Academy of Sciences
Shenzhen, China
mx.xu@siat.ac.cn

Huijun Tang*
Durham University
Durham, United Kingdom
huijun.tang@durham.ac.uk

Abstract

Graph Neural Networks (GNNs) have demonstrated remarkable capabilities in handling graph data. Typically, GNNs recursively aggregate node information, including node features and local topological information, through a message-passing scheme. However, most existing GNNs are highly sensitive to neighborhood aggregation, and irrelevant information in the graph topology can lead to inefficient or even invalid node embeddings. To overcome these challenges, we propose a novel Space Gravity-based Graph Neural Network (Gravity-GNN) guided by Deep Reinforcement Learning (DRL). In particular, we introduce a novel similarity measure called "node gravity", inspired by gravity between particles in space to compare nodes in graph data. Furthermore, we employ DRL technology to learn and select the most suitable number of adjacent nodes for each node. Our experimental results on various real-world datasets demonstrate that Gravity-GNN outperforms state-of-the-art methods regarding node classification accuracy, while exhibiting greater robustness against disturbances.

CCS Concepts

• **Computing methodologies** → **Artificial intelligence**; **Neural networks**; • **Information systems** → *Data mining*.

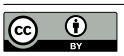
Keywords

Graph Data; Graph Neural Network; Node Gravity; Deep Reinforcement Learning; Node Embeddings

ACM Reference Format:

Huaming Wu, Lei Tian, Chaogang Tang, Pengfei Jiao, Minxian Xu, and Huijun Tang. 2025. Gravity-GNN: Deep Reinforcement Learning Guided Space Gravity-based Graph Neural Network. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management (CIKM '25), November 10–14, 2025, Seoul, Republic of Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3746252.3761337

*Corresponding author.



CIKM '25, Seoul, Republic of Korea.

ACM ISBN 979-8-4007-2040-6/2025/11
https://doi.org/10.1145/3746252.3761337

1 Introduction

Graph data is ubiquitous in a multitude of practical applications, with numerous real-world phenomena naturally exhibiting graph structures, e.g., social networks, expression networks, and knowledge graphs [16]. A wide spectrum of tasks depends on the analysis and manipulation of such graph data, including anomaly detection [25], molecular structure generation [4, 22], social network analysis [6] and wireless communication systems [12, 29, 31]. These tasks hinge on the ability to efficiently construct node embeddings, which serve as a foundational step in graph processing. The successful development of Graph Neural Networks (GNNs) and their variants has substantially advanced the fields mentioned above. GNNs have demonstrated remarkable capabilities in effectively managing and analyzing graph data. At the core of GNNs lies the fundamental concept of designing effective message-passing and information-aggregation strategies that enable the seamless propagation of information across graph topologies.

Graph data is often noisy, with the central node frequently connected to neighbors that introduce harmful information for downstream tasks. For example, in social networks, the presence of fake users or incorrect connections can cause topology anomalies, leading to irrelevant neighbors being associated with the central node. Unfortunately, state-of-the-art GNNs have exhibited suboptimal performance in fusing node features and topology, leading to the acquisition of inefficient or ineffective node representations [27]. This is because the presence of outlier neighbor nodes dilutes the true underlying information and hinders effective learning [2]. Thus, the development of robust techniques to filter outlier neighbor nodes for each node in GNNs remains a challenging and imperative issue. The ability of GNNs to fuse node features and topology in recent studies is far from optimal or even satisfactory, which will lead to GNNs learning inefficient or even ineffective node representations [27]. A large amount of information gleaned from anomalous neighbors dilutes the true underlying information [2, 19, 27]. Thus, how to filter outlier neighbor nodes for each node in GNNs is a challenging and imperative issue.

To address the aforementioned challenges, we propose a two-fold approach. First, we introduce a novel similarity measure called "node gravity" to measure the similarity between nodes in graph data. Inspired by the concept of gravity in particle physics, we utilize the information entropy of nodes as the masses of particles

and the cosine distance of node features as the distance between particles. By incorporating both the local topology information of the nodes and their features, the node gravity can more effectively express the similarity between nodes. Second, we propose a policy of adaptively selecting the number of abnormal neighbor nodes for filtering according to different nodes. The main contributions can be summarized as follows:

- To effectively measure the similarity between nodes in graph data, we design a novel metric called "node gravity". The metric incorporates both the local topology information between nodes and features, enabling it to capture the underlying similarities between nodes more comprehensively.
- We propose a novel GNN framework called Gravity-GNN, which incorporates the concept of gravity between nodes to improve graph representation learning. Gravity-GNN models the process of node filtering neighbors in GNN training as a Markov Decision Process (MDP), and applies DRL techniques to dynamically optimize the number of sampled nodes for each node in the graph. To the best of our knowledge, Gravity-GNN is the first work to use the concept of space gravity for graph representation learning, while also supporting reinforcement learning.
- Extensive experiments conducted on four real-world datasets clearly demonstrate that Gravity-GNN outperforms current popular GNNs on node classification tasks and achieves state-of-the-art performance. The evaluation across multiple benchmarks verifies the superiority of the proposed Gravity-GNN on node classification tasks, achieving state-of-the-art performance on four real-world datasets.

The remainder of this paper is structured as follows. Section 2 provides an overview of existing research on GNNs, including recent work on reinforcement learning for graph representation. In Section 3, we present the proposed Gravity-GNN framework, along with a theoretical analysis of its key components. Section 4 presents the results of extensive experiments conducted to evaluate the performance of Gravity-GNN. Section 5 concludes this paper.

2 Related work

In this section, we briefly review existing work on GNNs and reinforcement learning on graph representation learning.

2.1 Graph Neural Networks

Since GNNs are well-suited for processing graph-structured data, they have gained widespread popularity recently as an effective approach for graph representation learning. The first model to apply Convolutional Neural Networks (CNNs) directly to graphs was proposed by Kipf *et al.* [7]. Their approach utilizes a local first-order approximation of spectral graph convolution, which leverages a convolutional architecture to learn hidden layer representations that encode both the local graph structure and node features. Dropedge [21] is an approach that randomly drops a certain proportion of edges from the input graph during each training epoch. This technique is used to slow down the convergence rate of over-smoothing and mitigate the resulting loss of information. However, Dropedge ignores the relationships between nodes, which can limit its effectiveness. RioGNN [19] employs a label-aware neural similarity measure to determine the importance of each relationship, allowing it to identify the individual significance of different edges. However, RioGNN only relies on node attributes to identify the importance of nodes, and it does not consider the local topology of nodes. PTDNet [13] is an approach that prunes task-independent edges to improve generalization. This is achieved by penalizing the number of edges in a sparse graph using a parameterized network. In contrast, prior approaches such as [27] have proposed similarity measures based on node attributes, such as cosine similarity, but have ignored the local structural information between nodes. Geom-GCN [18] is an approach that addresses the issue of missing structural information of nodes in communities by employing an information aggregation scheme. This scheme includes node embeddings, structural neighborhoods, and bi-level aggregation.

2.2 DRL-Empowered Graph Neural Networks

DRL has begun to play a pivotal role in applications involving graph data, as it can further exploit the capabilities of DNNs for sequential decision-making with reinforcement learning [15] and improve the applicability of GNNs, e.g., edge computing [24, 28], vaccine supply [10] and job-shop scheduling [11].

For instance, GCPN [30] uses DRL to learn how to generate molecular maps, while Policy-GNN [8] employs DRL to train the original framework for feature learning on nodes with varying numbers of aggregation iterations. CARE-GNN [3] employs Bernoulli Multi-armed Bandit (BMAB) and leverages DRL for fraud detection. Despite the success of the above work to a certain extent, it may still suffer from poor generalization performance. BN-GNN [33] is a brain network representation framework that leverages DRL to automatically determine the optimal number of layers for GNNs, thereby enhancing the performance of traditional GNNs and their performance in brain network analysis tasks. Pairnorm [32] learns k-neighbor subgraphs by restricting the selection of edges to at most k for each node to achieve robust graph representation learning. However, the k-neighbor assumption imposes a limit on the learning ability and could result in poor generalization performance.

2.3 Qualitative Comparison

The Gravity-GNN framework proposed in this paper mainly addresses the issue of GNNs being very sensitive to neighborhood aggregation. Unlike the existing solutions, our approach not only preserves the advantages of GCN but also incorporates the node gravity defined in the graph data, which includes both the local topology information of nodes and the measure of node attribute relationships. In addition, we employ DRL techniques to learn to optimize the threshold for filtering neighbors for different nodes. By doing so, our model can fully exploit the potential most relevant connections between nodes, resulting in enhanced performance.

3 Our Approach

3.1 Preliminary

In this paper, we focus on semi-supervised node classification in attribute graphs $\mathbb{G} = (V, E)$ with $|V|$ nodes, where each node $v \in V$ is represented and edges $(v_i, v_j) \in E$ indicate a connection between nodes v_i and v_j. The adjacency matrix is represented as $A \in \mathbb{R}^{N \times N}$,

where $A_{ij} = 1$ indicates that there is an edge between nodes v_i and v_j. The node features are represented by $X = \{x_1, x_2, \cdots, x_n\} \in \mathbb{R}^{N\times M}$, where M represents the number of features for each node. The degree of each node is denoted by $deg(v_i)$, which represents the sum of all edges connected to node v_i. The degree matrix D is diagonal and composed of $\{deg(v_1), deg(v_2), \cdots, deg(v_i), \cdots, deg(v_n)\}$.

3.1.1 Graph Convolutional Network. The architecture of Graph Convolutional Network (GCN) can be summarized by:

$$z_i^{(l+1)} = \sigma\left(\sum_{v_j\in\mathcal{N}(i)\cup v_i} \frac{1}{\sqrt{deg(v_i)}\sqrt{deg(v_j)}} z_i^{(l)} w^{(l)} + b^{(l)}\right), \quad (1)$$

where $z_i^{(l+1)}$ is the node representation of the $l+1$-th layer for node v_i, σ represents a nonlinear activation function, $deg(v_i)$ is the degree of node v_i with self-loop edges, $w^{(l)}$ is denoted as a parameter matrix, $b^{(l)}$ is the bias of the l-th layer, and $\mathcal{N}(i)$ refers to the set of neighbors of node v_i.

3.1.2 Deep Deterministic Policy Gradient (DDPG). DDPG leverages DNNs to approximate actor-networks $\mu(s, a; \theta^\mu)$ and critic networks $Q(s, a; \theta^Q)$, which estimate policy and Q-value functions, respectively [9]. To improve the stability and speed of the learning process, DDPG employs a dual neural network architecture for both the policy and value functions. This architecture includes target actor-network μ' with parameters $\theta^{\mu'}$ and target critic network Q' with parameters $\theta^{Q'}$.

The policy gradient can be updated by the chain rule [23] as follows:

$$\begin{aligned}\nabla_{\theta_\mu} J &\approx E_{\mu'}[\nabla_{\theta^\mu} Q(s, a; \theta^Q)|_{s=s_t, a=\mu(s_t;\theta^\mu)}] \\ &= E_{\mu'}[\nabla_a Q(s, a; \theta^Q)|_{s=s_t, a=\mu(s_t;\theta^\mu)} \nabla_{\theta^\mu}\mu(s; \theta^\mu)|_{s=s_t}], \quad (2)\end{aligned}$$

which employs the gradient ascent algorithm for optimization calculations to increase the expectation of discounted cumulative rewards:

$$J(\mu) = E_\mu[r_1 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^n r_n], \quad (3)$$

where γ is the reward discount factor.

The critic network is updated by means of updating the value of Deep Q-Network (DQN) [14], and the gradient is expressed as:

$$L(\theta^Q) = E_{\mu'}[(y_t - Q(s_t, a_i; \theta^Q))^2] \quad (4)$$

where $y_t = \gamma Q(s_{i+1}, \mu(s_{t+1}); \theta^Q)$.

3.1.3 Node Classification Loss Function. The node classification loss function is the cross-entropy loss function over all training nodes, which can be defined as follows:

$$\mathcal{L}_{loss} = -\sum_{l\in L}\sum_{i=1}^{C} y_{il} \ln \hat{y}_{il}, \quad (5)$$

where L is the training set, $Y_l = [y_{il}] \in \mathbb{R}^{n\times C}$ is the real label, $\forall l \in L$, and the predicted label is $\hat{Y} = [\hat{y}_{il}] \in \mathbb{R}^{n\times C}$, where $\hat{Y} = softmax(\hat{Z})$ and $\hat{Z}$ is denoted as the output of the last layer.

3.2 Gravity-GNN

Gravity-GNN introduces node gravity and a DRL-based neighbor selection optimizer, enhancing representation learning by capturing node similarities and filtering outlier neighbors. The overall architecture of Gravity-GNN, as illustrated in Fig. 1, involves several steps. Firstly, we sample B nodes from the GNN training set as input, and their similarity scores are calculated based on the node gravity. Then, the node attributes are fed into the DDPG algorithm to obtain the node selection neighbor thresholds for each node in different GNN layers. Next, the nodes perform neighbor samplings based on these thresholds, which are subsequently input to the GNN for training. The DDPG algorithm is updated based on the signal fed back by the GNN training, and this iterative process continues until the GNN training is completed.

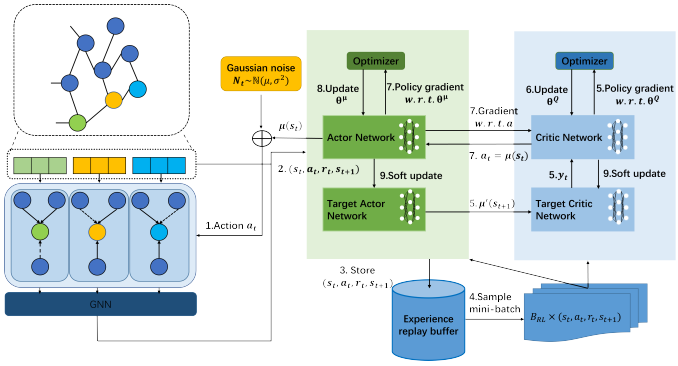


Figure 1: The aggregation process of Gravity-GNN in each training episode.

3.2.1 Node Gravity Measurement. It not only incorporates the impact of the nodes' topological structure, but also considers the relationships between node features.

- **Space Gravity** The magnitude of gravitation is related to the mass of an object and the distance between two particles. The greater the mass of the objects, the greater the gravitational force between them. Additionally, the farther the objects are, the less the gravitational force between them [17].

$$F = G\frac{Mm}{r^2}, \quad (6)$$

where G is the gravitational constant that represents the strength of the gravitational force between two objects. The masses of the two particles are M and m, respectively, and r is the spatial distance between the particles.

- **Node Information Entropy** The information entropy of a node reflects the nature of the interaction between particle masses. The information entropy of each node v_i in the graph $\mathbb{G}$ can be calculated by:

$$E_i = \sum_{v_j\in\Gamma_{v_i}} H_{v_i v_j} = \sum_{v_j\in\Gamma_{v_i}} p_{v_i v_j} \log\frac{1}{p_{v_i v_j}}, \forall v_i \in V, \quad (7)$$

where Γ_{v_i} is the first-order neighbor set of node v_i, $p_{v_i v_j} = \frac{d_j}{\sum_{v_l\in\Gamma_{v_i}} d_l}$ and d_j is the degree of node u_j. In addition, we have $\sum_{v_l\in\Gamma_{v_i}} p_{v_l v_i} = 1$.

- **Node Distance** Similarly, we adopt the cosine distance between node features to characterize the distance between particles. The distance between node v_i and node v_j can be described as:

$$dis(i,j) = 1 - \cos(x_i, x_j) = 1 - \frac{x_i x_j}{|x_i||x_j|}, \tag{8}$$

  where x_i and x_j are node features of nodes v_i and v_j, respectively.
- **Node Gravity** Finally, we define the "node gravity" in graph data as follows:

$$F_{i,j} = G_n \frac{E_i E_j}{dis(i,j)^{\alpha}}, \tag{9}$$

  where G_n is the node gravity constant and α is the weight tuning coefficient that balances node topology information and node attributes. The node gravity metric comprehensively captures node similarities by considering topology and features, resulting in more accurate node embeddings.

3.2.2 Neighbor Selection Optimizer. First, we specify node gravity between the central node and its neighbors, and then compute the similarity score, which is normalized to the range $[0, 1]$. The overall process is detailed in **Algorithm 1**. Next, we design a neighbor selection optimizer to sample the k-nearest neighbors adaptively. We employ the DDPG algorithm to train the GNNs and find the optimal threshold during training. DDPG determines the number and selection of nodes for each node in each GNN layer by providing a threshold for the central node.

Algorithm 1 Node Gravity Metric Score Calculation

Input: Central node v_i and its neighbor set $\mathcal{N}(v_i)$; Node Gravity constant G_n, weight tuning coefficient α
Output: The similarity score between the central node v_i and the neighbor nodes $\mathcal{N}(v_i)$

while $v_j \in v_i \cup \mathcal{N}(v_i)$ **do**
  while $(v_j, v_n) \in \mathbb{E}$ **do**
    $p_{v_j v_n} = \frac{d_{v_n}}{\sum_{v_l \in \Gamma_{v_i}} d_{v_l}}$
  end while
  $E_j = -\sum_{v_n \in \Gamma_{v_j}} p_{v_n v_j} \log p_{v_n v_j}$
end while
while $v_j \in \mathcal{N}(v_i)$ **do**
  $dis(i,j) = 1 - \frac{x_i x_j}{|x_i||x_j|}$
  $F_{i,j} = G_n \frac{E_i E_j}{dis(i,j)^{\alpha}}$
end while
$\max_F = 0, \min_F = 0$
while $v_j \in \mathcal{N}(v_i)$ **do**
  $\max_F = \max(\max_F, F_{i,j})$
  $\min_F = \min(\min_F, F_{i,j})$
end while
while $v_j \in \mathcal{N}(v_i)$ **do**
  $score_{ij} = \frac{F_{i,j} - \min_F}{\max_F - \min_F}$
end while
return score

The next step is to describe how the DRL-based neighbor selection optimizer works, and how to formalize the process of learning the optimal neighbor selection policy as an MDP. As illustrated in Fig. 1, the agent selects actions based on the current state, and the reward is obtained by quantifying the performance of the GNN model on the validation set. This process is repeated until a termination state is reached. The process of nodes filtering neighbors in GNN training can be represented as an MDP, which is characterized by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{T}$ is the state transition function, and $\mathcal{R}$ is the reward given by the GNNs training.

- **State $\mathcal{S}$:** The state $s_t \in \mathcal{S}$ at time step t is defined as the node attribute of the current node. In batch training during the GNNs training process, the state can be expressed as $s_t = [X, Mask_t]$, where $Mask_t \in \mathbb{R}^{N \times 1}$ represents the mask matrix at time step t. Specifically, elements 0 and 1 in $Mask_t$ represent an unsampled node and a sampled node, respectively.
- **Action $\mathcal{A}$:** Denoted as $a_t \in \mathcal{A}$, includes all nodes' neighbor filter thresholds at each layer. Specifically, at each time step t, the action can be expressed as $a_t = [a_0, a_1, \cdots, a_{B-1}] \in \mathbb{R}^{B \times L}$, where $a_i \in [0, 1], i \in [0, B-1]$. Here, B represents the mini-batch size of GNNs and L represents the number of layers in the GNN stacking.
- **State transition $\mathcal{T}$:** Based on the action $a_t \in \mathcal{A}$ generated by the agent, the node features of the next nodes are selected among the neighbors based on the threshold values defined in a_t. Specifically, when the similarity score between the central node and its neighbor exceeds the threshold, among the selected neighbors, those not in the set V_{visited} are chosen with medium probability as the state for the next time slot. In case there are no selected neighbor nodes satisfying the threshold condition, we randomly sample nodes from the training set, and take their node features as the state in the next time slot.
- **Reward $\mathcal{R}$:** we set the reward function as:

$$\mathcal{R} = \omega\Big(P(s_t, a_t) - \frac{1}{b-1} \sum_{i=t-b}^{t-1} P(s_i, a_i)\Big), \tag{10}$$

  where $P(s_t, a_t)$ is the performance evaluation metric on the validation set during the GNNs training process, and ω is a parameter that adjusts the sensitivity of the agent to changes in the performance of GNNs. Specifically, a larger value of ω means the agent is more sensitive to performance changes. Moreover, we include a wait-and-see period of b steps between the current training performance and the historical performance.
- **Termination condition:** We define the termination condition of an episode as the completion of a full traversal of the training set during the GNNs training process. This is achieved when the set of visited state indices $T_{visited}$ is a superset of T_t, where T_t represents the current training set being used.

As illustrated in Fig. 2, the dotted line represents the entire training set node. Initially, we randomly sample nodes from the training set as the starting states. Then, based on the action selected

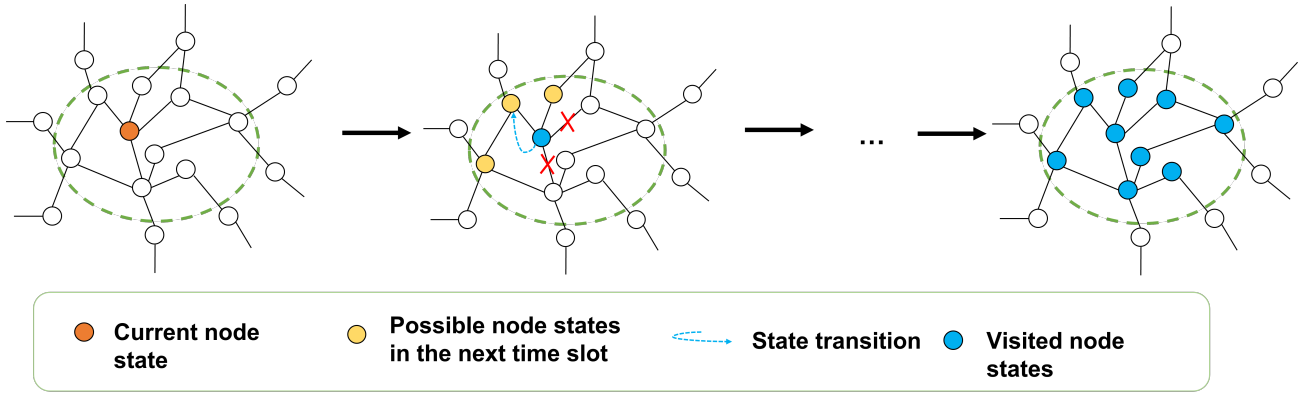


Figure 2: Schematic diagram of the state transition process.

by the neighbor selection optimizer, we choose the state node for the next time slot randomly among the selected neighbors. This process continues until all the nodes in the training set have been visited, serving as the termination condition for the episode.

3.2.3 Neighbor Selection Optimizer on DDPG. Due to the high-dimensional state space and continuous action space, we design a DDPG-based neighbor selection optimizer algorithm, as shown in **Algorithm 2**.

First, we initialize the online actor network $\mu(s, a; \theta^{\mu})$ and the online critic network $Q(s, a; \theta^{Q})$ with the parameters θ^{μ} and θ^{Q}, respectively. The online network parameters are then assigned to their corresponding target network parameters, i.e., $\theta^{Q'} \longleftarrow \theta^{Q}$ and $\theta^{\mu'} \longleftarrow \theta^{\mu}$. Additionally, we initialize the GNN and record the indices of the training state set $T_{visited}$. During a single experience trajectory episode, an action a_t is generated by adding behavioral noise N_t at time step t, as follows:

$$a_t = \mu(s, a : \theta^{\mu}) + N_t, \tag{11}$$

where N_t samples the Gaussian distribution $N_t \sim \mathbb{N}(\mu_n, \sigma_n^2)$, μ_n is the mean and σ_n^2 is the variance.

The GNN is then trained on the action a_t produced by the agent, which returns rewards r_t and new states s_{t+1}. The agent then stores the generated tuple of information (s_t, a_t, r_t, s_{t+1}) in R as training samples for the online network. Subsequently, the agent randomly samples B_{RL} (s_t, a_t, r_t, s_{t+1}) from R as mini-batch training data. The actor target network μ' outputs actions $\mu'(s_{t+1})$, and y_i is calculated as $y_t = \gamma Q(s_{i+1}, \mu(s_{t+1}); \theta^{Q})$. The critic network's parameter θ^{Q} is updated using the stochastic gradient descent algorithm. Then, the actor-network $\mu(s, a; \theta^{\mu})$ is updated as follows:

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N} \sum_{i} \nabla_a Q(s, a; \theta_Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^{\mu}} \mu(s; \theta^{\mu})|_{s_t}. \tag{12}$$

Finally, DDPG employs a soft update method to update the target actor-network μ' and critic network Q' as follows:

$$\theta^{Q'} \longleftarrow \tau\theta^{Q'} + (1-\tau)\theta^{Q}, \tag{13}$$

$$\theta^{\mu'} \longleftarrow \tau\theta^{\mu'} + (1-\tau)\theta^{\mu}, \tag{14}$$

where the default value of the soft update factor τ is 0.001.

3.2.4 GNN Message Passing Module. After sampling the neighboring nodes, the subsequent step is to aggregate the information from these nodes. We assume that the most effective neighboring nodes have been selected for a specific downstream task. To retain the benefits of traditional GCN, we adopt the node message passing scheme as follows.

$$z_i^{(l+1)} = \sigma\left(\sum_{j \in \hat{\mathcal{N}}(i) \cup v_i} \frac{1}{\sqrt{\hat{deg}(v_i)}\sqrt{\hat{deg}(v_j)}} z_i^{(l)} w^{(l)} + b^{(l)} \right), \tag{15}$$

where $z_i^{(l+1)}$ is the node representation of the $l+1$-th layer for node v_i, σ represents a nonlinear activation function, $\hat{deg}(v_i)$ is degree of sampled node v_i with self-loop edges, $w^{(l)}$ is denoted as a parameter matrix, $b^{(l)}$ is the bias of the l-th layer and $\hat{\mathcal{N}}(i)$ refers to the set of the sampled neighbor nodes.

3.3 Time Complexity Analysis of Gravity-GNN

Our proposed Gravity-GNN demands calculating the feature mapping of all nodes and node gravitation between nodes and their neighbors. As shown in **Algorithm 1**, there are two loops (line 1 and line 2) to calculate the gravitational similarity score between the central node and the neighbor node. Therefore, the time complexity of **Algorithm 1** is $O(d_m^2)$, and d_m is the average degree of the node. Furthermore, as shown in **Algorithm 2**, the Gravity-GNN's message passing scheme has a complexity of $O(|\mathbb{V}|^2)$ due to matrix multiplication, where $|\mathbb{V}|$ is the number of nodes in the graph. The time complexity of the cross-entropy loss function is also linear and $O(|\mathbb{V}|)$. Therefore, the overall time complexity of the proposed Gravity-GNN is $O(|\mathbb{V}|^2 + d_m^2)$.

3.4 Convergence Analysis of Gravity-GNN

In this section, we will analyze the convergence of the proposed algorithm, and the proof heavily refers to [20], which provides a set of practical sufficient conditions for the convergence of the single-agent DDPG algorithm in the continuous action spaces.

Theorem 1. *Given the following assumptions:*

(1) The critic step-size sequence $\alpha(t) > 0, \forall t \geq 0$. Further, the sequence is monotonically decreasing and satisfies $\sum_{t \geq 0} \alpha(t) =$

Algorithm 2 Neighbor Selection Optimizer based on DDPG

Input: Number of training episode E; Training set T_t of GNNs, Critic network learning rate α_{Critic}; Actor-network learning rate α_{Critic}; Experience relay buffer R; Mini-batch size B_{RL}; Train set in GNNs training T_t

1: Initialize the weight parameters θ^{μ} and θ^{Q} of actor-network $\mu(s,a;\theta^{\mu})$ and critic network $Q(s,a;\theta^{Q})$, respectively.
2: Initialize the weight parameters $\theta^{Q^{'}} \longleftarrow \theta^{Q}$ and $\theta^{\mu^{'}} \longleftarrow \theta^{\mu}$ of target critic network $Q^{'}$ and target actor-network $\mu^{'}$, respectively.
3: Empty experience relay buffer R
4: **while** each $episode = 1, 2, \cdots, E$ **do**
5: Reset the initial state s_0 of training the GNNs algorithm.
6: Reset the visited state index set $T_{visited}$, and add the index of the state s_0 to $T_{visited}$.
7: **while** $T_{visited} \not\supset T_t$ **do**
8: Calculate the action $a_t = \mu(s, a : \theta^{\mu}) + N_t$ at the current time step based on the current policy $\mu(s, a : \theta^{\mu})$ with noise.
9: Perform actions a_t, train the GNNs algorithm, and record the reward r_t and the state s_{t+1} of the next time step.
10: Store transfer tuple (s_t, a_t, r_t, s_{t+1}) in experience relay buffer R.
11: Update $T_{visited}$, adding the index of the state s_{t+1} to $T_{visited}$.
12: Randomly sample mini-batches B_{RL} of transitions (s_t, a_t, r_t, s_{t+1}) from R.
13: $y_i = r_i + Q'(s_{t+1}, \mu'(s_{t+1};\theta^{\mu'});\theta^{Q'})$
14: Minimizing the loss to update the critic network $Q(s, a : \theta^{Q})$.
15: Update actor-networks using gradient policy algorithm.
16:

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_{i} \nabla_a Q(s,a;\theta_Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^{\mu}} \mu(s;\theta^{\mu})|_{s_t}$$

17: Soft update the target network via Eqs. (13) and (14).
18: **end while**
19: **end while**

∞, $\sum_{t\geq 0} \alpha^2(t) < \infty$. *The actor step-size sequence* $\{\beta(t)\}_{t\geq 0}$ *satisfies* $\lim_{t\to\infty} \frac{\beta(t)}{\alpha(t)} = 1$.

(2) (a) $sup_{t\geq 0}\|\theta_t^{Q}\| < \infty \quad a.s.$
(b) $sup_{t\geq 0}\|\theta_t^{\mu}\| < \infty \quad a.s.$

(3) *The state transition kernel* $p(\cdot \mid s, a)$ *is continuous.*

Then the limit $(\bar{\theta}_{\infty}^{Q}, \bar{\theta}_{\infty}^{\mu})$ *of the algorithm satisfies:* $\widetilde{\nabla} L(\bar{\theta}_{\infty}^{Q}, \bar{\theta}_{\infty}^{\mu}, \mu_{1\infty}) = 0$ *and* $\widetilde{\nabla} J\big(\bar{\theta}_{\infty}^{Q}, \bar{\theta}_{\infty}^{\mu}, \mu_{1\infty}^{s}\big) = 0$.

Proof. In the algorithm, the policies μ and critics Q are fully connected feedforward neural networks that possess twice continuously differentiable activation functions. The reward function $r\colon \mathcal{S}\times\mathcal{A} \to \mathbb{R}$ is continuous.

When we do not consider experience replays, i.e., $B_{RL} = 1$, the parameters of θ^{Q} and θ^{μ} are iteratively updated as follows:

$$\theta_{t+1}^{Q} = \theta_t^{Q} + \alpha(t)\nabla_{\theta^{Q}} L(\theta_t^{Q}, \theta_t^{\mu}, s_t, a_t), \tag{16}$$

$$\theta_{t+1}^{\mu} = \theta_t^{\mu} + \beta(t)\nabla_{\theta^{\mu}} J(\theta_t^{Q}, \theta_t^{\mu}, s_t). \tag{17}$$

The loss gradient of Eq. (16) is given by

$$\begin{aligned}\nabla_{\theta^{Q}} L(\theta_t^{Q}, \theta_t^{\mu}, s_t, a_t) = \nabla_{\theta^{Q}} Q\big(s_t, a_t; \theta_t^{Q}\big)\Big(r(s_t, a_t) + \\ \gamma \int Q(y, \mu(y;\theta_t^{\mu});\theta_t^{Q}) p(dy \mid s_t, a_t) - Q\Big(s_t, a_t; \theta_t^{\mu}\Big)\Big).\end{aligned} \tag{18}$$

The loss gradient of Eq. (17) is given by

$$\begin{aligned}&\nabla_{\theta^{\mu}} J(\theta_t^{Q}, \theta_t^{\mu}, s_t) \\ &= \nabla_{\theta^{\mu}} \mu\big(s_t;\theta_t^{\mu}\big) \times \nabla_a Q\big(s_t, \mu\big(s_t;\theta_t^{\mu}\big);\theta_t^{Q}\big).\end{aligned} \tag{19}$$

We then construct continuous-time trajectories that have identical limiting behavior and downgrade the analysis to understand the asymptotic behavior of these trajectories.

First, we utilize the given step size sequence to divide the time axis as follows:

$$m_0 = 0 \text{ and } m_t = \textstyle\sum_{n=0}^{t-1} \alpha(n), \forall t \geq 1.$$

We define $\bar{\theta}^{Q} \in C([0,\infty), \mathbb{R}^q)$ and $\bar{\theta}^{\mu} \in C([0,\infty), \mathbb{R}^p)$ as follows:

(1) $\bar{\theta}^{Q}(m_t) = \theta_t^{Q}$ and $\bar{\theta}^{\mu}(m_t) = \theta_t^{\mu}, \forall t \geq 0$,
(2) $\bar{\theta}^{Q}(m) = \bar{\theta}^{Q}(m_t) + \frac{m-m_t}{m_{t+1}-m_t}[\bar{\theta}^{Q}(m_{t+1}) - \bar{\theta}^{Q}(m_t)], \forall m \in (m_t, m_{t+1})$ and $m \geq 0$,
(3) $\bar{\theta}^{\mu}(m) = \bar{\theta}^{\mu}(m_t) + \frac{m-m_t}{m_{t+1}-m_t}[\bar{\theta}^{\mu}(m_{t+1}) - \bar{\theta}^{\mu}(m_t)], \forall m \in (m_t, m_{t+1})$ and $m \geq 0$,

where $\mathbb{R}^q$ and $\mathbb{R}^p$ respectively are the parameter spaces of θ_t^{Q} and θ_t^{μ}.

Similar to [20], we define a process of probability measures on $\mathcal{S}\times\mathcal{A}$ by

$$\mu_1(m) = \delta(s_t, a_t), \tag{20}$$

and the associated process on $\mathcal{S}$ by

$$\mu_1^{s}(m) = \mu_1(m)(\cdot, \mathcal{A}) = \delta_{s_t}, \tag{21}$$

where $m \in (m_t, m_{t+1})$, $\delta(s_t, a_t)$ and δ_{s_t} denotes the Dirac measure.

Then, we define

$$\widetilde{\nabla} L(\theta^{Q}, \theta^{\mu}, v) = \int \nabla_{\theta^{Q}} L(\theta^{Q}, \theta^{\mu}, s, a) v(ds, da), \tag{22}$$

$$\widetilde{\nabla} J(\theta^{Q}, \theta^{\mu}, v^{s}) = \int \nabla_{\theta^{\mu}} J(\theta^{Q}, \theta^{\mu}, s) v^{s}(ds), \tag{23}$$

where v is the probability measure on $\mathcal{S}\times\mathcal{A}$ and v^{s} is the associated marginal measure on $\mathcal{S}$.

According to [20], under the above assumptions, the limits $(\bar{\theta}_{\infty}^{Q}, \bar{\theta}_{\infty}^{\mu})$ of the algorithm satisfy $\widetilde{\nabla} L(\bar{\theta}_{\infty}^{Q}, \bar{\theta}_{\infty}^{\mu}, \mu_{1\infty}) = 0$ and $\widetilde{\nabla} J\big(\bar{\theta}_{\infty}^{Q}, \bar{\theta}_{\infty}^{\mu}, \mu_{1\infty}^{s}\big) = 0$.

The following analyzes the situation when experience replays are included, which means that in every iteration t, instead of only using the tuple $(s_t,\ a_t,\ r_t,\ s_{t+1})$ for training, the tuple encountered is stored. At time step T, the agent randomly samples a mini-batch

of size $B_{RL} < B$ tuples from the experience replay, and the weights are updated as follows:

$$\theta_{t+1}^{Q} = \theta_t^{Q} + \alpha(t)\left[\frac{1}{B_{RL}}\sum_{i=1}^{B_{RL}} \nabla_{\theta^Q} L(\theta_t^{Q}, \theta_t^{\mu}, s_{k(t,i)}, a_{k(t,i)})\right], \quad (24)$$

$$\theta_{t+1}^{\mu} = \theta_t^{\mu} + \beta(t)\left[\frac{1}{B_{RL}}\sum_{i=1}^{B_{RL}} \nabla_{\theta^\mu} J(\theta_t^{Q}, \theta_t^{\mu}, s_{k(t,i)})\right], \quad (25)$$

where $T - B + 1 \leq k(t,i) \leq T$.

Next, for $m \in [m_t, m_{t+1})$, we redefine $\mu_1(m)$ to be the probability measure on $\mathcal{S} \times \mathcal{A}$ that places a mass of $1/B_{RL}$ on $\left(s_{k(t,i)}, a_{k(t,i)}\right)$ for $1 \leq i \leq B_{RL}$.

For $m = m_t$,

$$\begin{aligned}
&\widetilde{\nabla} L\left(\bar{\theta}^{Q}(m), \bar{\theta}^{\mu}(m), \mu_1(m)\right) \\
&= \int \nabla_{\theta^Q} L(\bar{\theta}^{Q}(m), \bar{\theta}^{\mu}(m), s, a)\mu_1(m) \\
&= \frac{1}{B_{RL}} \Sigma_{i=1}^{B_{RL}} \nabla_{\theta^Q} L\left(\theta_t^{Q}, \theta_t^{\mu}, s_{k(t,i)}, a_{k(t,i)}\right). \quad (26)
\end{aligned}$$

Similarly,

$$\begin{aligned}
&\widetilde{\nabla} J\left(\bar{\theta}^{Q}(m), \bar{\theta}^{\mu}(m), \mu_1^{s}(m)\right) \\
&= \int \nabla_{\theta^\mu} J(\bar{\theta}^{Q}(m), \bar{\theta}^{\mu}(m), s)\mu_1^{s}(m) \\
&= \frac{1}{B_{RL}} \Sigma_{i=1}^{B_{RL}} \nabla_{\theta^\mu} J\left(\theta_t^{Q}, \theta_t^{\mu}, s_{k(t,i)}\right). \quad (27)
\end{aligned}$$

It can be shown that the analysis follows the same lines as the redefined measure process and see [20] for details.

□

4 Performance Evaluation

In order to verify the effectiveness of Gravity-GNN, different datasets and baselines are used for the experiment. In particular, we adopt a semi-supervised node classification task. First, we introduce some details about the datasets. Then we list the comparative baselines and some variants of Gravity-GNN.

4.1 Experimental Setting

4.1.1 Datasets. Extensive experiments are conducted on widely used real-world citation networks, which are listed as follows:

- **Cora, Citeseer and Pubmed** [7]: These datasets belong to the Academic Paper Citation Network, where nodes represent publications and edges represent citation links. Each document is assigned a unique tag based on its subject, and node attributes are represented as a bag of words. All nodes are classified into 7, 6, and 3 classes, respectively.
- **ACM** [26]: The network is drawn from the ACM collection, where nodes represent papers and edges represent the co-authorship relationship between them. Node features are constructed based on the keywords of the papers, and all papers are divided into three categories: database, wireless communication, and data mining.

In addition, other details of the above datasets are summarized in Table 1.

Table 1: Statistical characteristics.

Datasets	Cora	Citeseer	PubMed	ACM
Classes	7	6	3	3
Feature	1,433	3,703	500	1,870
Nodes	2,708	3,327	19,717	3,025
Edges	5,429	4,732	44,338	13,138
Training	140	120	60	60
validation	500	500	500	500
Test	1,000	1,000	1,000	1,000

4.1.2 Baselines. We employ the most popular GNNs as benchmarks for comparison, and the specific details are listed below.

- **GCN** [7]: It aims to learn the encoding of local graph structures and node feature representations by utilizing local first-order approximations of spectral graph convolutions. This approach leverages the inherent properties of the spectral graph convolution, allowing for more efficient and scalable computation of the graph convolutional operation.
- **FastGCN** [1]: It interprets graph convolution as an integral transformation of the embedding function under probabilistic measures. It enhances the performance of the model by leveraging importance sampling augmentation. This enables the model to encode graph structures and node feature representations more effectively.
- **Dropedge-GCN** [21]: It aims to prevent the model from over-smoothing and enhance generalization by removing a fixed proportion of edges at each training iteration.
- **AM-GCN** [27]: It is an adaptive multi-channel GCN and applies an attention mechanism to simultaneously extract node embeddings from node attributes, topology, and their combinations.
- **PTDNet-GCN** [13]: It improves the robustness and generalization performance of GNNs by utilizing a parameterized topological denoising network, which learns to discard task-irrelevant edges. This helps prevent the GNN from being affected by noisy or irrelevant information in the graph.

4.1.3 Variants of Gravity-GNN. We explore the effects of the three key modules in Gravity-GNN by implementing a series of variants. Specifically, we modify the node gravity measure and neighbor selection optimizer to investigate their impact on the model's performance. The variants are designed to examine the crucial mechanisms of the two modules.

- **Cosine-GNN**: The model employs a straightforward approach that involves utilizing the cosine similarity of node features and the DDPG algorithm to select and filter the number of neighbor nodes.
- **Gravity-GNN-R**: The proposed node gravity is utilized to measure node similarity, and the threshold for selecting neighbor nodes is determined by randomly generating a number. Specifically, a random number is generated in the range of [0, 1] for each node, which is used as the threshold for selecting neighboring nodes.

- **Gravity-GNN-H**: The model utilizes the novel node similarity measure "node gravity" to measure similarity and selects thresholds from a range of $[0.1, 1]$ as hyperparameters.
- **Gravity-GNN-TD3**: The model employs TD3 [5] as the node neighbor optimizer, while the remaining modules use the same settings as those of Gravity-GNN.

To comprehensively evaluate our proposed Gravity-GNN, we adopted the parameters suggested in the original paper for all baselines and further adjusted the hyperparameters to achieve better performance. Specifically, we set the number of layers of all GCNs to 2 and the hidden layer units to 128. The learning rate was set within the range of $\{0.01, 0.02, 0.03, 0.04, 0.05\}$, and the dropout rate was set to 0.4, 0.5. The node gravitational constant is generally set to 1 by default for simplicity. Furthermore, we set the hidden layer units of MLP to $128, 256$ and the learning rate as {1e-4, 1e-3} in DDPG. The experience replay buffer capacity was set to 10,000, and other parameters were adjusted based on the original paper.

4.2 Performance Comparison

As presented in Table 2, our proposed Gravity-GNN consistently outperforms both baselines and variants in terms of node classification accuracy. The comparisons demonstrate the superiority of our model for node classification. It can be observed that Gravity-GNN and its variants achieve improvements ranging from 0.58% to 2.04%, 1.49% to 4.29%, 0.86% to 1.93%, and 2.9% to 5.56% over currently popular methods on the four datasets, respectively. Gravity-GNN-R and Gravity-GNN-H are inferior to our adaptive neighbor selection method Gravity-GNN, which proves the superiority of our DRL-based Neighbor Selection Optimizer. The performance improvement over the baselines justifies the necessity of our proposed neighbor selection optimizer modeled as an MDP. Moreover, the comparison between Cosine-GNN and Gravity-GNN verifies that the node gravity metric can more effectively express the similar relationship between node features and topological information.

Table 2: Node classification accuracy (%) (Bold: best)

Methods	Cora	Citeseer	PubMed	ACM
GCN	80.68	70.44	79.12	85.84
FastGCN	78.57	69.74	79.09	86.18
Dropedge-GCN	80.85	70.97	78.74	84.46
AM-GCN	76.91	69.44	79.58	88.56
PTDNet-GCN	82.14	72.54	79.81	87.12
Cosine-GNN	81.74	72.18	79.70	86.78
Gravity-GNN-R	77.31	69.46	78.62	84.38
Gravity-GNN-H	78.45	70.82	79.66	85.91
Gravity-GNN-TD3	81.31	73.34	**80.67**	87.03
Gravity-GNN	**82.72**	**74.03**	79.89	**90.02**

We evaluated the robustness of Gravity-GNN on the ACM dataset by randomly introducing noisy edges into the original graph. For instance, with a perturbation ratio of 5%, 50 edges would be randomly added to a dataset with 1,000 original edges. As shown in Fig. 3, our proposed model exhibited remarkable robust performance under different levels of perturbation. This resilience is attributed to its node gravity measure, which effectively captures both node features and local topology, along with the neighbor selection optimizer based on DDPG. In contrast, the traditional GCN model suffers from dilution of potential value information due to the presence of noisy edges. Other baselines, such as FastGCN and Dropedge-GCN, lack effective node similarity measures and adaptive neighbor selection strategies, making them less capable of handling noisy edges. Cosine-GNN, which relies solely on the cosine similarity of node features for neighbor selection, proved less robust than Gravity-GNN due to its absence of topological information.

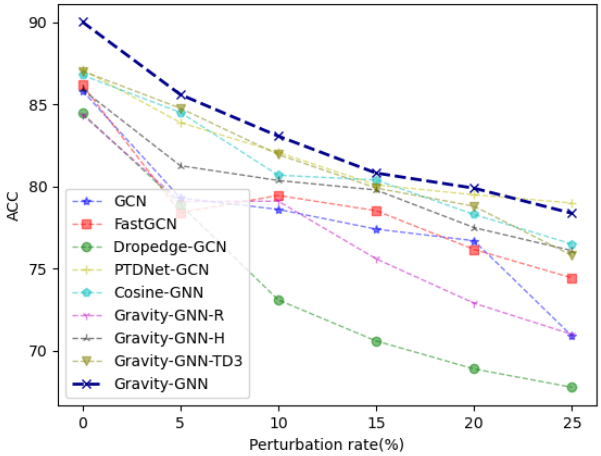
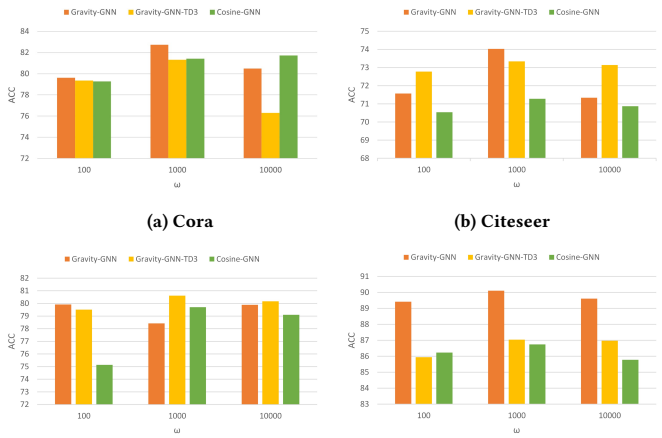


Figure 3: The results (%) of Gravity-GNN and variants compared to baselines at different perturbation rates on ACM.

The variants Gravity-GNN-R and Gravity-GNN-H, which employ random assignment and traditional hyperparameter assignment methods for setting neighbor selection thresholds, respectively, were also found to be inferior to Gravity-GNN with the DDPG-based neighbor selection optimizer. In addition, while PTDNet-GCN uses a parameterized topology denoising network to filter out task-irrelevant edges, it generally underperformed compared to Gravity-GNN, except at a 25% perturbation rate on the ACM dataset. Overall, Gravity-GNN consistently achieved the highest classification accuracy compared to all baseline algorithms and variants, except for a slight dip in performance at the 25% perturbation level on the ACM dataset, where it ranked second best.

4.3 Hyperparameter Sensitivity

4.3.1 Impact of hyperparameter ω. We conducted an experiment to assess the impact of the hyperparameter ω on the performance of the Gravity-GNN model. As previously mentioned, ω controls the sensitivity of the model to performance variations, with larger values making the agent more sensitive to changes in the GNN's performance. To evaluate the effect of ω, we varied its value at values of 100, 1,000, and 10,000, and measured the corresponding accuracy percentages. As shown in Fig. 4, the accuracy initially increases but subsequently decreases as ω becomes larger. This suggests that heightened sensitivity to feedback does not necessarily lead to better performance. Based on our findings, we recommend setting ω to 1,000 to achieve optimal results.

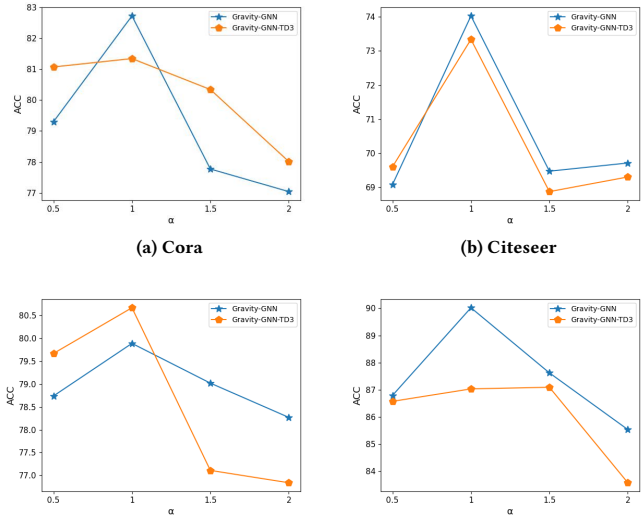


(a) Cora (b) Citeseer

(c) PubMed (d) ACM

Figure 4: The effect of hyperparameter ω on the experimental results (%) in different datasets.

4.3.2 Impact of hyperparameter α. We evaluated the effect of the hyperparameter α on the accuracy across different datasets. Specifically, α balances the influence of node topology information and node attributes. We varied α from 0.5 to 2 and recorded the resulting accuracy changes, as illustrated in Fig. 5. The results indicate that accuracy initially increases with α but begins to decline as α continues to increase. Based on these observations, setting α to 1 typically yields the best performance.

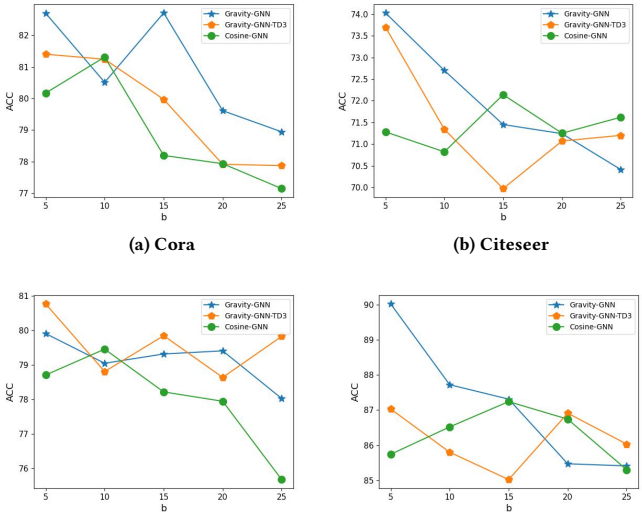


(a) Cora (b) Citeseer

(c) PubMed (d) ACM

Figure 5: The effect of hyperparameter α on the experimental results (%) in different datasets.

4.3.3 Impact of hyperparameter b. We conducted experiments to evaluate the impact of the hyperparameter b on the classification accuracy across different datasets. Fig. 6 shows the results, where b denotes the window size used to track historical performance changes of GNNs during training. We varied b from 5 to 25 and observed the corresponding accuracy changes. Generally, an increasing trend in b values led to varied effects on accuracy, except for the PubMed dataset, where a consistent decline in classification accuracy was noted as b increased.



(a) Cora (b) Citeseer

(c) PubMed (d) ACM

Figure 6: The effect of hyperparameter b on the experimental results (%) in different datasets.

5 Conclusion

In this paper, we introduce a novel GNN framework that effectively reduces noise interference by leveraging both node topology information and node attributes. To achieve this, we define a new node similarity measure, called "node gravity", which applies classical mechanics from physics to graph data. We combine the node gravity metric and the DRL-based neighbor selection optimizer into the Gravity-GNN framework, improving graph representation learning and achieving state-of-the-art performance in node classification tasks. In addition, we apply the concept of gravity from classical mechanics to GNNs, offering a new perspective on graph data analysis and representation learning. Specifically, we model the process of node filtering neighbors in GNN training as an MDP and utilize DRL to generate a threshold for filtering neighbors for each node, which enhances the model's generalization ability. We conduct extensive experiments on real-world datasets to evaluate the superior performance of our Gravity-GNN against state-of-the-art models. Our results demonstrate the potential robustness against noisy data with varying quality, and the generalizability of Gravity-GNN across various network settings.

Acknowledgments

This work was supported by the National Key Research and Development Program of China (31400), the National Natural Science Foundation of China (62071327 and 62401190), the Emerging Frontiers Cultivation Program of Tianjin University Interdisciplinary Center, and the UK EPSRC CHEDDAR Communications Hub (EP/X040518/1 and EP/Y037421/1).

GenAI Usage Disclosure

The authors disclose that no Generative AI (GenAI) tools were used at any stage of the research, including data collection, code development, or manuscript writing.

References

[1] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=rytstxWAW

[2] Hande Dong, Jiawei Chen, Fuli Feng, Xiangnan He, Shuxian Bi, Zhaolin Ding, and Peng Cui. 2021. On the Equivalence of Decoupled Graph Convolution Network and Label Propagation. In *Proceedings of the Web Conference 2021* (Ljubljana, Slovenia) *(WWW '21)*. ACM, New York, NY, USA, 3651–3662. doi:10.1145/3442381.3449927

[3] Yingtong Dou, Zhiwei Liu, Li Sun, Yutong Deng, Hao Peng, and Philip S. Yu. 2020. Enhancing Graph Neural Network-based Fraud Detectors against Camouflaged Fraudsters. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management* (Virtual Event, Ireland) *(CIKM '20)*. ACM, New York, NY, USA, 315–324. doi:10.1145/3340531.3411903

[4] Joshua Dean Ellis, Razib Iqbal, and Keiichi Yoshimatsu. 2024. Deep Q-Learning-Based Molecular Graph Generation for Chemical Structure Prediction From Infrared Spectra. *IEEE Trans. Artif. Intell.* 5, 2 (2024), 634–646. doi:10.1109/TAI.2023.3287947

[5] Scott Fujimoto, Herke van Hoof, and David Meger. 2018. Addressing Function Approximation Error in Actor-Critic Methods. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018 (Proceedings of Machine Learning Research, Vol. 80)*. PMLR, 1582–1591. http://proceedings.mlr.press/v80/fujimoto18a.html

[6] Mengzhou Gao, Pengfei Jiao, Ruili Lu, Huaming Wu, Yinghui Wang, and Zhidong Zhao. 2024. Inductive Link Prediction via Interactive Learning Across Relations in Multiplex Networks. *IEEE Trans. Comput. Soc. Syst.* 11, 3 (2024), 3118–3130. doi:10.1109/TCSS.2022.3176928

[7] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=SJU4ayYgl

[8] Kwei-Herng Lai, Daochen Zha, Kaixiong Zhou, and Xia Hu. 2020. Policy-GNN: Aggregation Optimization for Graph Neural Networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Virtual Event, CA, USA) *(KDD '20)*. ACM, New York, NY, USA, 461–471. doi:10.1145/3394486.3403088

[9] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. http://arxiv.org/abs/1509.02971

[10] Lu Ling, Washim Uddin Mondal, and Satish V. Ukkusuri. 2024. Cooperating Graph Neural Networks With Deep Reinforcement Learning for Vaccine Prioritization. *IEEE J. Biomed. Health Inform.* 28, 8 (2024), 4891–4902. doi:10.1109/JBHI.2024.3392436

[11] Chien-Liang Liu and Tzu-Hsuan Huang. 2023. Dynamic Job-Shop Scheduling Problems Using Graph Neural Network and Deep Reinforcement Learning. *IEEE Trans. Syst., Man, Cybern. Syst.* 53, 11 (2023), 6836–6848. doi:10.1109/TSMC.2023.3287655

[12] Mengbing Liu, Chongwen Huang, Ahmed Alhammadi, Marco Di Renzo, Mérouane Debbah, and Chau Yuen. 2025. Beamforming Design and Association Scheme for Multi-RIS Multi-User mmWave Systems Through Graph Neural Networks. *IEEE Trans. Wireless Commun.* (2025), 1–1. doi:10.1109/TWC.2025.3563529

[13] Dongsheng Luo, Wei Cheng, Wenchao Yu, Bo Zong, Jingchao Ni, Haifeng Chen, and Xiang Zhang. 2021. Learning to Drop: Robust Graph Neural Network via Topological Denoising. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining* (Virtual Event, Israel) *(WSDM '21)*. ACM, New York, NY, USA, 779–787. doi:10.1145/3437963.3441734

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (Feb. 2015), 529–533. doi:10.1038/nature14236

[15] Sai Munikoti, Deepesh Agarwal, Laya Das, Mahantesh Halappanavar, and Balasubramaniam Natarajan. 2024. Challenges and Opportunities in Deep Reinforcement Learning With Graph Neural Networks: A Comprehensive Review of Algorithms and Applications. *IEEE Trans. Neural Netw. Learn. Syst.* 35, 11 (2024), 15051–15071. doi:10.1109/TNNLS.2023.3283523

[16] Jingchao Ni, Shiyu Chang, Xiao Liu, Wei Cheng, Haifeng Chen, Dongkuan Xu, and Xiang Zhang. 2018. Co-Regularized Deep Multi-Network Embedding. In *Proceedings of the 2018 World Wide Web Conference* (Lyon, France) *(WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 469–478. doi:10.1145/3178876.3186113

[17] Hans C. Ohanian and Remo Ruffini. 2013. *Gravitation and Spacetime*. Cambridge University Press. doi:10.1017/cbo9781139003391

[18] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-GCN: Geometric Graph Convolutional Networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=S1e2agrFvS

[19] Hao Peng, Ruitong Zhang, Yingtong Dou, Renyu Yang, Jingyi Zhang, and Philip S. Yu. 2021. Reinforced Neighborhood Selection Guided Multi-Relational Graph Neural Networks. *ACM Trans. Inf. Syst.* 40, 4, Article 69 (Dec. 2021), 46 pages. doi:10.1145/3490181

[20] Adrian Redder, Arunselvan Ramaswamy, and Holger Karl. 2022. 3DPG: Distributed Deep Deterministic Policy Gradient Algorithms for Networked Multi-Agent Systems. arXiv:2201.00570

[21] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2020. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=Hkx1qkrKPr

[22] Mallikharjuna Rao Sakhamuri, Shagufta Henna, Leo Creedon, and Kevin Meehan. 2024. Molecular Adversarial Generative Graph Network Model for Large-scale Molecular Networks. In *2024 35th Irish Signals and Systems Conference (ISSC)*. 01–06. doi:10.1109/ISSC61953.2024.10603046

[23] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic Policy Gradient Algorithms. In *Proceedings of the 31st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 32)*. PMLR, Bejing, China, 387–395. https://proceedings.mlr.press/v32/silver14.html

[24] Guangchen Wang, Peng Cheng, Zhuo Chen, Branka Vucetic, and Yonghui Li. 2024. Inverse Reinforcement Learning With Graph Neural Networks for Full-Dimensional Task Offloading in Edge Computing. *IEEE Trans. Mob. Comput.* 23, 6 (2024), 6490–6507. doi:10.1109/TMC.2023.3324332

[25] Kai Wang, Shuaiyi Lyu, Yang Liu, and Bailing Wang. 2025. Graph Optimization Via Decoupled Edge Tuning for Efficient Industrial Anomaly Detection. *IEEE Trans. Netw. Sci. Eng.* (2025), 1–16. doi:10.1109/TNSE.2025.3567671

[26] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous Graph Attention Network. In *The World Wide Web Conference* (San Francisco, CA, USA) *(WWW '19)*. ACM, New York, NY, USA, 2022–2032. doi:10.1145/3308558.3313562

[27] Xiao Wang, Meiqi Zhu, Deyu Bo, Peng Cui, Chuan Shi, and Jian Pei. 2020. AM-GCN: Adaptive Multi-channel Graph Convolutional Networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Virtual Event, CA, USA) *(KDD '20)*. ACM, New York, NY, USA, 1243–1253. doi:10.1145/3394486.3403177

[28] Yixiao Wang, Huaming Wu, and Ruidong Li. 2024. Deep Graph Reinforcement Learning for Mobile Edge Computing: Challenges and Solutions. *IEEE Network* 38, 5 (2024), 314–323. doi:10.1109/MNET.2024.3383242

[29] Yuzhi Yang, Zhaoyang Zhang, Yuqing Tian, Richeng Jin, and Chongwen Huang. 2023. Implementing Graph Neural Networks Over Wireless Networks via Over-the-Air Computing: A Joint Communication and Computation Framework. *IEEE Wireless Commun.* 30, 3 (2023), 62–69. doi:10.1109/MWC.012.2200552

[30] Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay S. Pande, and Jure Leskovec. 2018. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. 6412–6422. https://proceedings.neurips.cc/paper/2018/hash/d60678e8f2ba9c540798ebbde31177e8-Abstract.html

[31] Kexin Zhao, Xiao Tang, Limeng Dong, Ruonan Zhang, and Qinghe Du. 2025. Graph Neural Network for Multi-User MISO Secure Wireless Communications. In *2025 IEEE Wireless Communications and Networking Conference (WCNC)*. 1–6. doi:10.1109/WCNC61545.2025.10978207

[32] Lingxiao Zhao and Leman Akoglu. 2020. PairNorm: Tackling Oversmoothing in GNNs. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=rkecl1rtwB

[33] Xusheng Zhao, Jia Wu, Hao Peng, Amin Beheshti, Jessica J.M. Monaghan, David McAlpine, Heivet Hernandez-Perez, Mark Dras, Qiong Dai, Yangyang Li, Philip S. Yu, and Lifang He. 2022. Deep reinforcement learning guided graph neural networks for brain network analysis. *Neural Networks* 154 (Oct. 2022), 56–67. doi:10.1016/j.neunet.2022.06.035