Neural Processing Letters (2024) 56:24
https://doi.org/10.1007/s11063-024-11496-1

GraphSAGE++: Weighted Multi-scale GNN for Graph Representation Learning

E. Jiawei[1] · Yinglong Zhang[1] · Shangying Yang[1] · Hong Wang[1] · Xuewen Xia[1] · Xing Xu[1]

Accepted: 8 December 2023 / Published online: 9 February 2024

Abstract
Graph neural networks (GNNs) have emerged as a powerful tool in graph representation learning. However, they are increasingly challenged by over-smoothing as network depth grows, compromising their ability to capture and represent complex graph structures. Additionally, some popular GNN variants only consider local neighbor information during node updating, ignoring the global structural information and leading to inadequate learning and differentiation of graph structures. To address these challenges, we introduce a novel graph neural network framework, GraphSAGE++. Our model extracts the representation of the target node at each layer and then concatenates all layer weighted representations to obtain the final result. In addition, the strategies combining double aggregations with weighted concatenation are proposed, which significantly enhance the model's discernment and preservation of structural information. Empirical results on various datasets demonstrate that GraphSAGE++ excels in vertex classification, link prediction, and visualization tasks, surpassing existing methods in effectiveness.

Keywords Graph neural network · Over-smoothing · Double aggregation strategy · Weighted concatenation

✉ Yinglong Zhang
zhang_yinglong@126.com

E. Jiawei
1092349066@qq.com

Shangying Yang
1762568501@qq.com

Hong Wang
1745172131@qq.com

Xuewen Xia
xwxia@whu.edu.cn

Xing Xu
xxustar@qq.com

[1] School of Physics and Information Engineering, Minnan Normal University, Zhangzhou 363000, Fujian, China

1 Introduction

Graph structures are very common in real life. Graph Neural Networks (GNNs) have emerged as a formidable tool for capturing and learning the low-dimensional representations of nodes in these structures. These learned vector representations can be applied to a variety of tasks on graphs, including vertex classification [1], relation learning [2] and link prediction [3–5]. GNNs enhance the expressiveness of these vectors while preserving the structural information of graphs through their unique hierarchical architecture. Wu et al.'s comprehensive survey on graph neural networks provides a broad background for the field, detailing various neighborhood aggregation and graph pooling schemes aimed at enhancing models' capabilities in understanding and representing graph information [6].

However, due to the complexity of network structure, as the number of network layers increases [7], node representations become difficult to distinguish, and classifiers find it difficult to assign correct labels to each node. Training accuracy decreases as the number of layers increases, and the over-smooth problem [8–10] also occurs. To solve this problem, strategies for jump connections are designed. Representative algorithms include GCNII [11], ResGCN [12], and JK-Nets [13], however, these methods do not account for the varying impacts of neighbors at different hops in the network.

At the same time, these variants of GNNs cannot effectively distinguish graph structure information on simple graphs(A "simple graph" refers to networks with fewer nodes and straightforward edge relationships, while a "complex graph" is described as having a dense network with intricate connections, potentially including a large number of nodes.) under certain conditions [14], as shown in Fig. 1. Nodes v and v' are the central nodes, and their representations are generated by aggregating neighboring features. Analyzing whether different structures can be distinguished under different aggregation function settings (if different structures can be identified, their representations of the two should be different). In Fig. 1a, the two nodes$'$ results aggregated by Mean and Max are [1/2,1/2] and [1,1], respectively. Mean and Max aggregation cannot distinguish their structures, while in Fig. 1b, the Max aggregation over the neighborhood of v and v' yields the same representation [1,1], the Max aggregation cannot distinguish their structures. The above example shows that in simple graph structures, some variants of GNN [15] cannot effectively distinguish graph structure information using aggregation functions under certain conditions. In addition, these variants of GNN pay more attention to local neighboring node information and relatively less consideration to the structure information of the entire graph during node updates, which cannot capture the relationships and global structure between nodes in the graph well. In summary, the existing classical GNNs expression ability are not powerful to capture different graph structures.

GIN [14] aggregates neighboring nodes, combines the aggregation results with the node features, and finally uses fully connected network that can fit any rule to make them have monomorphic properties. It introduces a learnable parameter to adjust its own features and adds the adjusted features to the aggregated neighboring features, ensuring that the central node and neighboring nodes are distinguishable. However, this model mainly focuses on the neighbor information and have a limited understanding of global information.

In our work, we propose an improved method based on GraphSAGE, which utilizes neighbor sampling and aggregation strategies to effectively capture local and global information in the graph structure. The main contributions are as follows:

- We propose a deep learning framework GraphSAGE++ based on GraphSAGE. It extracts the feature representation of the target node at each layer and then concatenates all layer

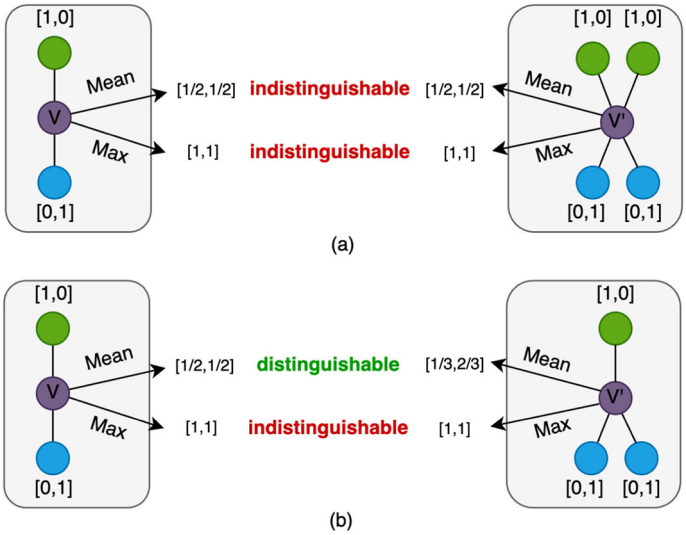

Fig. 1 Motivation for this paper. **a** Demonstrates that when graph structures with nodes labeled [1,0] and [0,1] are aggregated using Mean and Max functions, the resulting vectors are indistinguishable, hence unable to capture unique structural information. **b** Shows that the Mean aggregation results in distinguishable vectors, whereas the Max function still produces an indistinguishable vector [1,1], highlighting the limitation of these aggregation functions in differentiating between graph structures. This indicates the need for more sophisticated aggregation mechanisms to accurately reflect the underlying graph topology in feature representations

feature representation to obtain final result, where the ith layer feature information equals i-hop (i=1,2,3...) structure information between each target node and its i-hop neighbors. Considering the different impact of neighbors at each layer, our model assigns corresponding weights and proportions based on the role of the ith layer feature information in the global representation. This final representation preserves neighborhood information from 1 to K hops, thus solving the over-smoothing problems.

- Introducing strategies that actively combine double aggregations with concatenation helps enhance the distinction of structural information and improve the model's expressive capabilities.
- We conduct extensive experiments on vertex classification and prediction tasks. These experiments are conducted on several real datasets. The experimental results showed that our method achieved better performance on graph datasets and demonstrated significant advantages in tasks such as node classification and link prediction.

2 Related Work

In recent years, many methods have emerged in graph representation learning, aiming to improve the expressive ability of graph structures.

Matrix decomposition-based methods: Matrix decomposition-based methods are a type of method that utilizes matrix decomposition techniques to process graph data. These methods typically represent the graph structure in matrix form and then apply matrix decomposition algorithms to decompose and reduce the dimensionality of the graph matrix to obtain low dimensional representations or features of the graph. In SVD-GCN [16, 17], singular value

decomposition is performed on the adjacency matrix of a graph to decompose it into a product of three matrices, including an orthogonal matrix, a diagonal matrix, and a transposed orthogonal matrix. This decomposition can extract the eigenvectors and eigenvalues of a graph, thereby capturing the structure and important information. NetMF [18, 19] decomposes the adjacency matrix of a graph into a product of a node embedding matrix and a feature embedding matrix. The node embedding matrix represents the position of a node in a low dimensional space, and the feature embedding matrix represents the feature representation of a node. By minimizing reconstruction errors and regularization terms, the embedding matrix of node features is optimized, and low dimensional embedding representations with good representation ability are learned. GraRep [20] integrates the obtained information by manipulating the probability matrix and captures the global structure by iteratively updating the representation of nodes. DEGREE [21] discovers nonlinear interactions between subgraphs throughout the aggregation process by decomposing their feedforward propagation process, which helps to detect incorrect predictions and improve the model's credibility. As the size of the graph increases, the computational and storage costs also increase, and information loss may occur. In contrast, methods based on graph neural networks can better and more flexibly handle large-scale graph data with high computational efficiency and scalability.

Subgraph isomorphism-based methods: Subgraph isomorphism-based methods have a wide range of applications in graph data analysis and learning. They are able to utilize the structural information of subgraphs to extract important features of graphs and perform tasks such as graph similarity metrics, graph matching and generation, etc. GSN [22] introduces a mechanism of subgraph isomorphism counting, which compares and matches local subgraphs in a graph with a set of predefined subgraph patterns. By counting the number of successfully matched subgraphs, the frequency of occurrence or distribution of different subgraph patterns in the graph can be obtained to capture the local structure and connectivity patterns of the graph more comprehensively, and to enhance the ability of graph neural networks to differentiate between different subgraphs. Algorithms such as WL [23], GraphSNN [24] determine subgraph isomorphism of nodes by comparing their labels with the label of their neighboring nodes. Two nodes are considered isomorphic if they have the same sequence of labels. This comparison based on subgraph isomorphism allows these algorithms to identify graphs with similar local structures and is used for graph isomorphism determination and graph classification tasks. However, due to the inability of WL [23] and GraphSNN [24] to deal with the order sensitivity of node labels and global structure information, they may not be able to accurately represent some graphs with complex global structures.

Methods based on graph neural networks: Methods such as GCN [25, 26], SGCN [27], GIN [14], and Causal GraphSAGE [28] aggregate information over neighboring nodes through convolution of multiple layers. These methods can capture the local structure of the nodes but have limitations in dealing with the global structure. There are some methods that focus on improving the neighbor sampling. GraphSAGE [29] is one of the classical methods that aggregate information by randomly sampling neighbor nodes, which can significantly and efficiently reduce the computational cost, enabling node embedding learning even on large graphs. Some methods introduce importance weights or attention mechanisms for neighboring nodes, such as GAT [30, 31], which use the degrees of neighboring nodes or the similarity between nodes to assign weights and aggregate node features at each layer by attention weights. These methods can more accurately select neighbor nodes for information aggregation and improve the quality of node representation. However, current approaches tend to treat multi-scale representation learning and neighbor sampling as independent steps and are not well integrated.

Methods based on random walk: MIRW [5] considers the mutual influence between nodes to better capture the complexity of the network. This method contrasts with traditional random walk strategies, which consider all nodes and links equally important and therefore cannot fully reflect the general structure of the graph. In addition, CSADW [4] enhances the link prediction ability in social networks by combining a new transition matrix with structural and attribute similarity. This combination improves traditional techniques by ensuring that random walks are biased towards nodes with similar structures, thereby capturing structural and unstructured similarities. They perform well in capturing local node interactions and structural attribute similarities, but they have limitations in handling global graph structures [32].

To effectively address these issues, we propose the GraphSAGE++ algorithm, which considers multi-scale graph structure information by a weighted multi-dimensional mechanism. This involves assigning different weights to different hop neighboring nodes based on their importance to redefine their dimensionality, thereby better capturing local and global graph structure information. Our method can more comprehensively model the structure of graphs and improve the performance of graph representation learning.

3 GraphSAGE++

In this section, we will introduce our framework. We first introduce the definition of the relevant problem, then introduce its overall framework, and finally discuss how to integrate the global structural information of the graph more effectively.

3.1 Problem Definition

This section provides some basic definitions.

Definition 1 Given a graph $G = (V, E, X)$, where $V = \{v_1, v_2, ..., v_n\}$, It represents a set of n nodes in a graph. $N = |V|$ is the number of nodes in the graph, and $|E|$ is the number of edges. $E = \{e_{i,j}\}_{i,j=1}^{n}$, Represents the set of all edges in a graph, When there is an edge between v_i and v_j, $e_{i,j} = 1$, otherwise $e_{i,j} = 0$. $X \in R^{N \times D}$ represents the feature matrix for all notdes with $x_v \in R^D$ representing the feature vector of a node v.

Definition 2 The feature vector of v at the i-th layer denoted by h_v^i and $h_v^0 = X_v$. $\mathcal{N}(v) = \{u \in V : (u, v \in E)\}$ represents the neighborhood set of vertex v in the graph.

3.2 GraphSAGE

The advantage of GraphSAGE [29] lies in its ability to effectively capture local structural information of nodes and its scalability to process large-scale graph data. The main idea is to obtain node representations by aggregating neighbor information through neighbor sampling. GraphSAGE is a K-layer network. For a node v, three operations of neighbor sampling, aggregating neighbor information, and node representation updating are performed at each layer. Specifically, at the ith layer the model firstly collects fixed neighbors $\mathcal{N}(v)$ of the node v. Then it aggregates the neighbor information, as shown in Eq. (1).

$$h_{\mathcal{N}(v)}^i \leftarrow AGGEREGATE\left(h_u^{i-1}, u \in \mathcal{N}(v)\right) \tag{1}$$

Finally, it updates the vector representation of node v, as shown in Eq. (2), where Sigmad function σ is a nonlinear function: $\frac{1}{1+e^{(-x)}}$, it is capable of extracting nonlinear relationships.

$$h_v^i \leftarrow \sigma\left(W_i \cdot CONCAT\left(h_v^{i-1}, h_{\mathcal{N}(v)}^i\right)\right). \tag{2}$$

W_i is the training parameter, which is obtained by training with the following objective function:

$$L(p,q) = -\sum(p(v)logq(v) + (1-p(v))log(1-q(v))) \tag{3}$$

This objective function mainly portrays the distance between the actual output (probability) and the desired output (probability). The probability distribution p is the desired output, and the probability distribution q is the actual output. The smaller the value, the closer the two probability distributions are.

3.3 Framework of GraphSAGE++

GraphSAGE suffers from the over-smoothing problems. And it mainly captures the local structural information of the nodes, while the global structural information of the graph is not effectively captured. To solve the problems, we have improved GraphSAGE. Specifically, for a node v, its direct neighbor nodes have the greatest influence on it. While the influence of i-hop neighbor nodes gradually decreases as the hop increases. The overall framework is shown in Fig. 2.

As shown in Fig. 2, At the ith layer the node v representation h_v^i is reserved after the three operations of neighbor sampling, aggregating neighbor information, and node representation updating. At the Kth layer, the representations from all layer are concatenated to obtain the final global representation of $[h_v^1, h_v^2, ..., h_v^K]$. Such a representation preserves the neighbor information from 1 to K hops, thus solving the over-smoothing problem.

Neighbors with different hops affect nodes differently, so the node vectors of each layer are composed of global representations with corresponding weights and proportions. The specific idea is that the dimension of the node representation decreases as the hops increases. Let d be the dimension of the final global representation(the value of d can be set by the user, and the value is 128 in the paper). N_i is the dimension of node representation from the ith layer, K is the maximum number of layers of the model(the value in the paper is 8). The ratio of the ith layer to d is p_i, and the specific formula is shown as follows. We firstly compute the value of the given ratio p_i by the Eqs. (4) and (5), and then use Eq. (6) to calculate to get the dimension N_i.

$$\vartheta = (K - i + 1)^{\frac{1}{2}} \tag{4}$$

$$p_i = \frac{\vartheta}{\sum_0^K \vartheta} \tag{5}$$

$$N_i = int(d * p_i) \tag{6}$$

Algorithm 1 shows the process of GraphSAGE++ forward propagation. Firstly, we randomly sample the neighbor nodes of v using the neighbor sampling function. Then, by means of the aggregation function, we obtain the representation h_u^i of the neighbor nodes of v. Next, we perform a concatenating operation and obtain the representation of v by applying the nonlinear change σ to the result of the concatenation. In line 8, we perform a normalization operation on the representation of the vertex. Eventually, after concatenating all nodes

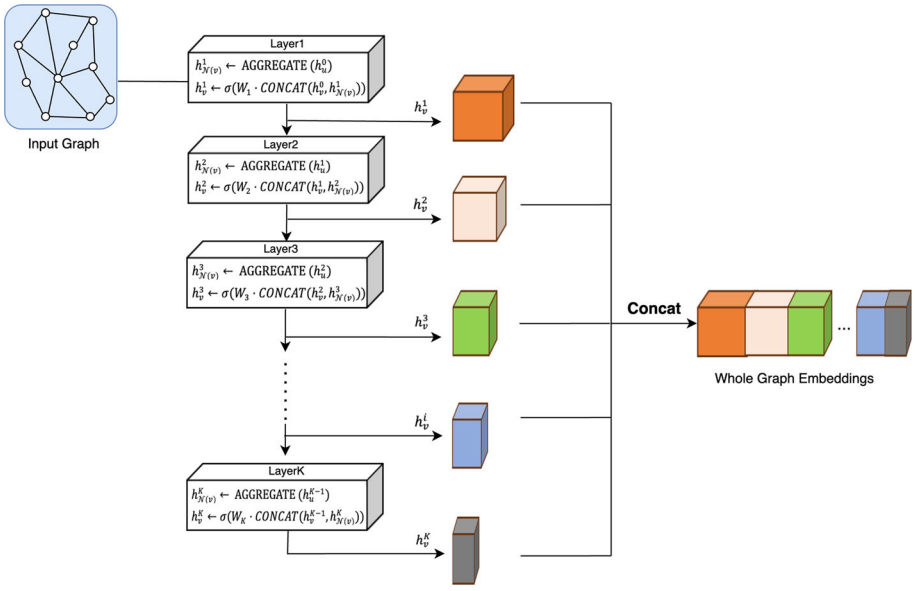

Fig. 2 GraphSAGE++ architecture

Algorithm 1 GraphSAGE++ forward_propagation algorithm

Require: $G(V, E)$; input features$\{X_v, \forall v \in V\}$; depth K; aggregatorfunctions AGG-REGATE; Neighborhood Sampling Function $\mathcal{N}$; non-linearity σ; Weight matrices $W = W_1, ..., W_K, \forall i \in \{1, ..., K\}$

Ensure: Representation vectors after concatenating operation H_v

1: Function forward_propagation(X_v, W)
2: $h_v^0 \leftarrow X_v, \forall v \in V$;
3: For $i = 1$ to K do
4: For $v \in V$ do
5: Calculate N_i assigned to the current layer according to equation (4)
6: $h^i_{\mathcal{N}(v)} \leftarrow \text{AGGEREGATE}(h_u^{i-1}, u \in \mathcal{N}(v))$
7: $h_v^i \leftarrow \sigma(W_i \cdot CONCAT(h_v^{i-1}, h^i_{\mathcal{N}(v)}))$
8: $h_v^i \leftarrow h_v^i / ||h_v^i||_2$
9: $H_v \leftarrow CONCAT(h_v^1, h_v^1, ..., h_v^K)$
10: Return H_v

v, we obtain the final vector representation H_v: $[h_v^1, h_v^2, ..., h_v^K]$. In line 5, we assign a fixed number of dimensions to the representation at each layer by using Eq. (6) to ensure that the important representation information is effectively preserved.

Algorithm 2 shows the GraphSAGE++ back-propagation process. The vector representation h_v of vertex v at each hop is obtained by forward-propagation process, see line 4. Lines 3 to 5 are the process of calculating the loss value, firstly, softmax operation is performed on the vector representation of each node to convert the node vector representation to a probability distribution p, and then the label y of the node is converted to a one-hot encoding, which is computed by the loss function to obtain the loss value. Lines 6 to 7 are the process of calculating the gradient and updating the parameters.

Algorithm 3 is the GraphSAGE++ algorithm. Firstly, the weight matrix W is initialized with Xavier.Then the global vector representation H_v of v is obtained by calling Algorithm 1

Algorithm 2 GraphSAGE++ backward_propagation algorithm

Require: $G(V, E)$; input features$\{X_v, \forall v \in V\}$; depth K; learning rate lr; gradient calculation function $gradients$; encoding function one_hot; Loss function value $loss$; Node label y
Ensure: Weight matrices W
1: Function backward_propagation(X_v, y, W, lr)
2: $h_v \leftarrow$ forward_propagation(X_v, W)
3: $p \leftarrow softmax(h_v)$
4: $q \leftarrow one\_hot(y)$
5: Calculate the $loss$ value using equation (1), $loss \leftarrow L(p, q)$
6: gradient $\leftarrow gradients(loss, W)$
7: $W \leftarrow lr *$ gradient
8: Return W

forward_propagation, see lines 4 to 5 of the algorithm. Finally, the parameter update is accomplished by calling Algorithm 2 back_propagation and the loop is iterated until the whole model converges.

Algorithm 3 GraphSAGE++ Training parameters framework

Require: $G(V, E)$; input features$\{X_v, \forall v \in V\}$; depth K; learning rate lr; Node label y
Ensure: Weight matrices W
1: Input $G(V, E)$
2: Initialize the weight parameter W with Xavier
3: Repeat
4: For $v \in V$ do
5: $H_v \leftarrow$ forward_propagation(X_v, W)
6: $W \leftarrow$ backward_propagation (H_v, y, W, lr)
7: Until 100 iterations have been completed

3.4 Model Optimization

In this section, we make some improvements. As shown in Fig. 1, the Mean aggregation and Max aggregation functions cannot effectively capture graph structure information. Mean aggregation takes the average of the neighboring features of a node, which means that each neighbor's feature has the same weight on the final aggregation result. This aggregation method can lead to information loss because it ignores the importance differences between different neighboring nodes. In some cases, mean aggregation may not be able to distinguish different neighboring nodes because their features are mixed together. Max aggregation takes the maximum value of the neighboring features of a node as the aggregation result. This aggregation method preserves the most important features in each neighboring node, but may also lose other information. Max aggregation usually focuses more on certain important neighbor nodes, but does not pay attention to other neighbors. In order to better distinguish the structural information and improve the expression ability, we have designed different methods: GraphSage++ with double aggregate (GraphSage++DA), GraphSage++ with double aggregate & concat (GraphSage++DAC) and GraphSage++ with double aggregate & mixed concat (GraphSage++DAMC).

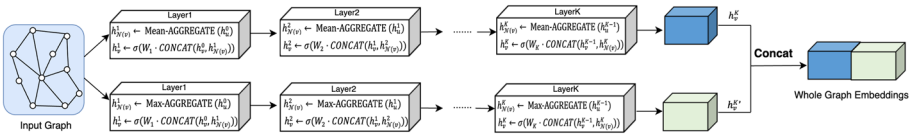

Fig. 3 GraphSAGE++DA architecture

3.4.1 GraphGAGE++DA

The GraphSAGE++DA is shown in Fig. 3. The main body consists of two GraphSAGE models aggregated using Mean and Max, respectively. At the ith layer, they share the parameter W_i. The GraphSAGE model with the aggregation function of Mean obtains the feature vector representation of the target node at the ith layer, h_v^K, and the GraphSAGE model with the aggregation function of Max get the feature vector representation $h_v^{K\prime}$ of the target node at the Kth layer. Concatenateing these two feature vectors to obtain the final vector representation of the target node $[h_v^K, h_v^{K\prime}]$. GraphSAGE++DA does not effectively capture both local and global graph structure information.

3.4.2 GraphGAGE++DAC

GraphSAGE++DA suffers from the same oversmoothing problem. For this reason Graphsage++DAC was designed. As shown in Fig. 4, the vector representation of each layer is saved, and the dimension of each layer is based on Eq. (5). h_v^i indicates the representation using the Mean aggregation function, and $h_v^{i\prime}$ is the aggregation function of the ith layer is the representation using the Max aggregation function. And the $2K$ vectors are concatenated to obtain the final vector representation of the target node $[h_v^1, h_v^2, ..., h_v^K, h_v^{1\prime}, h_v^{2\prime}, ..., h_v^{K\prime}]$, where the first K vectors are representations of K layers whose aggregation function is Max.

3.4.3 GraphGAGE++DAMC

The only difference between GraphSAGE++DAC and GraphSAGE++DAMC is that the final vector representation of the concat is different, where the concatenated final vector is represented as $[h_v^1, h_v^{1\prime}, h_v^2, h_v^{2\prime}, ..., h_v^K, h_v^{K\prime}]$.

4 Experiments

For the implementation of our models, we utilized the PyTorch Geometric (PyG) framework, which is specifically designed for deep learning on irregularly structured input data such as graphs. PyG is well-regarded for its efficient handling of large-scale graph data and its extensive library of optimized graph neural network layers. All experiments were conducted on a computing setup powered by an Apple M1 Pro(14 Core)@3.20Ghz processor, which provided the necessary computational capacity for the intensive training and evaluation phases of our graph neural network models. Our code and implementation details can be found on: https://github.com/ejwww/SAGE-plus.

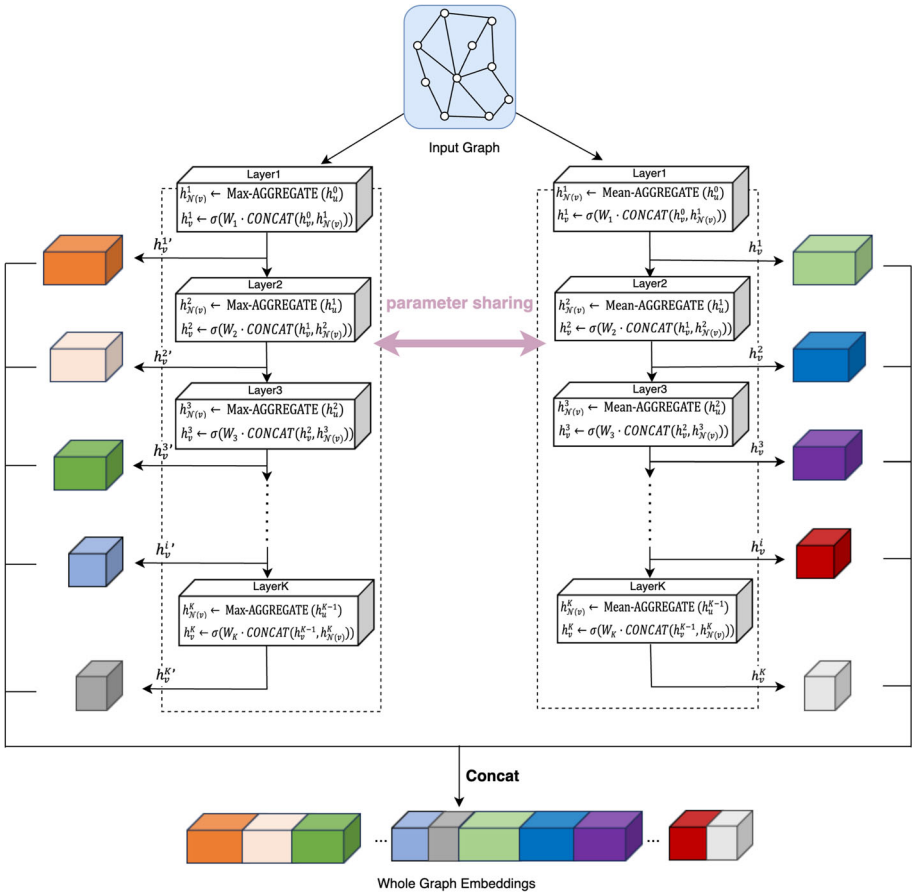

Fig. 4 GraphSAGE++DAC architecture

4.1 Experimental Settings

To evaluate the performance of different GNN modes, the evaluation metrics listed in Table 1 were used, where TP, TN, FP, and FN represent the number of true positive, true negative, false positive, and false negative, respectively.

Parameter settings: For all algorithms, the size of the training set varies from 0.1 to 0.9 and the rest are test sets. The value of the dimension d set in the paper is 128. The learning rate lr of the model is set to a value of 0.001. The batch size is set to 64. he value of the maximum number of hops K of the model is set to 8. To optimize the model parameters, we employed the backpropagation algorithm combined with the Adam optimizer. The Adam optimizer is favored in the deep learning community for its adaptive learning rate capabilities and momentum incorporation, which facilitate faster convergence and robust generalization when training complex models. For regularization, we integrated L_2 regularization into our training process to mitigate the risk of overfitting. This approach penalizes the square magnitude of parameters, effectively constraining the complexity of the model and promoting simpler, more generalizable solutions that perform better on unseen data. In order to increase the

Table 1 The evaluation indicators used in this study

Metric	Vertices
Recall(Detection rate)	$\frac{TP}{TP+FN}$
Precision	$\frac{TP}{TP+FP}$
F1-Score	$2 \times \frac{Reecall \times Precision}{Recall+Precision}$
Accuracy	$\frac{TP+TN}{TP+FP+TN+FN}$

generalization ability of the model and reduce the risk of overfitting, we introduced the dropout parameter p. The value of p is set to 0.5, because a smaller value of p means that more elements are discarded, the capacity of the model decreases, thereby reducing the risk of overfitting; A larger value of p means that fewer neurons are discarded and the model's capacity increases, but it may also increase the risk of overfitting. The value of the maximum number of hops K of the model is set to 8. The neighborhood sample sizes $S_1 = 20$ and $S_2 = 15$ when the value of K is 2. When K is extended to higher values, we define that $S_3 = 10, S_4 = 5, S_5 = 4, S_6 = 3$, and so on, decreasing layer by layer. We use more neighbor sampling for layers near the input layer, while layers near the output layer use less sampling. This method can reduce computational complexity and avoid over smoothing, preventing excessive use of computing and storage resources, especially when there are many network layers.

4.1.1 Datasets

We test with seven real datasets commonly used in vertex embeddings. The details of these datasets are shown in Table 2, including social networks, protein datasets and citation networks. $Cora$ [33] is the standard citation network benchmark data set. Here, the vertex represents the paper, the edge represents the reference of one paper to another, the vertex feature is the word bag representation of the paper, and the vertex label is the academic topic of the paper. The dataset, $Pubmed$ [33] consists of 19,717 scientific publications on diabetes from the Pubmed database, in which it categorizes the entire data into three categories. $Amazon$ [33], the graph structure of the dataset represents the relationship between users and goods, where users and goods are the nodes of the graph and the purchases between users and goods are the edges of the graph. PPI [34] is a subgraph generated from human proteins with different labels for different proteins; there are fifty labels in this dataset. $Citeseer$ [33] is a citation network where nodes in the dataset represent scholarly literature, directed edges represent citation relationships between the literature, and the nodes contain some characteristic information about the literature. $ENZYMES$ [35] is a collection of graph data constructed on the basis of the protein structure of biomolecules, with a total of 600 graphs corresponding to 600 samples (protein molecules) with six structures. $KarateClub$ [36] describes the social relationships of members of a karate club with 34 members as nodes, and adds an edge between nodes if two members remain socially connected outside the club.

4.1.2 Baseline algorithm

We use the following representative algorithms in comparative experiments.

GCN [25] is one of the earliest proposed graph convolutional network model. It uses adjacency matrices and node feature matrices for node feature updating and propagation.

Table 2 Network datasets statistics

Dateset	Vertices	Edges	Labels
Cora	2707	5429	7
Citeseer	3327	4732	6
Pubmed	19, 717	44, 327	3
Amazon	13, 381	245, 778	10
PPI	3980	38, 397	50
ENZYMES	19, 580	74, 565	6
KarateClub	34	78	2

At each layer the feature representation of the current node is updated by aggregating the features of neighboring nodes.

SGCN [27] is a simplified version of the graph convolution neural network model that simplifies the graph convolution operation to a single linear operation by removing the nonlinear transformation and polynomial fitting, reducing model complexity and computational overhead, and improving model interpretability and efficiency.

GraphSAGE [29] is the graph convolutional neural network algorithm, which aggregates the neighbor information of the vertices through an aggregation function and updates it through training, and as the number of iterations increases, the vertices are able to aggregate to higher-order neighbor information.

JK-Net [13] is a framework for graph neural networks that aims to improve the performance of GNNs on graph data tasks such as node classification, graph classification, etc. The core idea of *JK-Net* is to integrate information from different levels of graph convolutional layers through jumping connections (jumping connections) to improve the model's ability to learn the representation of the graph data.

GIN [14]:In graph isomorphism detection, the goal is to determine whether two given graphs are isomorphic, i.e., their nodes can be mapped one-to-one and keep the edges connected in the same relationship, GIN applies this idea to graph neural networks. Its learns the graph representation by combining the features of the nodes with the features of the neighboring nodes, which are then nonlinearly varied by a multilayer perceptron.

GCNII [11] introduces a cross layer information transmission and importance coefficient mechanism, which adaptively adjusts the importance of different layers, taking into account the information of multiple neighboring orders at each layer, and weighting them with different importance coefficients.

GraphSNN [24] proposes a new message-passing framework, GMP, which can inject local structures into aggregation schemes based on overlapping subgraphs Performance is improved on different graph structures.

SVD-GCN [16] is a graph convolutional neural network model based on singular value decomposition, which obtains low-order features and high-order features by performing singular value decomposition of the adjacency matrix, and performs downscaling and representation learning of graph data by preserving low-order features and truncating high-order features.

DGCN [37] is a graph neural network model that further optimizes the topology of the graph by decoupling the processes of feature learning and graph structure learning, where the feature learning phase exploits the similarity of node features for feature propagation and representation learning, and the graph structure learning phase focuses on the connectivity between nodes to further optimize the topology of the graph.

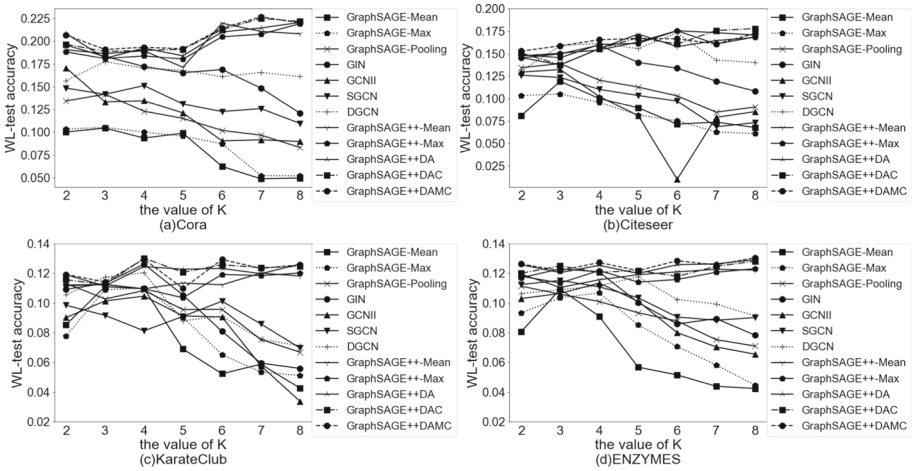

Fig. 5 Accuracy results of different algorithms for WL isomorphism detection with increasing K value on different datasets, where our models can achieve higher WL-test accuracy

4.2 Weisfeiler-Lehman Test

The WL test is an iterative graph local feature encoding algorithm for detecting graph isomorphism. We make some adjustments to the algorithm. For each node in the graph, we initialize a label for it, and then iteratively use the GraphSAGE++ to aggregate the features of neighboring nodes. At the same time, the aggregated features are mapped to new labels, and the labels of nodes in the two subgraphs are compared. If the labels of the two subgraphs are the same, they are considered isomorphic. The aggregation functions used for the WL isomorphism test in this paper are the same as those used in the fixed-point classification and link prediction tasks, which are Mean aggregation function and Max aggregation function. This section validates the effectiveness of the algorithm on the WL isomorphism test task based on experimental results and compares it with other baseline algorithms. The average of ten experiments is taken as the result here.

Figure 5 shows the comparison of the results of the WL isomorphism detection accuracy of different algorithms on different datasets. As can be seen from the figure, at K=2 hops, the method proposed in this paper has been more competitive, and GIN on the Citeseer dataset is comparable to the GraphSAGE++DAC and GraphSAGE++DAMC performances. Compared to GraphSAGE, our schemes have obtained better performances on both the Cora dataset and the Citeseer dataset, GraphSAGE++DAMC improves the isomorphism detection accuracy by seventy percent compared to GraphSAGE. As the number of hops increases, the ability of this paper's scheme to perform WL isomorphism detection becomes more competitive. When K is greater than 4, the isomorphic detection accuracy of GIN, GCINII, GraphSAGE, and SGCN all show a significant downward trend. With the increase of hop, GraphSAGE++ achieved good performance and tended to stabilize, among which the comprehensive performance of GraphSAGE++DAMC is more prominent.

Table 3 f1_micro values obtained by completing the vertex classification task on the Cora dataset

Algorithm	10%	30%	50%	70%	90%
GCN	0.2256	0.2171	0.2448	0.2031	0.2098
SGCN	0.2238	0.2508	0.2201	0.2065	0.2412
GIN	0.2401	0.2342	0.2309	0.2445	0.2426
GCNII	0.2426	0.2238	0.2227	0.2534	0.2489
DGCN	0.2565	0.2442	0.2616	0.2308	0.2412
SVD-GCN	0.2688	0.2735	0.2628	0.2898	0.2911
GraphSNN	0.2609	0.2870	0.2617	0.2902	0.2734
GraphSAGE-mean	0.2511	0.2393	0.2526	0.2891	0.2518
GraphSAGE-max	0.2692	0.2719	0.2190	0.2194	0.2685
GraphSAGE-pooling	0.2712	0.2738	0.2556	0.2889	0.2733
SAGE-based JK-net	0.2898	0.2974	0.2886	0.3001	0.2905
GraphSAGE++mean	0.2834	0.3012	0.2812	0.2983	0.3009
GraphSAGE++max	0.2821	0.2715	0.2784	0.2946	0.2912
GraphSAGE++DA	0.2868	0.2830	0.2903	0.3013	0.2861
GraphSAGE++DAC	0.3009	**0.3124**	0.2997	**0.3066**	0.3018
GraphSAGE++DAMC	**0.3073**	0.2992	**0.3057**	0.3052	**0.3082**

4.3 Experimental Result

In this section, we verify the effectiveness of our methods in link prediction, classification, and visualization. And we compare them with other baseline algorithms. It is noteworthy that throughout the tables in this section (Tables 3–12), the best achieved results are highlighted in bold.

4.3.1 Classification Task

The purpose of vertex classification tasks is to predict the labels of each vertex.

Tables 3, 4, 5, 6, 7 and Fig. 6 demonstrate the performance of our methods and several other algorithms in performing classification tasks on 5 real datasets. During the evaluation process, we ran the program ten times and then derived their averages to ensure fair competition. The experimental results show that GraphSAGE++ achieves good results in all datasets. On the Amazon dataset, GraphSAGE based JK-Net performed better (when the training set is 70%), and compared to GraphSAGE, GraphSAGE+DAC improved performance on classification tasks by about 13% (when the training set is 90%). On the PPI dataset, GraphSAGE+DAMC improved by about 7% compared to GraphSAGE (when the training set is 50%). On the Citeseer dataset, GraphSAGE++DAMC showed a performance improvement of about 6% compared to GraphSAGE++DA (when the training set is 70%), while GraphSAGE++DAC showed a performance improvement compared to GraphSAGE++DA. Overall, in terms of classification tasks, GraphSAGE++DAC and GraphSAGE++DAMC have achieved better performance compared to other algorithms.

Figure 7 demonstrates the time consumed by different algorithms to complete the node classification task on the PPI dataset, which still has a relatively large overhead in time compared to the traditional algorithms due to the use of concatenating operations. Although

Table 4 f1_micro values obtained by completing the vertex classification task on the Pumbed dataset

Algorithm	10%	30%	50%	70%	90%
GCN	0.1776	0.1790	0.2042	0.1903	0.1844
SGCN	0.1752	0.2157	0.2023	0.2212	0.2076
GIN	0.1873	0.2356	0.2200	0.2107	0.2098
GCNII	0.1894	0.2457	0.2169	0.2214	0.2180
DGCN	0.2176	0.2412	0.2108	0.2519	0.2307
SVD-GCN	0.2423	0.2770	0.2566	0.2647	0.2779
GraphSNN	0.2568	0.2820	0.2874	0.2626	0.2612
GraphSAGE-mean	0.2072	0.1983	0.1702	0.2227	0.2175
GraphSAGE-max	0.2253	0.2197	0.2190	0.2470	0.2308
GraphSAGE-pooling	0.2442	0.2283	0.2367	0.2411	0.2389
SAGE-based JK-net	0.2594	0.2912	0.2513	0.2530	0.2610
GraphSAGE++mean	0.2677	0.2426	0.2412	0.2460	0.2552
GraphSAGE++max	0.3087	0.2879	0.2594	0.2544	0.2581
GraphSAGE++DA	0.2810	0.2969	0.2598	0.2459	0.2645
GraphSAGE++DAC	0.3149	**0.3286**	**0.2644**	0.2600	0.2759
GraphSAGE++DAMC	**0.3179**	0.2858	0.2574	**0.2667**	**0.2818**

Table 5 f1_micro values obtained by completing the vertex classification task on the PPI dataset

Algorithm	10%	30%	50%	70%	90%
GCN	0.4385	0.4351	0.4321	0.4311	0.4300
SGCN	0.4395	0.4378	0.4370	0.4410	0.4322
GIN	0.4405	0.4512	0.4434	0.4497	0.4366
GCNII	0.4456	0.4530	0.4477	0.4510	0.4482
DGCN	0.4497	0.4578	0.4409	0.4598	0.4490
SVD-GCN	0.4503	0.4601	0.4506	0.4631	0.4566
GraphSNN	0.4621	0. 4612	0.4695	0.4586	0.4577
GraphSAGE-mean	0.4443	0.4430	0.4326	0.4439	0.4422
GraphSAGE-max	0.4512	0.4498	0.4560	0.4487	0.4563
GraphSAGE-pooling	0.4549	0.4601	0.4599	0.4613	0.4568
SAGE-based JK-net	0.4599	0.4723	0.4660	0.4715	0.4600
GraphSAGE++mean	0.4549	0.4621	0.4733	0.4638	0.4575
GraphSAGE++max	0.4487	0.4696	0.4599	0.4637	0.4609
GraphSAGE++DA	0.4462	0.4687	0.4658	0.4650	0.4521
GraphSAGE++DAC	0.4603	0.4739	0.4760	**0.4825**	**0.4713**
GraphSAGE++DAMC	**0.4628**	**0.4762**	**0.4770**	0.4714	0.4696

Table 6 f1_micro values obtained by completing the vertex classification task on the Citeseer dataset

Algorithm	10%	30%	50%	70%	90%
GCN	0.2451	0.2171	0.2448	0.2230	0.2118
SGCN	0.2489	0.2780	0.2431	0.2274	0.2412
GIN	0.2491	0.2533	0.2774	0.2552	0.2603
GCNII	0.2523	0.2450	0.2787	0.2732	0.2689
DGCN	0.2760	0.2728	0.2931	0.2508	0.2612
SVD-GCN	0.3394	0.3556	0.3572	0.3609	0.3548
GraphSNN	0.3402	0. 3389	0.3516	0.3128	0.3050
GraphSAGE-mean	0.3312	0.3258	0.2919	0.3294	0.2714
GraphSAGE-max	0.3369	0.3354	0.3216	0.3547	0.2780
GraphSAGE-pooling	0.3321	0.3069	0.3266	0.3550	0.2994
SAGE-based JK-net	0.3155	0.3252	0.3369	0.3412	0.3349
GraphSAGE++mean	0.3078	0.3186	0.3330	0.3598	0.3538
GraphSAGE++max	0.3303	0.3033	0.3225	0.3381	0.3538
GraphSAGE++DA	0.3126	0.2928	0.3478	0.3390	0.3472
GraphSAGE++DAC	**0.3618**	0.3384	**0.3721**	0.3430	**0.3565**
GraphSAGE++DAMC	0.3232	**0.3628**	0.3470	**0.3612**	0.3479

Table 7 f1_micro values obtained by completing the vertex classification task on the Amazon dataset

Algorithm	10%	30%	50%	70%	90%
GCN	0.2730	0.2809	0.2980	0.3146	0.2853
SGCN	0.2998	0.3018	0.3206	0.3355	0.3110
GIN	0.3115	0.3309	0.3152	0.3286	0.3191
GCNII	0.3125	0.3509	0.2993	0.3390	0.3276
DGCN	0.3518	0.3690	0.3714	0.3552	0.3611
SVD-GCN	0.3812	0.3797	0.3868	0.3985	0.3770
GraphSNN	0.3810	0.3728	0.3956	0.3954	0.3837
GraphSAGE-mean	0.3730	0.3743	0.3713	0.3640	0.3605
GraphSAGE-max	0.3773	0.3759	0.3679	0.3674	0.3642
GraphSAGE-pooling	0.3840	0.3912	0.3887	0.3923	0.3748
SAGE-based JK-net	0.4048	0.3995	0.4023	**0.4101**	0.3978
GraphSAGE++mean	0.3880	0.3912	0.3821	0.3948	0.3719
GraphSAGE++max	0.3791	0.3854	0.3882	0.3926	0.3879
GraphSAGE++DA	0.3963	0.4010	0.3916	0.3809	0.3890
GraphSAGE++DAC	0.4078	0.3996	**0.4126**	0.4080	**0.4104**
GraphSAGE++DAMC	**0.4135**	**0.4082**	0.3906	0.4057	0.3827

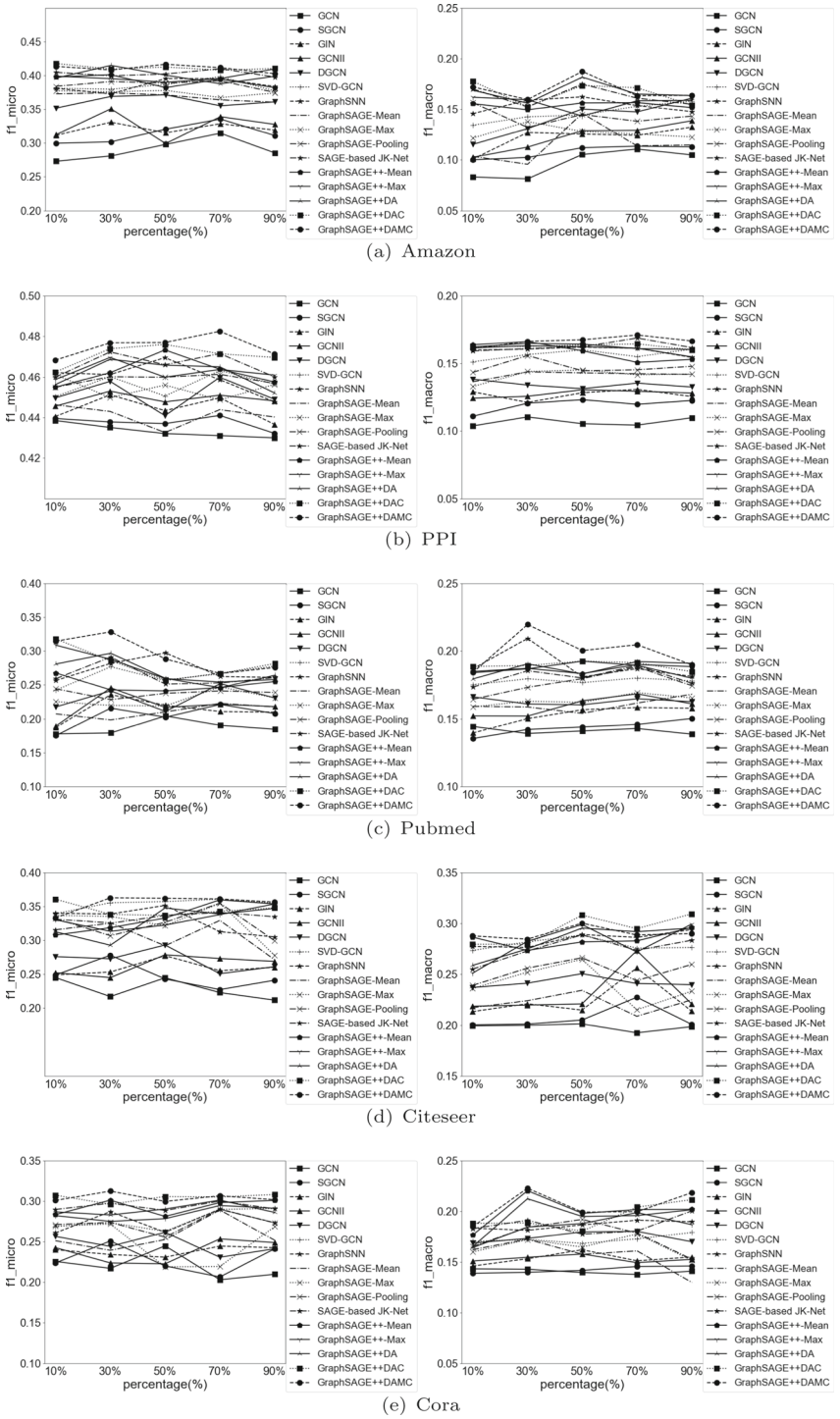

Fig. 6 Values of f1_micro and f1_macro obtained by different algorithms for classification tasks on different datasets

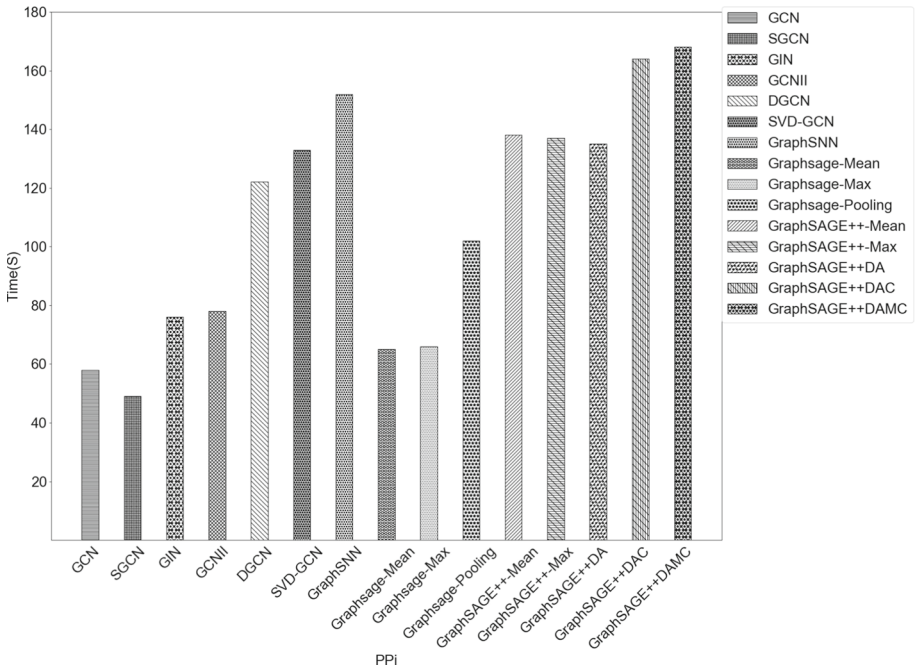

Fig. 7 Comparison of algorithms' time-consuming on the link prediction tasks

the algorithm proposed in this paper cannot outperform SVD-GCN, GraphSNN and DGCN etc. in terms of time efficiency, it achieves better performance than them in most of the cases in link prediction and classification tasks.

4.3.2 Link Prediction

The purpose of link prediction is to predict possible connections or edges based on known information about the network structure.

Tables 8, 9, 10 display the results of the link prediction task on the Amazon,Pubmed and Cora datasets. In the Cora dataset, our scheme achieved better performance as the training set increased. Compared to GraphSAGE, GraphSAGE++DAMC increased by about 7%, compared to GraphSAGE++Mean and GraphSAGE++Max increased by about 2%, and compared to GraphSAGE++DA increased by about 3% (when the training set is 50%). In the Pubmed dataset, GraphSAGE++DAC increased by about 20% compared to GraphSAGE (when the training set was 70%), and GraphSAGE++DAMC increased by 17% compared to GraphSAGE (when the training set was 90%). It is worth mentioning that overall, GraphSAGE+DAMC performs significantly better than other algorithms.

Figure 8 shows the effect of the different values of K in the experiment on the acc score of each algorithm when performing the classification task on Cora, Citeseer, Amazon, and Pubmed datasets, and the results show that our algorithms are not affected by the value of K and the overall shows an upward trend. Among them, as the depth increases, the DGCN algorithm realizes the function of controlling its expression power to strengthen or weaken gradually through the decoupling operation, so the effect of DGCN to become less effective by the increase of the value of K is not significant. The baseline algorithms show a significant

Table 8 f1_micro values obtained by completing the link prediction task on the Amazon dataset

Algorithm	10%	30%	50%	70%	90%
GCN	0.5005	0.5000	0.5914	0.5746	0.5765
SGCN	0.5337	0.5461	0.5875	0.5896	0.5502
GIN	0.5664	0.6109	0.5897	0.5959	0.5601
GCNII	0.5753	0.6086	0.5847	0.5916	0.5773
DGCN	0.6068	0.6185	0.5934	0.6003	0.5732
SVD-GCN	0.6157	0.6264	0.6733	0.6770	0.6682
GraphSNN	0.6198	0.6110	0.6790	0.6548	0.6226
GraphSAGE-mean	0.6130	0.6087	0.6495	0.6356	0.5740
GraphSAGE-max	0.6146	0.5995	0.6612	0.6420	0.6060
GraphSAGE-pooling	0.6229	0.6501	0.6668	0.6808	0.6462
SAGE-based JK-net	0.6230	0.6547	0.6632	0.6748	0.6523
GraphSAGE++mean	0.6289	0.6301	0.6758	0.6848	0.6423
GraphSAGE++max	0.6174	0.6089	0.6702	0.6790	0.6194
GraphSAGE++DA	0.6190	0.6200	0.6693	0.6812	0.6610
GraphSAGE++DAC	0.6273	0.6662	0.6770	**0.6931**	0.6596
GraphSAGE++DAMC	**0.6382**	**0.6730**	**0.6842**	0.6057	**0.6648**

Table 9 f1_micro values obtained by completing the link prediction task on the Pubmed dataset

Algorithm	10%	30%	50%	70%	90%
GCN	0.4706	0.4978	0.5112	0.4804	0.4536
SGCN	0.4998	0.5140	0.5205	0.4956	0.4706
GIN	0.5005	0.5230	0.5355	0.4877	0.4624
GCNII	0.5167	0.5334	0.5189	0.4785	0.4660
DGCN	0.5343	0.5580	0.5628	0.5079	0.5121
SVD-GCN	0.5765	0.5886	0.5902	0.5456	0.5556
GraphSNN	0.5694	0.5845	0.5910	0.5403	0.5314
GraphSAGE-mean	0.5455	0.5389	0.5199	0.4877	0.4748
GraphSAGE-max	0.5646	0.5535	0.5307	0.4801	0.4906
GraphSAGE-pooling	0.5621	0.5846	0.5897	0.5445	0.5500
SAGE-based JK-net	0.5732	0.5925	0.5876	0.5531	0.5630
GraphSAGE++mean	0.5789	0.5612	0.5796	0.5106	0.5030
GraphSAGE++max	0.5830	0.5735	0.5848	0.5263	0.5124
GraphSAGE++DA	0.5664	0.5995	0.5775	0.5117	0.5050
GraphSAGE++DAC	0.5779	0.6002	0.6163	**0.5884**	0.5880
GraphSAGE++DAMC	**0.5990**	**0.6108**	**0.6220**	0.5772	**0.5935**

Table 10 Auroc values obtained by completing the link prediction task on the Cora dataset

Algorithm	10%	30%	50%	70%	90%
GCN	0.7894	0.8267	0.8584	0.8138	0.8189
SGCN	0.8552	0.8252	0.8438	0.8210	0.8318
GIN	0.8509	0.8557	0.8612	0.8319	0.8348
GCNII	0.8546	0.8703	0.8455	0.8256	0.8417
DGCN	0.8633	0.8752	0.8820	0.8408	0.8566
SVD-GCN	0.8790	0.8848	0.9014	0.8603	0.8857
GraphSNN	0.8752	0.8803	0.9006	0.8608	0.8664
GraphSAGE-mean	0.8273	0.8725	0.8669	0.8864	0.8314
GraphSAGE-max	0.8736	0.8602	0.8978	0.8610	0.8631
GraphSAGE-pooling	0.8823	0.8737	0.8890	0.8901	0.8695
SAGE-based JK-met	0.8845	0.8633	0.8910	0.8864	0.8919
GraphSAGE++mean	0.8804	0.8537	0.9017	0.8813	0.8729
GraphSAGE++Max	0.8716	0.8860	0.9016	0.8792	0.8594
GraphSAGE++DA	0.8848	0.8608	0.8821	0.8915	0.8655
GraphSAGE++DAC	**0.9175**	0.8531	0.9188	**0.8992**	0.8591
GraphSAGE++DAMC	0.8652	**0.8957**	**0.9223**	0.8685	**0.9240**

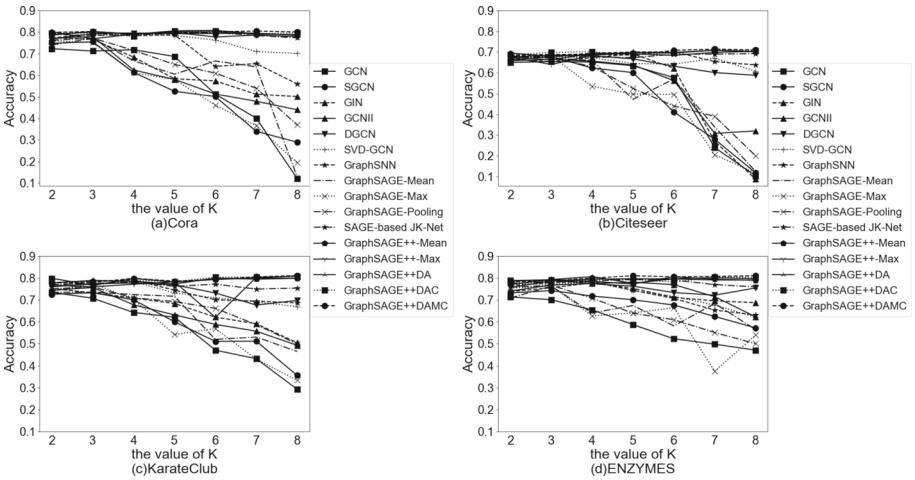

Fig. 8 Accuracy values of different algorithms as K values increase on different datasets

decreasing trend in the acc score after the value of K is 6. In addition, compared to the baseline algorithms, our algorithms consistently maintain high accuracy scores across various values of K, with the GraphSAGE++DAMC variant demonstrating significant advantages. Specifically, on the Pubmed dataset, GraphSAGE++DAMC outperforms GraphSAGE++Mean, GraphSAGE++Max, and GraphSAGE++DA by 3–4% in terms of accuracy.

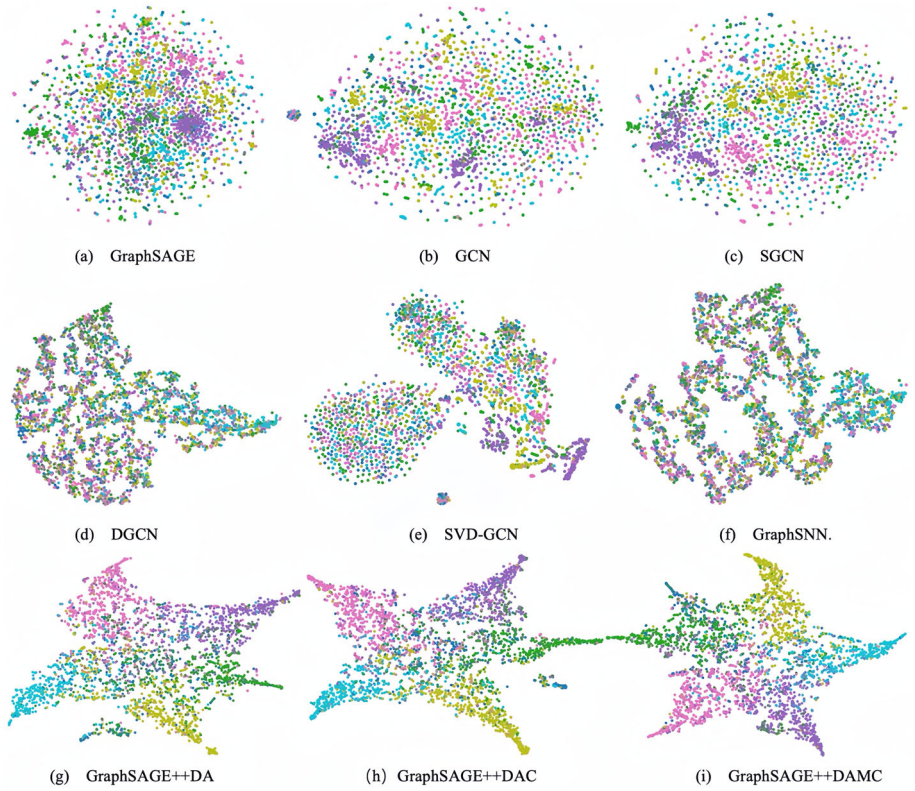

Fig. 9 Visualization results on the Citeseer dataset. Each dot represents a document and the color of the dot indicates the category

4.3.3 Visualization

We first learn the representation vectors of the vectors by different models and then map the representation vectors to a two-dimensional space using t-SNE [38], where the vertices of different classes are labeled with different colors so that the ideal visualization result should be that the vertices of the same class are closer together.

The visualization results are shown in Fig. 9. We choose the Citeseer dataset as the dataset for the visualization task, and we can see that as far as the baseline algorithm is concerned, effective clustering occurs only in small and scattered areas, and the total overall view is still confusing and indistinguishable. Our algorithm is able to distinguish different points more clearly, and in contrast to the other algorithms, the visualization results of GraphSAGE++DAMC are much more outstanding. The experimental results show that our algorithmic framework is a more desirable global graph structure.

4.4 Ablation Experiment and Analysis

The ablation experiments aim to analyze and evaluate the impact of different components in the graph neural network model on the performance of the model in order to reveal the

Table 11 Comparison of the accuracy of different GraphSAGE++ models in ablation experiments

Model	Accuracy (%)
GraphSAGE++DA (with proportional weights)	79.5
GraphSAGE++DAC (with proportional weights)	81.3
GraphSAGE++DAMC (with proportional weights)	**82.2**
GraphSAGE++DA (without proportional weights)	76.3
GraphSAGE++DAC (without proportional weights)	77
GraphSAGE++DAMC (without proportional weights)	78.4

Table 12 Comparison of accuracy between GraphSAGE and GraphSAGE++models in ablation experiments

Model	Accuracy (%)
GraphSAGE-Mean (without concat)	74.2
GraphSAGE-Max (without concat)	74.1
GraphSAGE-Pooling(without concat)	76.8
GraphSAGE++DA (with concat)	80.8
GraphSAGE++DAC(with concat)	81.6
GraphSAGE++DAMC (with concat)	**82.4**

role and importance of these components in the task. n this section, we evaluate the impact of these components on the overall performance of the Cora dataset by removing the use of proportional weight allocation and concatenating strategies. The value of K is set to 3.

Table 11 shows the impact of using a proportional weight allocation strategy on the results of three different concatenating algorithms in GraphSAGE++. We use Eq. (2) to calculate the dimensionality of each layer representation. Through experimental comparison, it was found that when we use a proportional weight allocation strategy, GraphSAGE++DA, GraphSAGE++DAC, and GraphSAGE++DAMC all achieved better results. This also confirms our viewpoint in Sect. 3.3 that neighbors with different hop numbers have different impacts on nodes, and a fixed dimensionality is allocated according to the weight proportion for each layer representation, This effectively preserves the important representation information in the node vector.

Table 12 shows the impact of using concatenating strategies on the results. Through experiments, it was found that the method using concatenating strategy (GraphSAGE++) showed a greater performance improvement compared to the method not using concatenating strategy (GraphSAGE). This also confirms our viewpoint that we introduce strategies which actively combine double aggregations with concatenation helps enhance the distinction of structural information and improve the model's expressive capabilities.

5 Conclusion

In this work, we have proposed the GraphSAGE++ framework that considers multi-scale graph structure information by a weighted multi-dimensional mechanism, effectively capturing both local and the global structure information. The final representation preserves neighborhood information from 1 to K hops, thus overcoming the common over-smoothing issue seen in deep network architectures of traditional graph neural networks successfully.

In addition, the strategies combining double aggregations with concatenation are introduced to better distinguish the structural information. It has been shown experimentally that the method is effective in improving the expressive ability of the model. The experimental results of this research confirm that GraphSAGE++ demonstrates superior performance across various datasets in tasks such as vertex classification, link prediction, and visualization. Notably, GraphSAGE+DAMC outperforms existing methods across multiple evaluation metrics.

Despite the results of our research, there are still some challenges and room for improvement. The proportional allocation of dimensions to each layer can be combined with the PageRank [39, 40] algorithm, which utilizes the linking structure between nodes to be able to better capture the correlation and importance between nodes. Furthermore, optimizing model training efficiency and extending our framework to accommodate increasingly intricate graph structures remain as pertinent objectives for forthcoming research endeavors.

Acknowledgements This work was supported by The Natural Science Foundation of Fujian Province of China, No. 2023J01922, 2021J011007 and 2021J011008; Headmaster Fund of Minnan Normal University No. KJ19009; the National Natural Science Foundation of China, No. 61762036, 62163016; Advanced Training Program of Minnan Normal University, No.MSGJB2023015.

Author Contributions JWE: Conceptualization, Methodology, Data curation, Software, Writing–original draft, Writing–review & editing. YLZ. Conceptualization, Methodology, Supervision, Writing–review & editing. SYY: Writing–review & editing. HW: Writing–review & editing. XWX: Supervision, Writing–review & editing. XX: Supervision, Writing–review & editing.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Tang J, Qu M, Wang M, Zhang M, Yan J, Mei Q (2015) Line: large-scale information network embedding. In: Proceedings of the 24th international conference on world wide web, pp 1067–1077
2. Wang X, Salim FD, Ren Y, Koniusz P (2020) Relation embedding for personalised translation-based poi recommendation. In: Advances in knowledge discovery and data mining: 24th Pacific-Asia conference, PAKDD 2020, Singapore, May 11–14, 2020, Proceedings, Part I 24, pp 53–64. Springer
3. Yu PS, Han J, Faloutsos C (2014) Link mining: models, algorithms, and applications. In: Link mining
4. Berahmand K, Nasiri E, Rostami M, Forouzandeh S (2021) A modified deepwalk method for link prediction in attributed social network. Computing 103:2227–2249
5. Berahmand K, Nasiri E, Forouzandeh S, Li Y (2022) A preference random walk algorithm for link prediction through mutual influence nodes in complex networks. J King Saud Univ Comput Inf Sci 34(8):5375–5387
6. Wu Z, Pan S, Chen F, Long G, Zhang C, Philip SY (2020) A comprehensive survey on graph neural networks. IEEE Trans Neural Netw Learn Syst 32(1):4–24
7. Liu M, Gao H, Ji S (2020) Towards deeper graph neural networks. In: Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining, pp 338–348

8. Chen D, Lin Y, Li W, Li P, Zhou J, Sun X (2020) Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In: Proceedings of the AAAI conference on artificial intelligence, vol 34, pp 3438–3445
9. Keriven N (2022) Not too little, not too much: a theoretical analysis of graph (over) smoothing. Adv Neural Inf Process Syst 35:2268–2281
10. Min Y, Wenkel F, Wolf G (2020) Scattering gcn: overcoming oversmoothness in graph convolutional networks. Adv Neural Inf Process Syst 33:14498–14508
11. Chen M, Wei Z, Huang Z, Ding B, Li Y (2020) Simple and deep graph convolutional networks, pp 1725–1735. PMLR
12. Li G, Muller M, Thabet A, Ghanem B (2019) Deepgcns: can gcns go as deep as cnns? In: Proceedings of the IEEE/CVF international conference on computer vision, pp 9267–9276
13. Xu K, Li C, Tian Y, Sonobe T, Kawarabayashi K-i, Jegelka S (2018) Representation learning on graphs with jumping knowledge networks. In: International conference on machine learning, pp 5453–5462. PMLR
14. Xu K, Hu W, Leskovec J, Jegelka S (2019) How powerful are graph neural networks? In: International conference on learning representations
15. Zhou J, Cui G, Hu S, Zhang Z, Yang C, Liu Z, Wang L, Li C, Sun M (2020) Graph neural networks: a review of methods and applications. AI Open 1:57–81
16. Peng S, Sugiyama K, Mine T (2022) Svd-gcn: a simplified graph convolution paradigm for recommendation. In: Proceedings of the 31st ACM international conference on information & knowledge management, pp 1625–1634
17. Liu Y, Zhang J, Dou R, Zhou X, Xu X, Wang S, Qi L (2022) Vehicle check-in data-driven poi recommendation based on improved svd and graph convolutional network. In: 2022 IEEE smartworld, ubiquitous intelligence & computing, scalable computing & communications, digital twin, privacy computing, metaverse, autonomous & trusted vehicles (SmartWorld/UIC/ScalCom/DigitalTwin/PriComp/Meta), pp 2040–2047
18. Qiu J, Dong Y, Ma H, Li J, Wang K, Tang J (2018) Network embedding as matrix factorization: unifying deepwalk, line, pte, and node2vec. In: Proceedings of the eleventh ACM international conference on web search and data mining, pp 459–467
19. Shanmugam Sakthivadivel S (2019) Fast-netmf: graph embedding generation on single gpu and multi-core cpus with netmf. PhD thesis, The Ohio State University
20. Cao S, Lu W, Xu Q (2015) Grarep: learning graph representations with global structural information. In: Proceedings of the 24th ACM international on conference on information and knowledge management, pp 891–900
21. Feng Q, Liu N, Yang F, Tang R, Du M, Hu X (2022) DEGREE: decomposition based explanation for graph neural networks. In: International conference on learning representations
22. Bouritsas G, Frasca F, Zafeiriou S, Bronstein MM (2022) Improving graph neural network expressivity via subgraph isomorphism counting. IEEE Trans Pattern Anal Mach Intell 45(1):657–668
23. Shervashidze N, Schweitzer P, Van Leeuwen EJ, Mehlhorn K, Borgwardt KM (2011) Weisfeiler-lehman graph kernels. J Mach Learn Res 12(9)
24. Wijesinghe A, Wang Q (2021) A new perspective on" how graph neural networks go beyond weisfeiler-lehman?". In: International conference on learning representations
25. Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: International conference on learning representations
26. Zhang S, Tong H, Xu J, Maciejewski R (2019) Graph convolutional networks: a comprehensive review. Comput Soc Netw 6(1):1–23
27. Wu F, Souza A, Zhang T, Fifty C, Yu T, Weinberger K (2019) Simplifying graph convolutional networks. In: International conference on machine learning, pp 6861–6871 . PMLR
28. Zhang T, Shan H-R, Little MA (2022) Causal graphsage: a robust graph method for classification based on causal sampling. Pattern Recogn 128:108696
29. Hamilton W, Ying Z, Leskovec J (2017) Inductive representation learning on large graphs. Adv Neural Inf Process Syst 30
30. Velickovic P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y (2018) Graph attention networks. In: International conference on learning representations
31. Brody S, Alon U, Yahav E (2022) How attentive are graph attention networks? In: International conference on learning representations
32. Ouyang M, Zhang Y, Xia X, Xu X (2023) Grarep++: flexible learning graph representations with weighted global structural information. IEEE Access
33. Sen P, Namata G, Bilgic M, Getoor L, Galligher B, Eliassi-Rad T (2008) Collective classification in network data. AI Mag 29(3):93–93

34. Li P, Wang Y, Wang H, Leskovec J (2020) Distance encoding: design provably more powerful neural networks for graph representation learning. Adv Neural Inf Process Syst 33:4465–4478
35. Menezes RA, Nievola JC (2015) Predicting the function of proteins using differential evolution. In: IADIS international conference information systems
36. Crane H, Dempsey W (2015) Community detection for interaction networks. Immunology 86(3):469–74
37. Cong W, Ramezani M, Mahdavi M (2021) On provable benefits of depth in training graph convolutional networks. Adv Neural Inf Process Syst 34:9936–9949
38. Maaten L, Hinton G (2008) Visualizing data using t-SNE. J Mach Learn Res 9(11)
39. Gleich DF (2015) Pagerank beyond the web. siam Rev 57(3):321–363
40. Klicpera J, Bojchevski A, Günnemann S (2018) Predict then propagate: graph neural networks meet personalized pagerank. In: ICLR