World Wide Web (2023) 26:421–440
https://doi.org/10.1007/s11280-022-01009-2

Bipartite graph capsule network

Xianhang Zhang[1] · Hanchen Wang[2] · Jianke Yu[2] · Chen Chen[2] · Xiaoyang Wang[2] · Wenjie Zhang[1]

Received: 18 October 2021 / Revised: 26 December 2021 / Accepted: 10 January 2022 /
Published online: 14 February 2022

Abstract
Graphs have been widely adopted in various fields, where many graph models are developed. Most of previous research focuses on unipartite or homogeneous graph analysis. In this graphs, the relationships between the same type of entities are preserved in the graphs. Meanwhile, the bipartite graphs that model the complex relationships among different entities with vertices partitioned into two disjoint sets, are becoming increasing popular and ubiquitous in many real life applications. Though several graph classification methods on unipartite and homogenous graphs have been proposed by using kernel method, graph neural network, etc. However, these methods are unable to effectively capture the hidden information in bipartite graphs. In this paper, we propose the first bipartite graph-based capsule network, namely Bipartite Capsule Graph Neural Network (BCGNN), for the bipartite graph classification task. BCGNN exploits the capsule network and obtains information between the same type vertices in the bipartite graphs by constructing the one-mode projection. Extensive experiments are conducted on real-world datasets to demonstrate the effectiveness of our proposed method.

Keywords Capsule network · Graph neural network · Bipartite graph · Graph classification

This article belongs to the Topical Collection: *Special Issue on Decision Making in Heterogeneous Network Data Scenarios and Applications*
Guest Editors: Jianxin Li, Chengfei Liu, Ziyu Guan, and Yinghui Wu

✉ Hanchen Wang
hanchenw.au@gmail.com

✉ Chen Chen
chenc@zjgsu.edu.cn

Xianhang Zhang
xianhang.zhang@unsw.edu.au

Jianke Yu
jiankey.zjgsu@gmail.com

Xiaoyang Wang
xiaoyangw@zjgsu.edu.cn

Wenjie Zhang
zhangw@cse.unsw.edu.au

[1] University of New South Wales, Kensington, Australia

[2] Zhejiang Gongshang University, Hangzhou, China

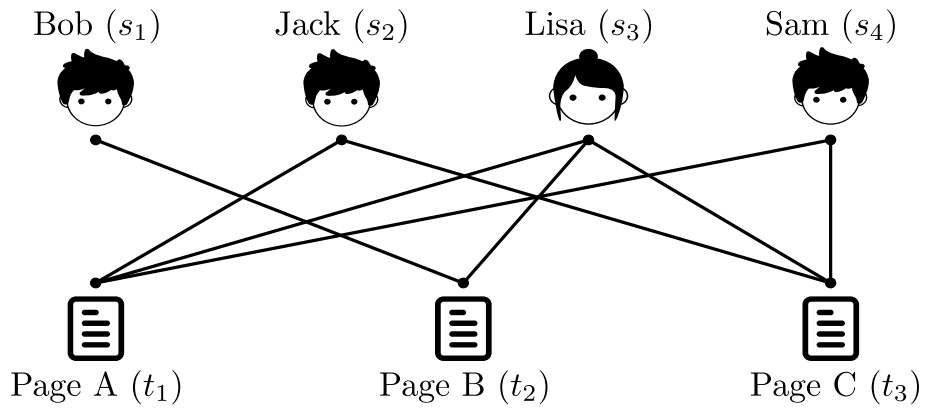


Fig. 1 Example of Bipartite Graph

1 Introduction

In recent years, human society has become increasingly informative, and new technologies that produce massive data, such as e-commerce platforms and social networks, have influenced every aspect of our daily life. Therefore, graph, an important representation structure of big data, has attracted considerable attention. Due to the increasing popularity of graph data, numerous research works have been devoted to mining the valuable information in graph-structured data. Existing research works mainly focus on the unipartite graph, in which the connections can link any pair of entities. However, compared to the unipartite graphs, multipartite graphs, such as bipartite graphs, received less attention from the academic community regardless of their popularity and ubiquity in real-life applications. A bipartite graph has two mutually independent vertex classes and edges only exist between vertices of different classes. For example, the users-page relationships between users and pages of Wiktionary can be represented by a bipartite graph, where edges indicate the editing action from the users on pages. In such graph, users (*resp.* pages) can be related to multiple pages (*resp.* users), but there is no user-user or page-page connection. Figure 1 presents an example of a bipartite graph of the user-page relationships. Bob (s_1) and Lisa (s_3) collaborate to edit the page B (t_1), while Jack (s_2), Lisa (s_3) and Sam (s_4) collaborate to edit the pages A (t_1) and C (t_3).

Due to the ubiquitous properties of bipartite graphs, the classification task of bipartite graphs has become a fundamental tool in various fields [5]. For example, in the user-page bipartite graph such as Wiktionary, users and corresponding pages in different languages forms different bipartite graphs. Similarly, there can be different bipartite graphs of the user-page relationship for the edit relationship under different topics. In these cases, the graph classification on the bipartite graphs can be utilized to determine the languages and topics preference of the users, and hence improve the user experience. Another example of the bipartite graph classification task can be used for money laundry detection. Considering the directed edges between a known cycle for money laundry in e-commerce platforms such as Amazon, we can learn the feature representations of these bipartite graphs and further use them to detect other potential money laundry cycles. In addition to anti-money laundering, the classification of the bipartite graph can also solve many other problems in e-commerce. For example, some unscrupulous

merchants will look for buyers to initiate bogus transactions. The merchants only need to mail some empty packages to the buyers and pay small commissions. The users can make many positive comments about these businessmen's goods to increase the exposure of the interests in the e-commerce platform and increase sales. The bipartite graph classification algorithm can help e-commerce platforms find these unscrupulous merchants. In recent years, some merchants have used the recommendation mechanism of e-commerce platforms to "Ride Item's Coattail" attacks have also become a matter of concern. The bipartite graph classification algorithm can also distinguish these cheating merchants. In addition, it is feasible to represent the interactions between secondary structures of proteins as bipartite graphs. Moreover, bipartite graph classification can be used as a basis for finding common substructures in proteins [26]. Thus, this task can also play an important role in protein discovery.

There have been a lot of work investigated in the graph classification problem. The graph classification problem is usually more complex than vertex classification problem, since more and higher-order information should be considered. Though traditional kernel methods and GNN methods have received great success in the vertex classification task [7, 21], they cannot be directly adapted to the graph classification task.

Based on graph neural networks and capsule networks, some prominent methods [2, 16, 30, 42] have been proposed for the graph classification task. For example, [10] not only utilizes a powerful neural network, but also separates numerous important features while still keeping them independent during training. It allows the model to capture hidden factors more clearly, which makes it achieve higher accuracy on the graph classification task. Further, HCGNN [39] takes the hierarchical information in the graph into account based on the capsule network and continuously synthesizes numerous fine information into more concentrated information, so that the final result can retain the details of the graph structures better, which has outstanding performance of graph classification. However, these methods only specialize in the classification of unipartite graphs. If they are directly applied to bipartite graph classification, the relationships between vertices of the same type cannot be fully retained. This is because there is no connection between vertices of the same type in the bipartite graph, and most methods perform the propagation along the edges to capture the relationship between vertices.

Compared to the traditional scalar-based neural network, capsule network, a vector-based neural network, can represent features using mutually independent sets of vectors [38]. As a result, capsule network can characterize the information of vertex or graph better. Therefore, the capsule network is the basis for our model to obtain the bipartite graph structure information better. It makes capsule network crucial in the work of bipartite graph classification.

Contributions In this paper, we propose a novel method, named *Bipartite Capsule Graph Neural Network* (BCGNN), to achieve classification performance better on bipartite graphs. To preserve the structure, nature and labeling information of the bipartite graph, BCGNN creates the connections between vertices of the same type to build it's one-mode projection. Then, it captures the features and performs better by using the hierarchical capsule network. Specifically, we first decide whether to establish connections between pairs of vertices in the same type depending on the number of their common neighbors. Then, to represent the overall structural information of the bipartite graph, the structural information in the one-mode projection is extracted layer by layer using hierarchical capsule network. Finally, class capsules at the last layer are used to perform the bipartite graph classification task. The main contributions of the paper are summarized as follows:

Table 1 Notation table

Notation	Description
G	Bipartite Graph
$\mathcal{V}, \mathcal{E}, \mathcal{T}$	Vertices, edges and timestamps in Graph
A, D	Adjacency matrix and diagonal degree matrix of G.
c, C	Weight parameter of votes and corresponding matrix.
I	Identity Matrix
K	Disentangle Feature Number
$\mathcal{L}$	Loss function
T	Parameters for Tagging Labels
W	Weight Matrix in neural network.
Z, z	Hidden Features and its reshaped vectors.
Θ	Capsule unit in the network.
Γ	Label Set of Graphs
d	Dimension of Capsules
L	The Number of Layers.
μ	Vote
σ	Nonlinear activation function
γ	Label of Graph
β, λ, m	Hyper-parameters for adjusting the importance of loss functions

- To the best of our knowledge, we are the first to design graph neural networks on bipartite graphs for graph classification task based on capsule networks.
- Our model combines hierarchical capsule network and one-mode projection, which allows us to better capture the relationships between vertices of the same type in a bipartite graph and preserve the structure information of the bipartite graph.
- Extensive experiments on real-world graphs prove that BCGNN outperforms the state-of-the-art baseline methods, in terms of the bipartite graph classification task.

Organization The rest of the paper is organized as follows. We present the related concepts in Section 2. In Section 3, we introduce the model developed. We report the experiment results on real-world datasets in Section 4. Finally, we review the related work in Section 5 and conclude the paper in Section 6.

2 Preliminaries

In this section, we introduce some key definitions and important notations used in this paper. Table 1 summarized the important notations frequently used throughout the paper.

Definition 1 (Bipartite Graph) A bipartite graph can be denoted as $G = (\mathcal{V}_S, \mathcal{V}_T, \mathcal{E})$, where $\mathcal{V}_S = \{s_1, s_2, ..., s_m\}, \mathcal{V}_T = \{t_1, t_2, ..., t_n\}$ are the mutually exclusive vertices sets. $\mathcal{E} \subset \mathcal{V}_S \times \mathcal{V}_T$ is a set of edges that connect vertices between two partitions.

It is important to explain that the bipartite graphs of the same type that we use are all subgraphs of a certain dynamic bipartite graph. Similarly, a dynamic bipartite graph can be defined as $G_t = (\mathcal{V}_S, \mathcal{V}_T, \mathcal{E}, \mathcal{T})$, where $\mathcal{T}$ is the timestamp set containing the timestamps corresponding to all connection moments. For convenience, we denote total vertex set as $\mathcal{V} = \mathcal{V}_S \cup \mathcal{V}_T$ and denote total number of vertices in the bipartite graph by $|\mathcal{V}| = |\mathcal{V}_S| + |\mathcal{V}_T|$.

Definition 2 (One-Mode Projection) One-mode projection on the bipartite graphs aims to construct a projection graph that exist links between the vertices of the same type, *i.e.,* to build graphs $G_S = (\mathcal{V}_S, \mathcal{E}_S)$, $G_T = (\mathcal{V}_T, \mathcal{E}_T)$ where $\mathcal{E}_S \subset \mathcal{V}_S \times \mathcal{V}_S$ and $\mathcal{E}_T \subset \mathcal{V}_T \times \mathcal{V}_T$. And their adjacency matrices are $\boldsymbol{A}_S \in \mathbb{R}^{|\mathcal{V}_S| \times |\mathcal{V}_S|}$ and $\boldsymbol{A}_T \in \mathbb{R}^{|\mathcal{V}_T| \times |\mathcal{V}_T|}$.

Definition 3 (Bipartite Graph Classification) A bipartite graph classification problem can be defined as the following. A learning machine receives a set of N training examples $(G_1, y_1), (G_2, y_2), (G_3, y_3), ..., (G_N, y_N)$, where each example (G_i, y_i) is given as a pair of a bipartite graph G_i and the class y_i, which is the label of the graph [13]. The bipartite graph classification problem is the problem of inferring the class label y_i corresponding to the graph G_i.

Graph Neural Networks Existing graph neural networks usually adopt an aggregate and combine scheme as follows:

$$z_u^{(k)} = \mathcal{COM}^{(k)}(z_u^{(k-1)}, \mathcal{AGG}^{(k)}\{z_{u'}^{(k)}; u' \in N(u)\}), \tag{1}$$

where $z_u^{(k)}$ is the vertex representation of vertex u at k^{th} layer of the graph neural network, $\mathcal{AGG}$ is the aggregation that iteratively updates the representation of vertex u by aggregating the representations of its neighbors, and $\mathcal{COM}$ is the combine operation that updates the representation of vertex u by the aggregated representations and its own representation $z_u^{(k-1)}$ from previous layer. The main difference between graph neural networks is the design of the aggregate and combine mechanism.

3 Model

In this section, we introduce the details of BCGNN. Section 3.1 introduces the framework of our model; Section 3.2 describes how to create edges between vertices of the same type using the one-mode projection; Sections 3.3 and 3.4 introduce the capsule network in detail and the following Section 3.5 illustrates the learning objective with auxiliary graph reconstruction loss.

3.1 Framework

Different from traditional graph neural networks, capsule networks use activity vectors or pose matrices to represent entities. As a result, capsule networks are able to isolate numerous hidden factors and discern relationships among them. Therefore, capsule networks can be very advantageous when being applied on the graphs with complex structures. However, due to the nature of bipartite graphs, vertices that are supposed to share the same type property lack connections with each other. Therefore, capsule networks cannot reach satisfactory performance if being applied on the bipartite graphs for graph classification

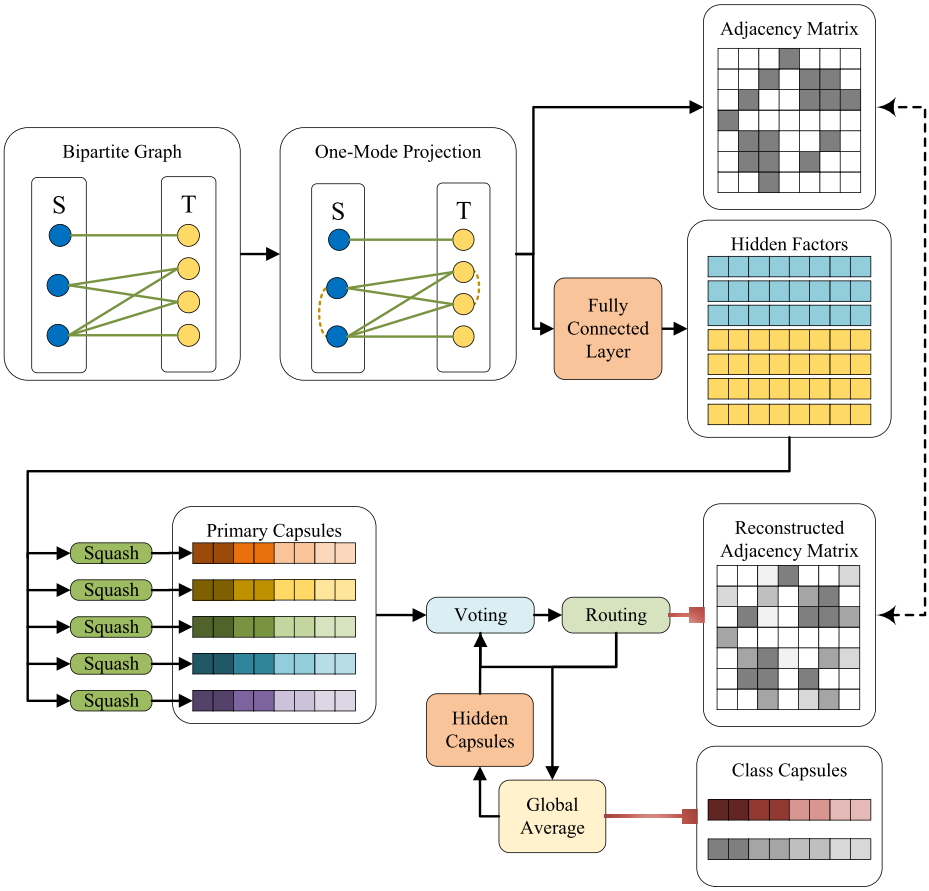


Fig. 2 The Framework of Proposed BCGNN

task since it cannot perform the message passing properly when there is no edge between vertices in the same set. To solve this problem, in this paper, we propose an effective model BCGNN to optimize the performance of conventional capsule network on bipartite graph classification task. BCGNN first generates edges between vertices in the same type based on the number of common neighbors between them. Consequently, BCGNN converts the bipartite graph to its one-mode projection, which enables the GNN part of the capsule network to better extract information between vertices of the same type. With the built one-mode projection, we design the graph capsule network on the bipartite graph to preserve the interaction relationship between the vertices within the same set and in two different sets Figure 2.

3.2 One-mode projection

Since the features of vertices are going to be aggregated along edges in capsule networks, the direct use of capsule networks on bipartite graphs usually results in unsatisfactory performance. Therefore, to enable our model to obtain relationships not only between two

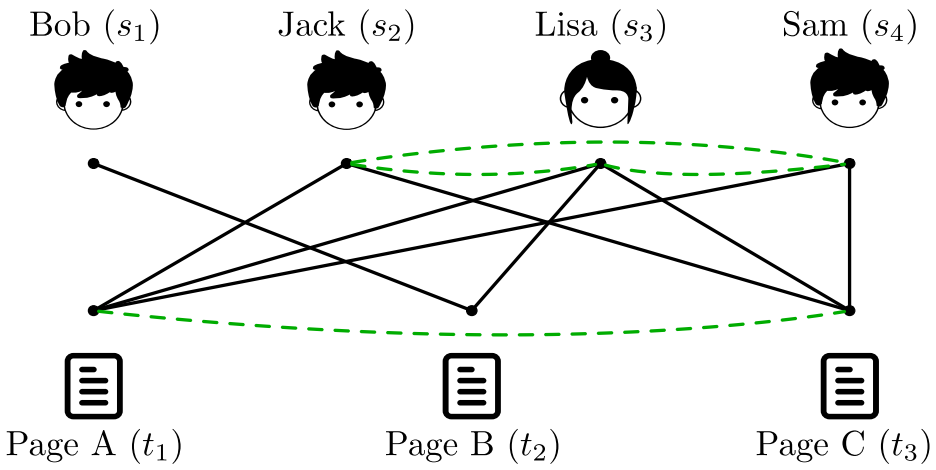


Fig. 3 One-Mode Projection of Bipartite

different vertex sets but also between vertices in the same type, we first generate the one-mode projection of the bipartite graph as the input of the capsule network. The basic idea of generating one-mode projections for bipartite graphs is to determine the number of common neighbors of all possible pairs of vertices in the same type. And then it adds connections between pairs of vertices whose number of common neighbors is greater than a certain threshold. Figure 3 shows the one-mode projection of the bipartite graph illustrated in Figure 1 with threshold value of 2. Since there are two common neighbors of s_2 and s_3, s_2 and s_4, s_3 and s_4, and three common neighbors between t_1 and t_3, which are equal to or greater than the threshold, then connections are established between these pairs of vertices. Although there is one common neighbor between s_1 and s_3, the number of common neighbor does not reach the threshold, so there is no connection established between them.

The implementation can be done by first finding all possible vertex pairs consisting of two vertices in the same type. Then it counts the number of common neighbors of two vertices in all vertex pairs. Finally, we establish connections between all pairs of vertices whose number of common neighbors is greater than or equal to a threshold value. However, the time complexity of this method for establishing edges between vertices of the same type is $O(|\mathcal{V}| \times |\mathcal{V}| \times |\mathcal{E}|)$, which is cost-prohibitive. Therefore, instead of obtaining the one-mode projection in that way, we will use a more efficient method indicated in Algorithm 1. The details are presented as follows.

Since the way to generate connections between vertices in the same part ($\mathcal{V}_S$ or $\mathcal{V}_T$) is the same for both parts of vertices, we introduce how to build connections between $s \in \mathcal{V}_S$ and the way to connect between $t \in \mathcal{V}_T$ are the same. First, each vertex s_i in vertex set $\mathcal{V}_S$ is obtained, and neighboring vertices set of s_i, denoted by $\mathcal{N}(s_i)$ is obtained by utilizing the edge set $\mathcal{E}$, The respective set of neighbor vertices $\mathcal{N}(t)$ of all vertices $t \in \mathcal{N}(s_i)$ can also be obtained. Due to the nature of bipartite graph, there will be no connection between vertices of the same type, so after conducting the above operation, the obtained vertices s_i and $s_j \in \mathcal{N}(t), s_j \neq s_i$ that are 2 hops away from each other must be a pair of vertices of the same type that have a set of common neighbors $\mathcal{N}(t) \subseteq \mathcal{V}_T$. After that, it is easy to count the number of neighbors existing between each pair of vertices in a container of size $|\mathcal{V}_S| \times |\mathcal{V}_S|$ to judge whether the number of common neighbors between the pair of vertices

reaches the threshold, i.e., whether a connection needs to be added between the pair of vertices. The time complexity of this algorithm is $O(|\mathcal{V}| \times |\mathcal{N}(s_i)| \times |\mathcal{E}|)$, where $|\mathcal{N}(s_i)| \ll |\mathcal{V}|$. Further analysis shows that Algorithm 1 is essentially performing a depth-first search with the depth of 2, so in fact, each vertex only needs to traverse at most $|E|$ edges. The time complexity of this algorithm is equivalent to $O(|\mathcal{V}| \times |\mathcal{E}|)$. The process of building the one-mode projection adjacency matrix on vertices in $\mathcal{V}_S$ is summarized in Algorithm 1. Using the above approach, it is possible to obtain graphs G_S, G_T and their adjacency matrices $\boldsymbol{A}_S \in \mathbb{R}^{|\mathcal{V}_S|\times|\mathcal{V}_S|}$, $\boldsymbol{A}_T \in \mathbb{R}^{|\mathcal{V}_T|\times|\mathcal{V}_T|}$, which only contain vertices of the same type. Finally, we can obtain the one-mode projection of the original bipartite graph, whose adjacency matrix can be represented as: $\boldsymbol{A}_{OM} = \boldsymbol{A}_O + \begin{bmatrix} \boldsymbol{A}_S & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A}_T \end{bmatrix}$, where $\boldsymbol{A}_{OM}$ is the adjacency matrix of the graph after one-mode projection and $\boldsymbol{A}_O \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ is the adjacency matrix of the original bipartite graph.

Algorithm 1 Create links between the vertices in $\mathcal{V}_S$.

Input: the set of vertices $\mathcal{V}_S$, edges of graph $\mathcal{E}$, threshold of common neighbors ξ
Output: the generated edge set $\mathcal{E}_S$ for connecting vertices in $\mathcal{V}_S$

$\eta \leftarrow 0$;
$\mathcal{E}_S \leftarrow \phi$;
for *each* $s_i \in \mathcal{V}_S$ **do**
  $\mathcal{N}_{s_i} \leftarrow ConnV(\mathcal{E}, s_i)$; #All vertices connected with s_i
  for *each* $t \in \mathcal{N}_{s_i}$ **do**
    $\mathcal{N}_t \leftarrow ConnV(\mathcal{E}, t) - s_i$;
    for *each* $s_j \in \mathcal{N}_t$ **do**
      if $\eta_{s_i,s_j} \geq \xi$ **then**
        continue;
      end
      $\eta_{s_i,s_j} \leftarrow \eta_{s_i,s_j} + 1$;
      if $\eta_{s_i,s_j} \geq \xi$ **then**
        $\mathcal{E}_S \leftarrow \mathcal{E}_S \cup Edge(s_i, s_j)$;
      end
    end
  end
end
return $\mathcal{E}_S$

3.3 Graph capsule framework

The problem of graph classification is based on the classification of structures of individual graphs. Conventional GNN model can also extract features from graph structure and attribute information for downstream learning objectives, such as vertex classification and link prediction. However, conventional GNNs cannot handle the heterogeneous information of the graphs, and cannot capture the hierarchical structure in the graph either. Consequently, conventional GNN lacks the capability to obtain better performance on the graph classification problem, especially in the classification tasks on the graphs with complex structure and information such as bipartite graphs. Different from conventional GNN, the feature vectors in capsule networks are disentangled into multiple vectors to represent different classes of features, and parameters of multi-layer perceptrons used for each disentangled feature vector are independent with each other during training, i.e., the parameters for each

feature are not shared in the neural network. Therefore, capsule networks have significant advantages over conventional GNN for graph classification problems.

In order to handle the different feature information embedded in the graph, we pass the feature vector of each vertex through multiple sets of mutually independent fully connected layers and activation functions to obtain multiple mutually independent features for representing different hidden factors. Then we use these factors to obtain the most primitive capsules that will be used afterwards. Specifically, given a vertex i in a bipartite graph G which have a feature vector $x_i \in \mathbb{R}^d$. We need to pass each disentangled feature vector through a fully connected layer with different parameters and a nonlinear activation function to obtain the most primitive capsule, which is formulated as:

$$\boldsymbol{Z}_{i,k} = \sigma(\boldsymbol{W}_k^T x_i) + b_k, \tag{2}$$

where $\boldsymbol{Z}_{i,k} \in \mathbb{R}^{\frac{d}{K}}$ is the k^{th} hidden feature of vertex i, $\boldsymbol{W}_k \in \mathbb{R}^{d \times \frac{d}{K}}$ and $b_k \in \mathbb{R}^{\frac{d}{K}}$ are k^{th} learnable weight matrix and bias, and each vertex has $\boldsymbol{K}$ hidden features. σ denotes the activation function. In (2), the feature vector i can be considered to have been converted into a vector set of feature vectors containing $\boldsymbol{K}$ hidden features. As a result, the capsule of vertex i is $\boldsymbol{Z}_i \in \mathbb{R}^{\boldsymbol{K} \times \frac{d}{K}}$. For simplicity, $\boldsymbol{Z}_i$ can be reshaped to the vector format $z_i \in \mathbb{R}^d$. As mentioned in [39], the capsule length of the corresponding disentangled entity represents the probability of the existence of the hidden feature it corresponds to, and the longer the capsule length, the higher the probability of the existence of the entity. Therefore, we need to preserve the vector direction while normalizing the length of the vector, and the squash function is implemented as follows:

$$\boldsymbol{\Theta}_i = squash(z_i) = \frac{|z_i|^2}{1 + |z_i|^2} \times \frac{z_i}{|z_i|} \tag{3}$$

Thus, we can transform the feature vector of each vertex into the lowest and most preliminary capsule $\boldsymbol{\Theta}_i^{(1)} \in \mathbb{R}^{d_1}$ and vertices can be converted into preliminary capsule set $\boldsymbol{\Theta}^{(1)} \in \mathbb{R}^{|\mathcal{V}| \times d_1}$, where d_1 is the overall length of any capsule in the preliminary capsule layer. Eventually, with the reducing number of capsules, we can obtain the final graph classification result while preserving the hierarchical graph structural information.

3.4 Graph capsule layers

In this section, we introduce the layers in our capsule network in detail. Obviously, hierarchical information plays an important role in graph classification. For example, accurate inference of some important substructures (functional groups) within a protein (chemicals) can greatly help us predict the properties of the protein or chemical compounds. In this paper, we utilize the capsule network to preserve the hierarchical information in the graph, and hence improve the graph classification performance of our proposed model. In order to obtain the hierarchical information, we need to map the bottom capsule to the top capsule layer by layer, continuously extract and integrate the structural information hidden in different levels. Finally, we obtain the last layer of capsules, in which the number of capsules is equal to the number of graph types. Based on the length of these capsules, the class of the graph can be predicted. We refer to this last layer as class capsule layer. More specifically, each capsule in the previous layer needs to generate a corresponding vote for each capsule in the later layer to obtain the features. To pass the features with attentional tendency, a weighting parameter needs to

be acquired for each vote. The next layer of capsules is then obtained by weighting and summing these votes. The weighting parameter is computed based on the similarity of these votes from the previous layer to the capsule in the next layer. The higher similarity indicates, the larger weight it has. So features in lower levels can be informatively and hierarchically transmitted to features in higher levels.

First, we need to use GNN to aggregate the N_l capsules of the l^{th} layer for N_{l+1} times, to obtain the votes from each capsule at the l^{th} layer to all the capsules at the $(l+1)^{th}$ layer, where N_l is the number of capsules in the l^{th} layer. As introduced in Section 2, the different GNNs utilizes different aggregate and combine mechanism. Specifically, in this work, we choose graph convolutional network [14] (GCN) as the GNN method to get votes. GCN aggregates the neighbor representations by summation over a normalized adjacency matrix $\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}$, where $\widetilde{\boldsymbol{A}}$ is the adjacency matrix $\boldsymbol{A}$ with self-loop as: $\widetilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_N$, $\widetilde{\boldsymbol{D}}$ is a diagonal degree matrix of $\widetilde{\boldsymbol{A}}$ where $\boldsymbol{D}_{ii} = \sum_j \widetilde{\boldsymbol{A}}_{ij}$. Consequently, GCN can be formulated with the following equations:

$$\boldsymbol{H}^{(l+1)} = \sigma(\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}), \tag{4}$$

where $\boldsymbol{H}^{(l)}$ is the hidden feature vector at l^{th} layer and $\boldsymbol{H}^{(0)}$ is the input representations of the vertices, $\boldsymbol{W}^{(l)}$ is a trainable weight matrix for l^{th} layer, and σ is the nonlinear activation function. With the help of GCN, BCGNN could capture the neighboring information of the graphs via message passing along the edges. Eventually, the vector representations for the vertices are obtained. In addition, to generate the feature vector of the vertex in the latter layer without losing the feature of the vertex in the current layer, it is necessary to add self-loop to all vertices, so the adjacency matrix used in aggregation is $\widetilde{\boldsymbol{A}}$. The degree matrix $\widetilde{\boldsymbol{D}}$ is also applied on the adjacency matrix for a normalization purpose.

In our capsule network, the capsules in the first layer are directly obtained from vertices in the graph, therefore, there are $|\mathcal{V}|$ capsules in total initially, where $|\mathcal{V}|$ is the number of vertices in the graph G. The GCN is directly used on the graph built by the one-mode projection described in Section 3.2 to obtain their votes for the next layer of capsules. It is worth noting that the graph applied to the capsule network is a one-mode projection of the bipartite graph, and its adjacency matrix is $\boldsymbol{A}^{(1)} \in \mathbb{R}^{|\mathcal{V}|^2}$. Practically, we perform one layer of aggregation on the input graph, so the GCN used in the BCGNN is formulated as follows:

$$\mu_j^{(l)} = \sigma[(\widetilde{\boldsymbol{D}}_{OM}^{(l)})^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}_{OM}^{(l)}(\widetilde{\boldsymbol{D}}_{OM}^{(l)})^{-\frac{1}{2}}\boldsymbol{\Theta}^{(l)}\boldsymbol{W}_j^{(l)}], \tag{5}$$

where $\mu_j^{(l)}$ is the vote of the l^{th} layer's capsules to the $(l+1)^{th}$ layer's capsule j, $\widetilde{\boldsymbol{A}}_{OM}^{(l)} = \boldsymbol{A}_{OM}^{(l)} + \boldsymbol{I}_N$, $\widetilde{\boldsymbol{D}}_{OM}^{(l)}$ is a diagonal degree matrix of $\widetilde{\boldsymbol{A}}_{OM}^{(l)}$, and $\boldsymbol{W}_j^{(l)}$ is a trainable weight matrix for l^{th} layer's capsules and is used to generate the votes for capsule j in $(l+1)^{th}$ layer. The operation of getting a vote is referred as **voting**.

Then, it is required to learn weight parameter c for every vote. In order to ensure that the weights corresponding to the votes in the same layer are normalized, we need to ensure that c sums to 1 for all the votes in the same layer, that means $\sum_{j=1}^{N_{l+1}} c_{i,j} = 1$, where $c_{i,j}$ denotes the weight for vote from capsule i in l^{th} layer to capsule j in $(l+1)^{th}$ layer. For this purpose, we need an auxiliary parameter b to learn the appropriate parameter c. Specifically, after initializing $b \leftarrow 0$, we iteratively perform the following steps for every capsule in the consequent layer:

– **1**. Applying softmax function, transforming b into c as follows:

$$c_{i,j}^{(l)} = \frac{exp(b_{i,j}^{(l)})}{\sum_k exp(b_{i,k}^{(l)})}, \tag{6}$$

where $c_{i,j}^{(l)}$ is the weight parameter of capsule i in the l^{th} layer for the vote of capsule j in the $(l+1)^{th}$ layer, and $b_{i,j}^{(l)}$ corresponds to $c_{i,j}^{(l)}$. Using (6), we can provide a set of weight $c^{(l)}$ for all votes in the l^{th} layer for capsule j in the $(l+1)^{th}$ layer.

– **2**. All the weighted votes of the l^{th} layer for capsule j in the $(l+1)^{th}$ layer are summed and squashed to obtain the feature vector of capsule j as follows:

$$\boldsymbol{\Theta}_j^{l+1} = squash(\sum_i c_{i,j}^{(l)} \mu_{i,j}^{(l)}), \tag{7}$$

where $\mu_{i,j}^{(l)}$ is the vote of capsule i to capsule j.

– **3**. Judging the similarity between the capsule j obtained from (7) and each vote in layer l for capsule j, then update the parameter b according to the similarity as follows:

$$b_{i,j}^{(l)} = b_{i,j}^{(l)} + \mu_{i,j}^{(l)} \boldsymbol{\Theta}_j^{l+1} \tag{8}$$

The act of repeating the above three operations is referred as **routing**.

After iterating the above three operations for R times, the more desirable capsule j in $(l+1)^{th}$ layer and the set of weight parameters $C_j^{(l)} \in \mathbb{R}^{N_l}$ for all the votes corresponding to capsule j are obtained. When all the capsules of $(l+1)^{th}$ layer are obtained, we will also get the parameter matrix $C^{(l)} \in \mathbb{R}^{N_l \times N_{l+1}}$ at the same time. Using $C^{(l)}$, we can get the adjacency matrix of the capsule of $(l+1)^{th}$ layer by the following method:

$$\boldsymbol{A}^{(l+1)} = \boldsymbol{C}^{(l)^T} \boldsymbol{A}^{(l)} \boldsymbol{C}^{(l)} \tag{9}$$

Please note that since $N_{l+1} < N_l$, the number of capsules involved in the computation reduces after each layer. Therefore, BCGNN could learn the representation for the graph by preserving its structural and attribute information hierarchically. After that, by repeating the above operation with the capsules and the adjacency matrix of $(l+1)^{th}$ layer, we can get the capsules of the next layer, until we obtain the class capsule layer for output, which has the same number of capsules as the number of graph classes.

In order to retain and transmit features from the previous layer to the next layer better, drawing on the approach of [23], we add a residual connection at each pair of consecutive capsule layers as follows:

$$\boldsymbol{\Theta}^{(l+1)} \leftarrow \widetilde{\boldsymbol{\Theta}}^{(l+1)} + \boldsymbol{M}(\boldsymbol{\Theta}^{(l)}), \tag{10}$$

where $\boldsymbol{M}(\cdot)$ indicates the global average function. $\widetilde{\boldsymbol{\Theta}}^{(l+1)}$ indicates the capsule layer of the $l+1$ layer that has not yet weighted the information of the previous capsule layer. $\boldsymbol{\Theta}^{(l+1)}$ is the final $l+1$ capsule layer.

3.5 Learning objectives

Once the class capsules in the output layer $\boldsymbol{\Theta}^L \in \mathbb{R}^{|\Gamma| \times d_L}$ are obtained, where Γ is the set of labels for the graph class, the probability for a certain class can be judged by the length of the capsule's feature vector [39]. Thus, the classification loss can be measured by the following margin loss function:

$$\mathcal{L}_m(\boldsymbol{\Theta}^{(L)}) = \sum_{\gamma \in \Gamma} [\boldsymbol{T}_\gamma max(0, m^+ - ||\boldsymbol{\Theta}_\gamma^L||)^2 + \lambda(1 - \boldsymbol{T}_\gamma) max(0, ||\boldsymbol{\Theta}_\gamma^L|| - m^-)^2], \quad (11)$$

where m^+ and m^- are the marginal coefficients which are set to 0.9 and 0.1 respectively in this work, T_γ is the class label indicator which equals to 1 iff $\boldsymbol{\Theta}_\gamma^{(L)}$ has label γ, otherwise $T_\gamma = 0$.

To preserve the original graph structural information during training and to improve the stability of the training, we use reconstruction loss to constrain the training. The core idea is to decode the adjacency matrix of the class capsule layer to obtain a matrix which is close to the adjacency matrix of the initial capsule layer.

Specifically, we take the output class capsule $\boldsymbol{\Theta}^L$ of BCGNN as the input, and use a fully connected network to map this capsule back to a matrix with the dimension as the primary capsules, i.e., $\mathbb{R}^{|\mathcal{V}| \times d_1}$, with the following equation:

$$\boldsymbol{Z}_r = \boldsymbol{\Theta}^{(1)} + (\boldsymbol{W}_r^T \boldsymbol{\Phi}(\boldsymbol{\Theta}^{(L)}) + b_r), \quad (12)$$

where $\boldsymbol{\Phi}$ is the mask operation, $\boldsymbol{W}_r \in \mathbb{R}^{(|\Gamma| \times d_L) \times d_1}$ is a learnable parameter matrix, $b_r \in \mathbb{R}^{d_1}$ is learnable bias vector, and $\boldsymbol{Z}_r \in \mathbb{R}^{|\mathcal{V}| \times d_1}$. Then, rely on the results $\boldsymbol{Z}_r$, a matrix with the same dimensions as the adjacency matrix of the preliminary capsules can be obtained by $\boldsymbol{A}_r = \boldsymbol{Z}_r \boldsymbol{Z}_r^T$, which is the preliminary adjacency matrix obtained by re-decoding the class capsules. Then, reconstruction loss can be implemented as follows:

$$\mathcal{L}_r(\boldsymbol{A}^{(1)}, \boldsymbol{A}_r) = -\frac{1}{N_1^2} \sum_{a=1}^{N_1} \sum_{b=1}^{N_1} [\boldsymbol{A}_{a,b}^{(1)} log(\boldsymbol{A}_{ra,b}) - (1 - \boldsymbol{A}_{a,b}^{(1)}) log(1 - \boldsymbol{A}_{ra,b})] \quad (13)$$

It is worth noting that since $A^{(1)}$ is the adjacency matrix of the one-mode projection of the original bipartite graph, $A \in \{0, 1\}^{N_1 \times N_1}$. We clamp the values greater than 1 in A_r to 1.

Finally, the loss function for optimization is shown below:

$$\mathcal{L} = \mathcal{L}_m(\boldsymbol{\Theta}^{(L)}) + \beta \mathcal{L}_r(\boldsymbol{A}^{(1)}, \boldsymbol{A}_r), \quad (14)$$

where β can be used to adjust the importance of the overall loss function $\mathcal{L}$ with respect to $\mathcal{L}_r$.

4 Experiment

In this section, we experimentally demonstrate the ability of BCGNN to classify bipartite graphs. We attempt to answer the following two research questions:

- **Q1**. Has the utilization of one-mode projection and hierarchical capsule network led to improved classification results?
- **Q2**. How much the proposed method improves the baselines?

To answer the above questions and further validate the superiority of our proposed method, experiments are conducted on seven sets of bipartite graphs which are generated from seven real-world temporal bipartite graphs.

Table 2 Statistics of Datasets

Original Data	$\lvert\mathcal{V}_S\rvert$	$\lvert\mathcal{V}_S\rvert$	$\lvert\mathcal{E}\rvert$	#Graph	Avg.#Vertex	Avg.#Edge
edit-nawiki	844	4,014	70,695	75	12.48	14.11
edit-dvwiktionary	197	1,017	4,497	183	17.35	21.68
edit-ltwikisource	221	1,755	4,944	171	15.12	16.68
edit-mswikibooks	223	1,852	5,593	131	18.37	26.07
edit-sswiktionary	231	1,287	4,703	154	13.71	14.86
edit-bgwikisource	362	3,424	9,712	216	13.46	15.34
edit-tawikiquote	316	2,771	9,857	150	12.89	16.91

4.1 Datasets and baselines

The dataset used in the experiments is generated from seven temporal bipartite graphs:

– **edit-nawiki**[1], **edit-dvwiktionary**[2], **edit-ltwikisource**[3], **edit-mswikibooks**[4], **edit-sswiktionary**[5], **edit-bgwikisource**[6] and **edit-tawikiquote**[7] contain users and pages from the Nauru Wikipedia, the Divehi Wiktionary, the Lithuanian Wikisource, the Malay Wikibooks, the Swati Wiktionary, the Bulgarian Wikisource and the the Tamil Wikiquote, connected by edit events. Each edge represents an edit. And each dataset includes the timestamp of each edit. The statistics of these datasets and the groups of graphs generated through each of them are summarized in Table 2.

To examine the effectiveness of our proposed framework, we compare BCGNN with the following baseline methods:

– **AWE** [11], **WWL** [27] are kernel based graph classification methods.
– **DGCNN** [42], **HaarPool** [33] are state-of-the-art deep neural network methods for graph classification.
– **CapsGNN** [38] is the first work to adapt capsule network to graph neural networks which achieves significant improvement on graph classification task compare to conventional graph classification models.
– **HCGNN** [39] utilizes the capsule network to preserve the hierarchical information in the graphs, and hence has the state-of-the-art performance for graph classification problem.

4.2 Experiment Settings

We generate new bipartite graphs from vertices and edges that appear in the same time slot based on the time-stamped separation of all edges in the temporal bipartite graph. Based on

[1] http://konect.cc/networks/edit-nawiki
[2] http://konect.cc/networks/edit-dvwiktionary
[3] http://konect.cc/networks/edit-ltwikisource
[4] http://konect.cc/networks/edit-mswikibooks
[5] http://konect.cc/networks/edit-sswiktionary
[6] http://konect.cc/networks/edit-bgwikisource
[7] http://konect.cc/networks/edit-tawikiquote

the fact that bipartite graphs generated from the same original bipartite graph have similar structures and attributed information, we group them into the same category, i,e., they have a same label. The specific steps for graph generation are shown in Algorithm 2. In the algorithm, the line 1 sorts the input temporal bipartite graph according to the timestamps of the edges generating the links in ascending order, then get the sequence of ordered edges $\mathcal{E}$ and the timestamps corresponding to these edges $\mathcal{T}$. The purpose of the line 2 is to decide a time slot length, which will be used to split the time interval for the same subgraph. Line 9 ensures that there are no duplicate edges in each subgraph. Line 16 controls the range of the number of edges in each subgraph, therefore, all generated graphs have similar graph size, and line 24 ensure that the generated graphs are connected graphs. If non-connected graphs exist, they are divided into multiple connected graphs and the graphs that do not have the required number of edges are removed. More specifically, for each graph of Ψ_L, we perform the following operations. We deposit each edge into the preparatory graph. The rule for depositing is that if the vertex of the edge appears in one of the preparatory graphs, it is added to that preparatory graph, otherwise, a new preparatory graph is created, and the edge is added. After this operation is performed on all edges, the preparatory graphs with duplicate vertices are merged to obtain the final set of connected graphs.

Algorithm 2 Generate subgraphs from temporal bipartite graphs.

Input: temporal bipartite graph G_t, threshold of the number of edges in result graph L_{min} and L_{max}, the relative time interval parameter used to adjust the selection of subplots $\alpha \in (0, 1)$
Output: a set of bipartite graphs Ψ_L which have the same label

1 $\mathcal{E}, \mathcal{T} \leftarrow SortByTime(G_t)$; #Sorted by time from early to late
2 $\theta = \alpha \times (max(\mathcal{T}) - min(\mathcal{T}))$; #Step size for splitting subgraphs
3 $T_{max} \leftarrow \theta + min(\mathcal{T})$;
4 $\psi_0, \Psi_0 \leftarrow \phi$;
5 $i, k, l \leftarrow 0$;
6 **while** $i<len(G)$ **do**
7 **if** $\mathcal{T}_i \leq T_{max}$ **then**
8 $i \leftarrow i + 1$;
9 **if** $\mathcal{E}_i \notin \psi_k$ **then**
10 $\psi_{k+1} \leftarrow \psi_k \cup \mathcal{E}_i$;
11 $k \leftarrow k + 1$;
12 **end**
13 **end**
14 **else**
15 $T_{max} \leftarrow T_{max} + \theta$;
16 **if** $L_{min} \leq len(\psi_k) \leq L_{max}$ **then**
17 $\Psi_{l+1} \leftarrow \Psi_l \cup \psi_k$;
18 $l \leftarrow l + 1$;
19 **end**
20 $\psi_0 \leftarrow \phi$;
21 $k \leftarrow 0$;
22 **end**
23 **end**
24 $\Psi_L \leftarrow CheckConn(\Psi_L)$; #Make sure all graphs are connected graphs
25 **return** Ψ_L

Consequently, we obtained a graph set with a total of 1080 graphs in seven classes. Among them, the largest class has 216 graphs while the smallest class has 75 graphs. In addition, we made disentangle feature number $\boldsymbol{K} = 4$, routing iteration number $\boldsymbol{R} = 3$, $\lambda = 0.5$, $\beta = 0.1$ and $\boldsymbol{L} = 2$ in our experiments, choose *Adam* as the optimizer with the

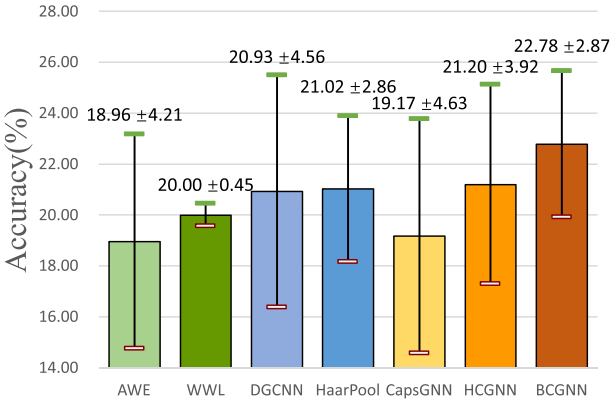


Fig. 4 Experiment Result for Bipartite Graph Classification

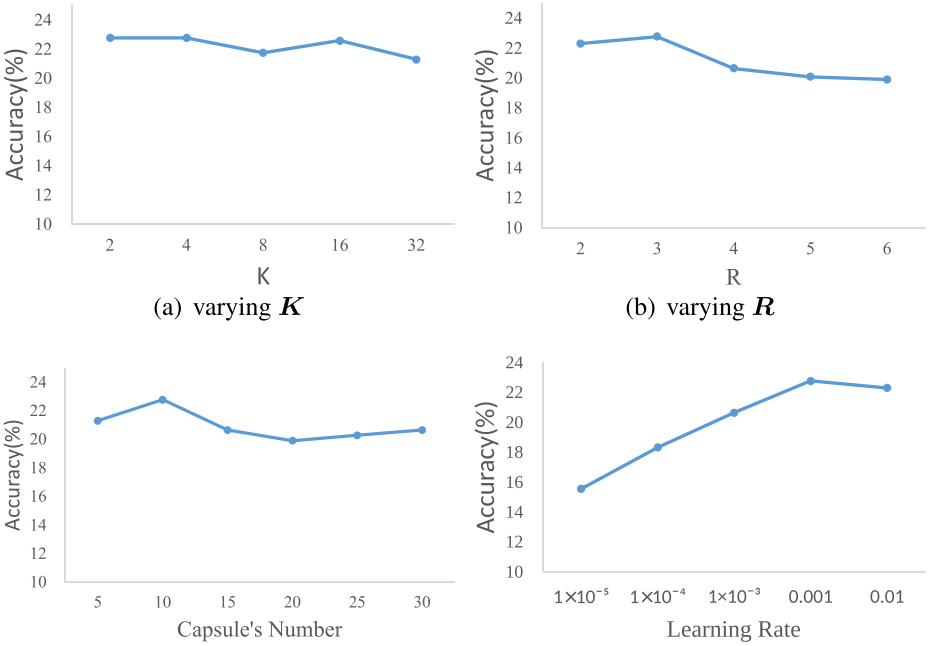
learning rate $lr = 0.001$ and used 10-fold cross-validation to train the model. The capsule dimension was set to 128, while the input feature dimensions of the vertices were generated based on the size of the bipartite graph and the degree of each vertex. In the experiments, the feature vector dimension of each vertex is 106. We take the average of these 10 predictions as the final accuracy and consider their standard deviation as the floating range of accuracy.

4.3 Bipartite graph classification results

The experimental results are presented in Figure 4. Our proposed BCDNN possesses a higher accuracy than all other baselines in completing the bipartite graph classification task.

Among them, AWE, WWL and CapsGNN are less accurate on the graph classification task because they do not consider the hierarchical information of the bipartite graph. It can be seen that the hierarchical information can play an important role in the graph classification problem. It is worth noting the WWL model. Inspired by WL, the WWL algorithm counts the ground distance between all pairs of vertices in two graphs, and then obtains the Wasserstein distance of the two graphs to predict the similarity of the structure between them. This full utilization of vertex features makes the model very robust. However, when applying this model to bipartite graphs, it does not extract hierarchical structure well, nor does it allow information to be exchanged between vertices of the same type well. Therefore, WWL cannot achieve the accuracy of BCGNN. Although DGCNN and HaarPool also consider hierarchical information, HCGNN, with the help of capsule network, is able to integrate the information better. As a result, the accuracy of HCGNN is better than that of DGCNN and HaarPool. Although CapsGNN also uses the capsule network, the results are not satisfactory, which shows that simply using the capsule network when completing the bipartite graph classification task does not yield the desired results.

And compared with HCGNN, BCDNN optimizes the method for the characteristics of bipartite graphs by establishing the one-mode projection for the original bipartite graph. So the preliminary capsules which generated by the same type of vertices can be aggregated better and exchange feature information better among themselves. So BCDNN can achieve higher accuracy when implementing bipartite graph classification than HCGNN. It can be



(a) varying $\boldsymbol{K}$ (b) varying $\boldsymbol{R}$

(c) varying hidden layer capsule number (d) varying learning rate

Fig. 5 Parameter Analysis Results in BCDNN

seen that using a one-mode projection of the bipartite graph before processing it in the GNN related algorithm is an outstanding way to enhance the effect.

4.4 Parameter Analysis

We conduct the parameter analysis experiments on the following parameters: The disentangle feature number $\boldsymbol{K}$, the number of routing iterations $\boldsymbol{R}$, the number of hidden layer capsule and learning rate lr. The analysis results are shown in Figure 5.

For BCDNN, the most important parameter is the disentangle feature number $\boldsymbol{K}$. We tested five values $\{2, 4, 8, 16, 32\}$ as our $\boldsymbol{K}$. The result of tests can be seen in Figure 5(a). We can conclude from the experimental results that $\boldsymbol{K}$ is robust to the accuracy of classification. The best accuracy value is achieved when $\boldsymbol{K}$ is 4. The performance of BCDNN worsens with the increment of $\boldsymbol{K}$. Therefore, $\boldsymbol{K}$ was set to be 4.

In addition, $\boldsymbol{R}$, which determines the number of routing iterations, is also an important parameter. We tested all values from 2 to 6 as the values of $\boldsymbol{R}$. The experimental results are shown in Figure 5(b). From the experimental results, we can learn that the accuracy reaches the highest at $\boldsymbol{R} = 3$. If the value is increased further, the accuracy decreases, so we choose 3 as the value of $\boldsymbol{R}$.

In order to test the effect of the number of capsules in the hidden layer on the experiment, we tested four cases when the number of capsules in the hidden layer was 5, 10, 15, 20, 25 and 30. The experimental results are shown in Figure 5(c). From the experimental results, After the number of capsules is greater than 10, as the growth in the

number of capsules decreases the accuracy of bipartite graph classification, we choose 10 as the value of the number of capsules in our hidden layer, which has better results.

Finally, we tested $\{1 \times 10^{-5}, 1 \times 10^{-4}, 5 \times 10^{-3}, 1 \times 10^{-3}, 0.01\}$ five learning rates. The experimental results can be seen in Figure 5(d). The experimental results show that the learning rate has a greater impact on the accuracy, so we choose the best result 1×10^{-3} as our learning rate.

5 Related work

In this section, we present the related works from the following four perspectives.

Bipartite graph related neural networks Numerous research works [8, 9, 17, 20, 34, 37, 41] have been proposed with the focuses on the analysis of bipartite graph neural networks. Among them, [17, 20, 37, 41] use neural networks on bipartite graphs to implement an efficient recommendation system, while [8, 34] focus on cancer survival prediction and drug-disease association prediction. [9] delves into the vertex representation learning problem. However, these methods are focused on dealing with microscopic vertex and edge information, and cannot be directly used to implement macroscopic graph classification task.

Graph classification Aiming to solve the graph classification problem, a variety of methods [12, 13, 15, 16, 18, 25, 31, 35, 36, 42] are proposed. These works are well implemented for graph classification utilizing the techniques such as mathematical programming[12, 35], multiview learning[36], reinforcement learning[16], feature selection [15], graph kernels[13, 18], and graph neural network[42]. However, these methods are designed for unipartite graphs and cannot be directly generalized to the bipartite graph classification problem.

Bipartite graph analytics Nowadays, with the increasing popularity of bipartite graphs, there are several methods proposed for bipartite graph analytics, such as [1, 3, 4, 22] which are able to find meaningful community structures on bipartite graphs. [26] presents a bipartite graph matching method on protein structure, which is consequently used for protein graph classification application.

Capsule network Recently, a method called capsule network [10] is proposed and achieved state-of-the-art performance on image classification problem. Due to its excellent performance, many methods [6, 19, 29, 32, 40] achieve excellent results on graph related problems by applying capsule network on graphs, and [24, 28, 38] also accomplish outstanding results on graph classification task. However, since these methods are mainly designed for unipartite graphs, there are still few investigations using capsule network that have excellent performance on graph classification problem involving bipartite graphs. In this paper, we utilize one-mode projection and hierarchical capsule network to improve the performance of GNN-based methods on the bipartite graph classification task, and demonstrate that it possesses excellent accuracy.

6 Conclusion

Bipartite graphs are becoming more and more common in practice, but little work has been done on them due to the complexity caused by the bipartite setting. In this paper, we propose the first capsule network on bipartite graphs for graph classification tasks. The proposed BCDNN first applies one-mode projection to bipartite graphs, allowing the capsule

network to better capture information between vertices of the same type consequently. BCDNN significantly improves the accuracy of bipartite graph classification by integrating the bipartite graph classification task based on one-mode projection and hierarchical capsule networks. Extensive experiments on the real-life bipartite graphs within seven classes demonstrate a significant improvement of BCDNN compared to the state-of-the-art methods.

Acknowledgements This work is supported by NSFC 61802345, ZJNSF LQ20F020007 and ZJNSF LY21F020012.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions.

Compliance with ethical standards

Conflicts of interest The authors declare that they have no conflict of interest.

References

1. Alzahrani, T., Horadam, K.J.: Community detection in bipartite networks: Algorithms and case studies. In: Complex Systems and Networks, pp. 25–50. Springer (2016)
2. Bai, L., Cui, L., Jiao, Y., Rossi, L., Hancock, E.: Learning backtrackless aligned-spatial graph convolutional networks for graph classification. IEEE Transactions on Pattern Analysis and Machine Intelligence **44**(2), 783–798 (2020)
3. Barber, M.J.: Modularity and community detection in bipartite networks. Physical Review E **76**(6), 066102 (2007)
4. Beckett, S.J.: Improved community detection in weighted bipartite networks. Royal Society Open Science **3**(1), 140536 (2016)
5. Cai, T., Li, J., Mian, A.S., Sellis, T., Yu, J.X., et al.: Target-aware holistic influence maximization in spatial social networks. IEEE Transactions on Knowledge and Data Engineering (2020)
6. Chen, H., Wang, W., Li, G., Shi, Y.: A quaternion-embedded capsule network model for knowledge graph completion. IEEE Access **8**, 100890–100904 (2020)
7. Chen, J., Zhong, M., Li, J., Wang, D., Qian, T., Tu, H.: Effective deep attributed network representation learning with topology adapted smoothing. IEEE Transactions on Cybernetics, pp. 1–12 (2021)
8. Gao, J., Lyu, T., Xiong, F., Wang, J., Ke, W., Li, Z.: Mgnn: A multimodal graph neural network for predicting the survival of cancer patients. In: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 1697–1700 (2020)
9. He, C., Xie, T., Rong, Y., Huang, W., Li, Y., Huang, J., Ren, X., Shahabi, C.: Bipartite graph neural networks for efficient node representation learning. arXiv:1906.11994 (2019)
10. Hinton, G.E., Sabour, S., Frosst, N.: Matrix capsules with em routing. In: International Conference on Learning Representations (2018)
11. Ivanov, S., Burnaev, E.: Anonymous walk embeddings. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, Proceedings of Machine Learning Research, vol. 80, pp. 2191–2200. PMLR, Stockholmsmässan, Stockholm Sweden (2018). http://proceedings.mlr.press/v80/ivanov18a.html
12. Jin, N., Young, C., Wang, W.: Graph classification based on pattern co-occurrence. In: Proceedings of the 18th ACM conference on Information and Knowledge Management, pp. 573–582 (2009)

13. Kashima, H., Inokuchi, A.: Kernels for graph classification. In: ICDM Workshop on Active Mining, vol. 2002, pp. 36–41 (2002)
14. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv:1609.02907 (2016)
15. Kong, X., Yu, P.S.: Semi-supervised feature selection for graph classification. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 793–802 (2010)
16. Lee, J.B., Rossi, R., Kong, X.: Graph classification using structural attention. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1666–1674 (2018)
17. Li, C., Jia, K., Shen, D., Shi, C.J.R., Yang, H.: Hierarchical representation learning for bipartite graphs. In: IJCAI, pp. 2873–2879 (2019)
18. Li, G., Semerci, M., Yener, B., Zaki, M.J.: Effective graph classification based on topological and label attributes. Statistical Analysis and Data Mining: The ASA Data Science Journal **5**(4), 265–283 (2012)
19. Li, J., Li, S., Zhao, W.X., He, G., Wei, Z., Yuan, N.J., Wen, J.R.: Knowledge-enhanced personalized review generation with capsule graph neural network. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, pp. 735–744 (2020)
20. Li, Z., Shen, X., Jiao, Y., Pan, X., Zou, P., Meng, X., Yao, C., Bu, J.: Hierarchical bipartite graph neural networks: Towards large-scale e-commerce applications. In: 2020 IEEE 36th International Conference on Data Engineering (ICDE), pp. 1677–1688. IEEE (2020)
21. Li, Z., Wang, X., Li, J., Zhang, Q.: Deep attributed network representation learning of complex coupling and interaction. Knowledge-Based Systems **212**, 106618 (2021)
22. Liu, X., Murata, T.: Community detection in large-scale bipartite networks. Transactions of the Japanese Society for Artificial Intelligence **25**(1), 16–24 (2010)
23. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3431–3440 (2015)
24. Mallea, M.D.G., Meltzer, P., Bentley, P.J.: Capsule neural networks for graph classification using explicit tensorial graph representations. arXiv:1902.08399 (2019)
25. Saigo, H., Nowozin, S., Kadowaki, T., Kudo, T., Tsuda, K.: gboost: a mathematical programming approach to graph classification and regression. Machine Learning **75**(1), 69–89 (2009)
26. Taylor, W.R.: Protein structure comparison using bipartite graph matching and its application to protein structure classification. Molecular & Cellular Proteomics **1**(4), 334–339 (2002)
27. Togninalli, M., Ghisu, E., Llinares-López, F., Rieck, B., Borgwardt, K.: Wasserstein weisfeiler–lehman graph kernels. In: Wallach, H. , Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett R. (eds.) Advances in Neural Information Processing Systems 32 (NeurIPS), pp. 6436–6446. Curran Associates, Inc. (2019)
28. Verma, S., Zhang, Z.L.: Graph capsule convolutional neural networks. arXiv:1805.08090 (2018)
29. Vu, T., Nguyen, T.D., Nguyen, D.Q., Phung, D., et al.: A capsule network-based embedding model for knowledge graph completion and search personalization. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp. 2180–2189 (2019)
30. Wang, H., Lian, D., Liu, W., Wen, D., Chen, C., Wang, X.: Powerful graph of graphs neural network for structured entity analysis. In: World Wide Web pp. 1–21 (2021)
31. Wang, H., Lian, D., Zhang, Y., Qin, L., Lin, X.: Gognn: Graph of graphs neural network for predicting structured entity interactions. arXiv:2005.05537 (2020)
32. Wang, Y., Xiao, W., Tan, Z., Zhao, X.: Caps-owkg: a capsule network model for open-world knowledge graph. Int. J. Mach. Learn. Cybern. **12**(6), 1–11 (2021)
33. Wang, Y.G., Li, M., Ma, Z., Montúfar, G., Zhuang, X., Fan, Y.: Haar graph pooling. In: ICML, pp. 9952–9962 (2020)
34. Wang, Z., Zhou, M., Arnold, C.: Toward heterogeneous information fusion: bipartite graph convolutional networks for in silico drug repurposing. Bioinformatics **36**(Supplement_1), i525–i533 (2020)
35. Wu, J., Pan, S., Zhu, X., Cai, Z.: Boosting for multi-graph classification. IEEE Transactions on Cybernetics **45**(3), 416–429 (2014)
36. Wu, J., Pan, S., Zhu, X., Zhang, C., Philip, S.Y.: Multiple structure-view learning for graph classification. IEEE Transactions on Neural Networks and Learning Systems **29**(7), 3236–3251 (2017)
37. Wu, Z., Song, C., Chen, Y., Li, L.: A review of recommendation system research based on bipartite graph. In: MATEC Web of Conferences, vol. 336, p. 05010 (2021)
38. Xinyi, Z., Chen, L.: Capsule graph neural network. In: International conference on learning representations (2018)

39. Yang, J., Zhao, P., Rong, Y., Yan, C., Li, C., Ma, H., Huang, J.: Hierarchical graph capsule network. arXiv:2012.08734 (2020)
40. Yang, R., Dai, W., Li, C., Zou, J., Xiong, H.: Ncgnn: Node-level capsule graph neural network. arXiv: 2012.03476 (2020)
41. Yin, R., Li, K., Zhang, G., Lu, J.: A deeper graph neural network for recommender systems. Knowledge-Based Systems **185**, 105020 (2019)
42. Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An end-to-end deep learning architecture for graph classification. In: AAAI, pp. 4438–4445 (2018)