Neurocomputing 439 (2021) 348–362

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Graph partitioning and graph neural network based hierarchical graph matching for graph similarity computation

Haoyan Xu [a,b], Ziheng Duan [a,b], Yueyang Wang [a,*], Jie Feng [b], Runjian Chen [b], Qianru Zhang [c], Zhongbin Xu [d]

[a] School of Big Data and Software Engineering, Chongqing University, Chongqing 401331, China
[b] College of Control Science and Engineering, Zhejiang University, Zhejiang 310027, China
[c] College of Foreign Languages, Harbin Institute of Technology, Heilongjiang 150001, China
[d] College of Energy Engineering, Zhejiang University, Zhejiang 310027, China

ARTICLE INFO

Article history:
Received 8 September 2020
Revised 6 January 2021
Accepted 19 January 2021
Available online 26 January 2021
Communicated by Zidong Wang

Keywords:
Graph deep learning
Graph similarity computation
Graph partition
Graph neural network

ABSTRACT

Graph similarity computation aims to predict a similarity score between one pair of graphs to facilitate downstream applications, such as finding the most similar chemical compounds similar to a query compound or Fewshot 3D Action Recognition. Recently, some graph similarity computation models based on neural networks have been proposed, which are either based on graph-level interaction or node-level comparison. However, when the number of nodes in the graph increases, it will inevitably bring about reduced representation ability or high computation cost.

Motivated by this observation, we propose a graph partitioning and graph neural network-based model, called PSimGNN, to effectively resolve this issue. Specifically, each of the input graphs is partitioned into a set of subgraphs to extract the local structural features directly. Next, a novel graph neural network with an attention mechanism is designed to map each subgraph into an embedding vector. Some of these subgraph pairs are automatically selected for node-level comparison to supplement the subgraph-level embedding with fine-grained information. Finally, coarse-grained interaction information among subgraphs and fine-grained comparison information among nodes in different subgraphs are integrated to predict the final similarity score. Experimental results on graph datasets with different graph sizes demonstrate that PSimGNN outperforms state-of-the-art methods in graph similarity computation tasks using approximate Graph Edit Distance (GED) as the graph similarity metric.

1. Introduction

Graph similarity computation, which predicts a similarity score between one pair of graphs, has been widely used in various fields, such as recommendation system [42,16], computer vision [15,35] and so on. However, most of the standard distance measures evaluating how similar two graphs are, like Graph Edit Distance (GED) [7], and Maximum Common Subgraph (MCS) [9], still suffer from large search spaces or excessive memory requirements. They are weak to compute exact graph distance for graphs with more than 16 nodes [5]. Traditional graph similarity computation methods such as *A** [37], *Hungarian* [25,36], *VJ* [12,18], and *Beam*[31], try to use pruning strategy or find approximate values instead of exact similarity to alleviate the problem. Nevertheless, by performing directly from the graphs' edges and node characteristics, these exact and approximate algorithms still have a high time-complexity for computing the GED or MCS between two graphs and are hard to be generalized to large graphs in real applications.

With the rapid development of deep learning technology, graph neural networks that automatically extract the graph's structural characteristics provide a new solution for similarity computation and matching of graph structures. Recently, researchers proposed some representative graph deep learning models for graph similarity computation. During the training stage, these models fit the similarity ground truth (label) in a supervised learning way and learn a mapping between a pair of graph inputs and the similarity score. Hence they are more time-efficient compared with traditional graph similarity computation methods during testing or actual applications [2].

In general, graph deep learning models for graph similarity computation can be categorized into two classes, namely embedding model and matching model (shown in Fig. 1). *Embedding-models* (e.g., GCN-Max, GCN-Mean [11]) directly embed the whole

* Corresponding author.
E-mail address: yueyangw@cqu.edu.cn (Y. Wang).

https://doi.org/10.1016/j.neucom.2021.01.068
0925-2312/

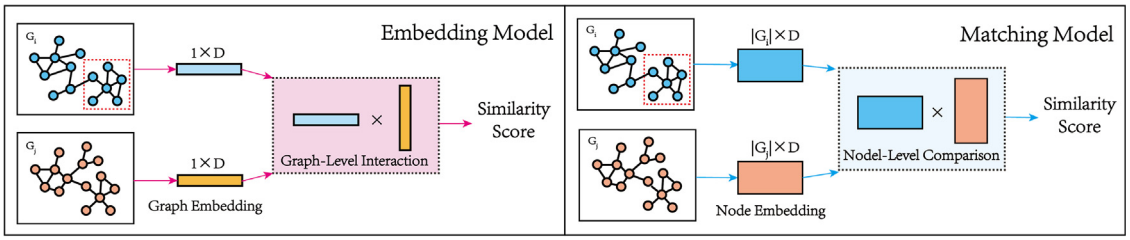


Fig. 1. Two deep learning frameworks for graph similarity computation: embedding-model and matching-model. $|G_i|$ and $|G_j|$ represent the number of nodes of G_i and G_j, respectively, and D represents the embedding dimension. The red dashed box in G_i represents the subgraph of the G_i.

graph to a graph-level vector and compute the similarity between vectors as the similarity of the corresponding graph pairs. These methods are time-efficient but are not effective due to the loss of much node-level comparison information. *Matching-models* (e.g. SimGNN [2], GSimCNN [4], GMN [27]) embed each node into a low-dimension vector which encodes both its own feature information and its local connection relationship information, and contain different pairwise interaction strategies to compute the graph similarity score. However, the pairwise node comparison process in these models needs at least quadratic computation cost with respect to the number of nodes in the graphs so that the problem of not being time-efficient remains on large-scale graph similarity computation.

In this paper, we focus on large graph similarity computation and try to address the following two challenges:

- For large-scale graphs, embedding-models are difficult to learn one vector representing the various features of a graph, so they always lose some prediction accuracy. On the other hand, although the matching-models contain fine-grained interaction and comparison of the graph pair, they can not efficiently calculate the similarity between graphs with many nodes. Thus, this challenge is *how to achieve the trade-off between the accuracy and the efficiency of computing large graph similarity?*
- Comparing with the graphs with fewer nodes, graphs with many nodes always contain distinct local features. For example, protein molecules are composed of many amino acids, determining the different functions of protein molecules. At the same time, amino acids are also composed of several atoms. As shown in Fig. 1, the subgraph in the red dashed box can be compared to an amino acid containing several atoms. When analyzing a protein molecule's function, only focusing on the whole protein molecule (the graph-level embedding) or only considering the atoms (the node-level embedding) may lose these local structural features. Thus, this challenge is *how to learn embeddings capturing the local structure of large graph?*

To solve these challenges, we propose an end-to-end model PSimGNN, i.e., Partition based Similarity Computation via Graph Neural Networks. First, the proposed model partitions each of the input graph into a set of subgraphs. To extract these local structural features, we design a novel graph neural network with an attention mechanism to map every subgraph into a subgraph-level embedding vector. Next, we design an information interaction architecture with two modules to balance the subgraph-level and node-level information. The first module conducts coarse-grained similarity computation by computing the similarity of these subgraph-level embedding vectors. The second module conducts fine-grained similarity computation by automatically selecting some of the subgraph pairs with higher similarity for node-level comparison, thus supplementing the subgraph-level comparison with fine-grained information. Finally, the model integrates coarse-grained interaction information between subgraphs and fine-grained comparison information between nodes in different subgraph pairs to predict the final similarity score. We evaluate the effectiveness and efficiency of our model on large graph similarity computation task. The experimental results show that PSimGNN outperforms the state-of-the-art graph similarity computation models. To summarize, our major contributions are:

- We first propose the graph partitioning based framework to address the challenging problem of similarity computation between large graphs. This framework achieves a good trade-off between accuracy and efficiency.
- We propose a novel model that effectively extracts and aggregates local information to conduct subgraph-level comparison. This can resolve the under-representation ability or high computation cost of many graph deep learning-based similarity computation models.
- We conduct extensive experiments on a prevalent graph similarity/distance metric, GED, based on different size datasets. These experiments and theoretical analysis demonstrate the effectiveness and efficiency of PSimGNN model in graph similarity computation tasks.

The rest of this paper is organized as follows. In Section 2, we discuss the related works. In Section 3, we describe our model PSimGNN for graph similarity computation in detail and analyze the computation cost in theory. In Section 4, we compare the proposed model with some existing graph similarity computation methods on both synthetic and real-world datasets, introduce the experiment settings, and present the experimental results. In Section 5, we offer in-depth discussions and conclusions, and point out future research directions.

2. Related work

In this section, we introduce the related works about graph partitioning, graph neural networks, graph similarity metrics and graph similarity computation.

2.1. Graph partitioning

Graph partitioning is a way of cutting a graph into smaller pieces, while the nodes of these pieces are mutually exclusive with each other. Graph partitioning is an effective way for complexity reduction or parallelization [6] and the partitioned graph may be better suited for analysis and problem-solving than the original [19]. With the advent of ever-larger instances in applications such as scientific simulation, social networks, or road networks, graph partitioning, therefore, becomes more and more critical, multi-faceted, and challenging [6]. Since graph partitioning is a hard problem, a variety of techniques and solutions are proposed. Global algorithms work with the entire graph and compute a solution

directly. These solutions can be improved using several heuristics, and high-quality graph partitioning solvers improve starting solutions. The most successful heuristic for partitioning large graphs is the multilevel graph partitioning approach. It consists of three phases: coarsening, initial partitioning, and uncoarsening [6].

2.2. Graph neural networks

Graph Neural Networks (GNNs) is a useful framework for representation learning of graphs, directly operating on the graph structure. GNNs follow a neighborhood aggregation scheme, where the representation vector of a node is computed by recursively aggregating and transforming representation vectors of its neighboring nodes. Many GNN variants have been proposed and have achieved state-of-the-art results on both node and graph classification tasks [29,20]. Despite GNNs revolutionizing graph representation learning, there is a limited understanding of their representational properties. Studies [43] have shown that popular GNN variants (such as graph convolutional networks and GraphSAGE [13]) have limited discriminative power, and they cannot learn to distinguish certain simple graph structures. In this paper, we use Graph Isomorphism Network (GIN) [43] since GIN has been proven to be theoretically the most powerful GNN under the neighbor aggregation framework [43].

2.3. GED & MCS

Graph Edit Distance (GED) [7] can be considered as an extension of the String Edit Distance [26] metric, which is defined as the minimum cost required to convert one graph to another through a sequence graph editing operations. Maximum Common Subgraph (MCS) [9] is equivalent to GED under the same cost function [8]. Both are the most common ways to calculate the similarity of graphs or the distance between graphs, which is the core operation of graph similarity search and many applications. However, this core operation, computing the GED or MCS between two graphs, is known to be NP-complete [9,45]. For a pair of graphs with more than 16 nodes, even the state-of-the-art algorithms cannot reliably compute the exact GED within reasonable time [5]. So, instead of calculating the exact similarity, some methods can find approximate values in a fast and heuristic way. However, these methods usually require complicated design and the computation cost is still sub-exponential or polynomial in the number of nodes in the graphs, such as *Hungarian* [25,36], *VJ* [12,18], *Beam* [31], etc.

2.4. Graph similarity computation

Computing the similarity of graphs is a basic and essential operation in many applications, including graph classification and clustering [30], social group network similarity identification [39,32], object recognition in computer vision [10], and biological molecular similarity search [24,40], etc. Graph similarity computation for metrics such as Graph Edit Distance (GED) is typically NP-hard, and existing heuristics-based algorithms usually achieve an unsatisfactory trade-off between accuracy and efficiency. Compared with traditional algorithms, which typically involve knowledge and heuristics specific to a metric, the neural network approaches learn graph similarity from data. During training, the parameters are learned by minimizing the loss between the predicted similarity scores and the ground truth; during testing, unseen pairs of graphs can be fed into these models for fast approximation of their similarities [3].

This paper is the first attempt towards large-scale graph similarity computation with deep learning methods. Current deep learning methods for graph similarity computation can be classified as embedding models and matching models. Embedding models such as GCN-Mean and GCN-Max, directly map each graph to a feature vector and compute the similarity score between these feature vectors. Embedding models are efficient, but the performance is usually low due to the lack of interactions across graphs. Matching models, including GMN, SimGNN, and GSimCNN, embed a pair of graphs at the same time with a cross-graph matching mechanism. They are more accurate, but the cross-graph matching process often brings a significant increase in time consumption (at least quadratic computation cost over the number of nodes). Thus, our work explores how to compute the similarity between large-scale graphs while maintaining high accuracy efficiently.

3. The proposed approach: PSimGNN

In this section, we formally define the problem of graph similarity computation, and then introduce the proposed method PSimGNN, i.e., *P*artition based *Sim*ilarity Computation via *G*raph *N*eural *N*etworks, which is an end-to-end neural network-based method to solve graph similarity computation problem. PSimGNN consists of four parts: (1) graph partitioning; (2) subgraph-level embedding interaction; (3) node-level comparison; (4) graph similarity score computation. An overview of PSimGNN is shown in the Fig. 2.

3.1. Problem definition

We define an undirected and unweighted graph $G = \{V, E\}$, where $V = \{v_1, \ldots, v_{|V|}\}$ is a set of nodes and $E = \{e_1, \ldots, e_{|E|}\}$ is a set of edges. $H \in \mathbb{R}^{N \times D}$ represents node features, where N is the number of nodes in graph G (or $N = |V|$) and D is the dimension of node feature vectors. We transform GED into a similarity metric ranging between 0 and 1. Our goal is to learn a neural network-based function that takes two graphs as input and outputs the similarity score that can be transformed back to GED through a one-to-one mapping.

3.2. Graph partitioning

Most neural network-based graph similarity computation models use appropriate mechanisms to generate graph-level embeddings and node-level embeddings and calculate the graph similarity score between different graphs combining a coarsen-grained graph-level interaction and a fine-grained node-level comparison. However, for graphs with a large number of nodes, these approaches may have several limitations:

- Only graph-level embedding may have limited ability to represent the whole graph. Sometimes we have to pay attention to some local structure characteristics.
- Due to a large number of nodes, the node-level comparison will bring high computation cost, and too much matching between nodes far away will also introduce some noise.

To overcome these limitations and better reflect large graphs' local structure characteristics, we partition a graph into k_f subgraphs using the graph partitioning method. In our experiments, the Fluid Communities algorithm (*FluidC*) [34] shows the best performance. Graph partitioning contains three steps:

- Step-1: Choose k_f nodes randomly in the graph as the initial nodes of k_f communities.
- Step-2: Iterate over all nodes in random order and update each node's community based on its community and the communities of its neighbors.
- Step-3: Repeat step-2 until convergence.

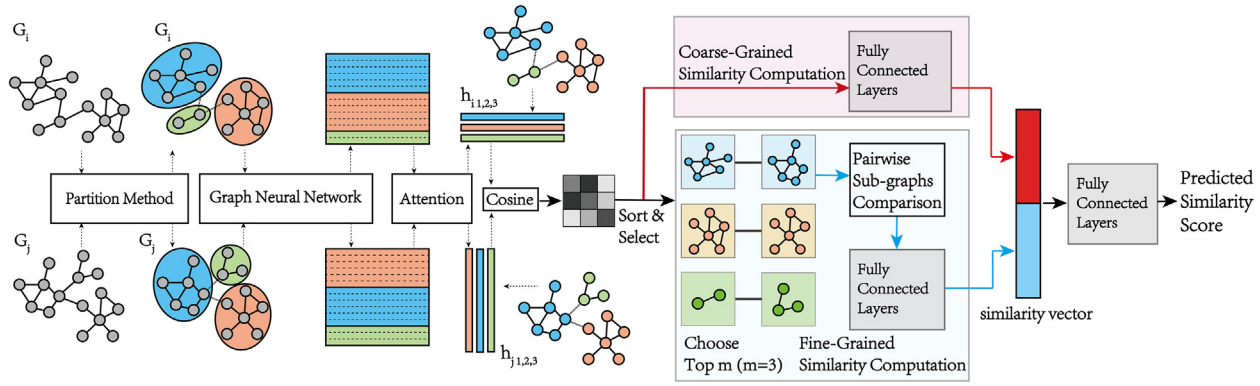


Fig. 2. The general architecture of our model. The red arrows denote the data flow for subgraph-level interaction and the blue arrows denote the data flow for node-level comparison. After the graph partitioning method, only the top m (here $m = 3$) subgraph pairs with the highest similarity scores will conduct the node-level comparison.

Here we focus on its updating strategy, or more precisely, step 2. For a graph $G = (V, E)$ composed of a set of vertices V and a set of edges E, FluidC initializes k_f fluid communities $\mathscr{C} = \{c_1, c_2, \ldots, c_{k_f}\}$, where $0 < k_f \leqslant |V|$. Each community $c \in \mathscr{C}$ is initialized in a different and random vertex $v \in V$. Each initialized community has an associated density d in the range (0,1]. More precisely, the density of a community is the inverse of the number of vertices that make up the community: $d(c) = 1/|v \in c|$. We can notice that a fluid community composed of a single vertex (for example, each community at initialization) has the largest possible density ($d = 1.0$).

The algorithm traverses all V in random order and uses the updating rule to update the community to which each vertex belongs. When the vertex distribution to the community has not changed in two consecutive steps, the algorithm has converged and ended. Next, we focus on its updating rules.

The updating rule for a specific vertex v returns the community or communities with maximum aggregated density within the ego network of v. The updating rule is formally defined in Eqs. (1) and (2).

$$\mathscr{C}'_v = argmax_{c \in \mathscr{C}} \sum_{w \in \{v, \Gamma(v)\}} d(c) \times \delta(c(w), c) \tag{1}$$

$$\delta(c(w), c) = \begin{cases} 1, & if\ c(w) = c \\ 0, & if\ c(w) \neq c \end{cases} \tag{2}$$

where v is the vertex which is going to be updated, $\mathscr{C}'_v$ is the set of candidates to be the new community of v, $\Gamma(v)$ are the neighbours of v, $d(c)$ is the density of community c, $c(w)$ is the community vertex w belongs to and $\delta(c(w), c)$ is the Kronecker delta.

In some exceptional cases, $\mathscr{C}'_v$ can contain multiple community candidates with equal maximum sum. If $\mathscr{C}'_v$ has the current community of vertex v, then v will not change its community. However, if $\mathscr{C}'_v$ does not contain the current community of v, the update rule will select a random community in $\mathscr{C}'_v$ as the new community of v. Here we give a formal representation of the updating rule:

$$c'(v) = \begin{cases} x \sim \mathscr{S}(\mathscr{C}'_v), & if\ c(v) \notin \mathscr{C}'_v \\ c(v), & if\ c(v) \in \mathscr{C}'_v \end{cases} \tag{3}$$

where $c'(v)$ is the community of the vertex v of the next step, $\mathscr{C}'_v$ is the set of candidate communities, and $x \sim \mathscr{S}(\mathscr{C}'_v)$ is the random sampling from the discrete uniformly distribution of $\mathscr{C}'_v$.

The entire process can refer to Fig. 3. At all times, each community has a total density of 1, which is equally distributed among the nodes it contains. If a node changes its community, node densities of affected communities are adjusted immediately. When a complete iteration over all nodes is done, such that no node changes the community it belongs to, the algorithm has converged and returns. Through *FluidC*, we can obtain a series of connected subgraphs (or communities) that can reflect local features. The similarity computation at the subgraph-level and node-level can be performed later.

3.3. Subgraph-level comparison

One useful graph-level embedding can efficiently preserve the structural information, and the similarity between two graphs can be computed by interacting with the two graph-level embeddings. For graphs with many nodes, by comparing the similarity between different subgraphs generated by some graph partitioning methods (like *FluidC* in our experiment), the local similarity between two large graphs can be better reflected. The entire process involves the following three parts: (1) **Subgraph node embedding**, which embeds the nodes of each subgraph into vectors, encoding its structural information; (2) **Subgraph embedding**, which embeds each subgraph into one graph-level vector considering the context information through an attention-based node aggregation way; (3) **Subgraph-subgraph interaction**, which receives two subgraph-level embeddings and returns the interaction score representing the similarity between subgraphs. Next, these subgraph interaction scores are further reduced to a final similarity score through Multilayer Perceptron, representing the similarity of the pair of large graphs. And the parameters involved in these three steps can be updated by comparing the final similarity score with the ground truth similarity score in the training process.

3.3.1. Part I: Subgraph node embedding

Among the existing multiple graph neural network methods, we choose Graph Isomorphism Network (GIN) [43] because it can not only efficiently gather information of neighboring nodes like Graph Convolutional Networks (GCN) [23,11] and GraphSAGE [13], but also learn accurate structural information. The hidden layer can be written as follow:

$$h_v^{(k)} = MLP^{(k)}\left((1 + \epsilon^{(k)}) \cdot h_v^{(k-1)} + \sum_{u \in \mathscr{N}(v)} h_u^{(k-1)}\right), \tag{4}$$

where $h_v^{(k)}$ is the k-th layer node embedding for the node v, MLP means Multilayer Perceptron [33], ϵ is a learnable parameter and $\mathscr{N}(v)$ represents the neighbor nodes of node v.

We treat each node as the same label for graphs with unlabeled nodes, thereby obtaining the same constant as the initial representation. After multiple GIN layers (3 layers in our experiment), the node-level embeddings information will be fed into the attention module as described below.

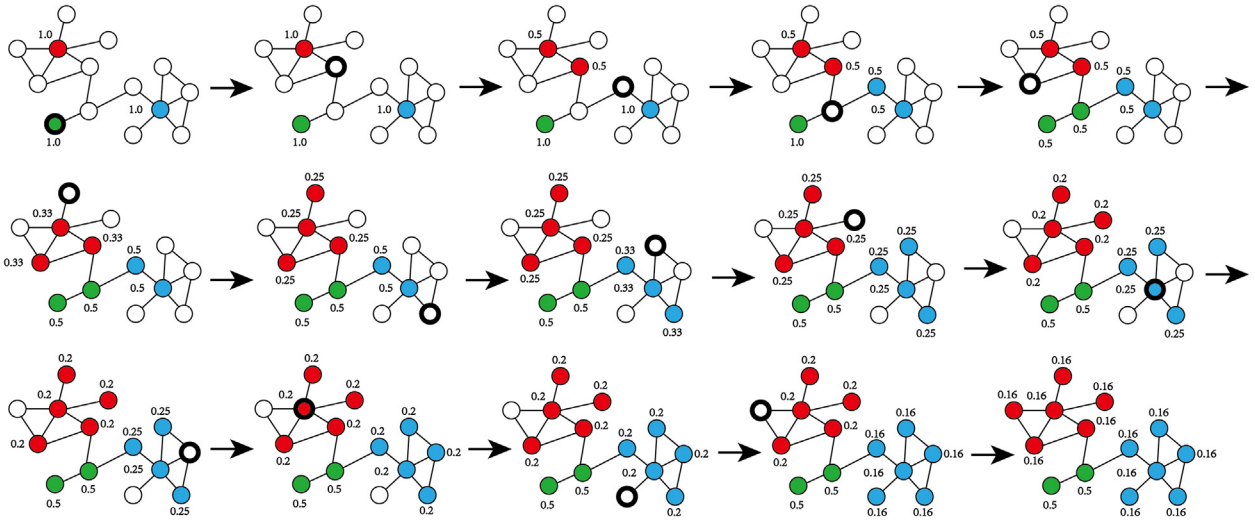


Fig. 3. The workflow of *FluidC* for $k_f = 3$ communities (red, green and blue). Each node assigned to a community is labeled with the density of that community. The update rule is evaluated on each step for the node highlighted in black. Only the schematic results of the first iteration are given here.

3.3.2. Part II: Subgraph embedding

This model uses a weighted sum method, where we use an attention mechanism to generate subgraph-level embeddings with a weighted sum method. Instead of averaging all nodes or giving each node different weights according to the node's degree, our attention module focuses more on the nodes that can better represent the full graph structure information.

After learning the node-level embedding, the node embeddings in subgraph can be expressed as $X \in \mathbb{R}^{N_s \times D}$, where N_s represents the number of nodes in the subgraph, and D is the dimension of each node embedding. The representation of the whole subgraph information can be written as $z \in \mathbb{R}^D$, which is a non-linear expression of the average value of N nodes embedding: $z = \tanh((\frac{1}{N}\sum_{i=1}^{N} u_i)W_z)$, where W_z is a learnable weight matrix.

By learning the weight matrix, z provides the subgraph's global structure and feature information suitable for a given similarity measure. Then based on z, we can calculate an attention weight for each node. In short, attention works as a memory-access mechanism by generating larger attentive coefficients for input features that are relevant to the learning task. In this paper, we adopt an attention mechanism to generate subgraph-level representation. For node i, to notify the global information, we take the inner product between its node embedding x_i and z. That is to say, the nodes that are more capable of expressing the features of the graph should be given higher weights. The sigmoid function $\sigma(x) = 1 + exp^{-x}$ is applied to the result to ensure that the attention weight is between (0, 1). Finally, subgraph embedding $h \in \mathbb{R}^D$ is the weighted sum of node embedding:

$$h = \sum_{j=1}^{N} \sigma(x_j \odot z) x_j = \sum_{j=1}^{N} \sigma\left(x_j \odot \tanh\left(\left(\frac{1}{N}\sum_{i=1}^{N} u_i\right) W_z\right)\right) x_j, \tag{5}$$

where $\odot$ represents the dot product between vectors.

3.3.3. Part III: Subgraph-subgraph interaction

Through the node embedding and attention mechanism mentioned above, we have achieved subgraph-level embedding. Good node embedding and attention mechanisms should embed graphs with similar structures and similar features in similar positions in space, so their distance should be relatively small. Here we use the *cosine* similarity to measure the similarity between a pair of subgraph embeddings:

$$s(h_1, h_2) = cos(h_1, h_2) = \frac{h_1 \odot h_2}{||h_1||_2 ||h_2||_2}, \tag{6}$$

where $||h||_2$ is the 2-norm of h.

A pair of large graphs are partitioned into k subgraphs respectively, and the similarity between two subgraphs between large graphs is calculated using the method mentioned above. After that, k^2 similarity scores are obtained, and Multilayer Perceptron (*MLP*) is used to map these k^2 scores to the final similarity score to characterize the similarity between the pair of large graphs:

$$s(G_1, G_2) = MLP\left(\bigoplus_{i=1,j=1}^{k} s(G_1^i, G_2^j)\right), \tag{7}$$

where $\oplus$ is the concatenation operation, $s(G_1, G_2)$ represents the similarity score between the pair of large graphs and $s(G_1^i, G_2^j)$ represents the similarity score between the i-th subgraph of G_1 and the j-th subgraph of G_2.

3.4. Node-level comparison

Only considering subgraph-level embedding interaction may lose some fine-grained node-level information, so we design the following node-level comparison component utilizing the interaction between nodes in subgraphs.

This component accepts a pair of subgraphs as its input, and calculates the similarity between them through the comparison of nodes within each subgraph and between the pair of subgraphs. An overview of the interaction is shown in the Fig. 4, where $h^{(t)}$ represents the node embeddings in each subgraph after k-th propagation layer. We assume that the input subgraph pair can be represented as G_1^m, G_2^n, and their node sets and edge sets are V_1^m, V_2^n and E_1^m, E_2^n, respectively. After t iterations within the graph and between graphs, the embedding of node i can be represented as $h_i^{(t)}$. In each interaction within the subgraph, the influence of node j on node i is:

$$v_{j \to i} = MLP(h_i^{(t)} \oplus h_j^{(t)}), \forall (i,j) \in E_1^m \cup E_2^n \tag{8}$$

Then is the interaction between subgraphs for cross-graph communication. An attention mechanism is used to give different weights to the nodes of another subgraph to indicate the importance of different nodes j to nodes i:

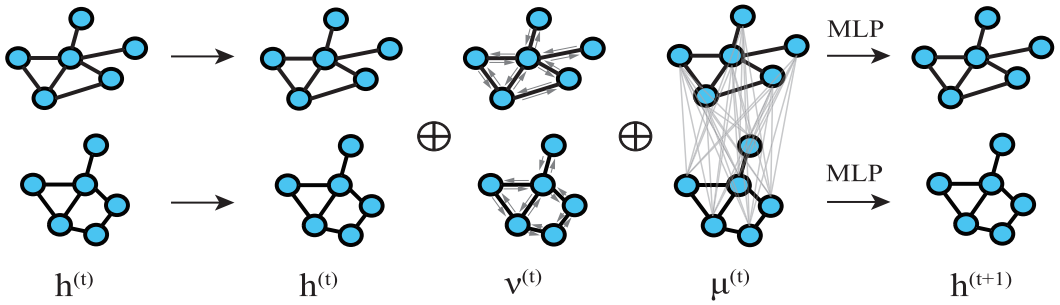


Fig. 4. Propagation layer in node-level comparison. Each round of iteration is based on the embeddings of the previous round and the node-level comparison within each graph and between graphs.

$$a_{j\to i} = \frac{\exp(\mathrm{h}_i^{(t)} \odot \mathrm{h}_j^{(t)})}{\sum_{j'} \exp(\mathrm{h}_i^{(t)} \odot \mathrm{h}_{j'}^{(t)})} \tag{9}$$

Through this attention mechanism, we magnify the influence between similar nodes in one pair of subgraphs, and use $\mu_{j\to i}$ to represent the interaction between node j and node i in different subgraphs:

$$\mu_{j\to i} = a_{j\to i}(h_i^{(t)} - h_j^{(t)}), \forall i \in V_1, j \in V_2, \text{ or } \forall i \in V_2, j \in V_1 \tag{10}$$

After obtaining the interactive information within each subgraph and between one pair of subgraphs, we merge the t-th round propagation node information with it, and then generate the $(t+1)$-th round propagation node information:

$$h_i^{(t+1)} = MLP\left(h_i^{(t)} \oplus \sum_j v_{j\to i} \oplus \sum_{j'} \mu_{j'\to i}\right) \tag{11}$$

After iterating through T rounds, we get the embedding of each node, denoted as $h^{(T)}$, and then through a self-attention mechanism aggregation layer, we get a subgraph-level embedding:

$$h_{agg} = MLP_{agg}\left(\sum_{i\in V} \sigma(MLP_{att}(h_i^{(T)})) \odot MLP(h_i^{(T)})\right) \tag{12}$$

After obtaining the fine-grained embedding of each subgraph, we use the *cosine* similarity to measure the similarity between one pair of graphs, which is expressed as:

$$s(h_{agg1}, h_{agg2}) = cos(h_{agg1}, h_{agg2}) = \frac{h_{agg1} \odot h_{agg2}}{||h_{agg1}||_2 ||h_{agg2}||_2} \tag{13}$$

3.5. Graph similarity score computation

It is worth mentioning that through the previous graph partitioning, each large graph is partitioned into k subgraphs, and there will be k^2 pairs of subgraphs. Here, we sort the subgraph-level similarities obtained before, and only the pairs with top m similarity score will perform a node-level comparison. We use (MLP) to integrate k^2 coarse-grained scores and m fine-grained scores to finally get the similarity between the large graphs:

$$s(G_1, G_2)_{coarse} = MLP\left(\bigoplus_{i=1,j=1}^{k} s(G_1^i, G_2^j)\right) \tag{14a}$$

$$s(G_1, G_2)_{fine} = MLP\left(\bigoplus_{t=1}^{m} s(G_1^{it}, G_2^{jt})\right) \tag{14b}$$

$$s(G_1, G_2) = MLP\left(s(G_1, G_2)_{coarse} \oplus s(G_1, G_2)_{fine}\right) \tag{14c}$$

After the similarity score, $s(G_1, G_2) \in \mathbb{R}$, is predicted, it is compared with the ground truth similarity score, $s(G_1, G_2)_{gt} \in \mathbb{R}$, using the following mean square error loss function:

$$\mathscr{L} = \frac{1}{|\mathscr{T}|} \sum_{(i,j)\in\mathscr{T}} \left(s(G_1, G_2) - s(G_1, G_2)_{gt}\right)^2 \tag{15}$$

,where $\mathscr{T}$ is the training graph pairs and $|\mathscr{T}|$ is the total number of the training graph pairs.

3.6. Efficiency analysis

For a pair of input graphs G_1 and G_2 with E_1, E_2 edges and N_1 and N_2 nodes separately, we can evaluate the efficiency of several types of models that are commonly used in graph similarity computation.

Then we analyze the efficiency of PSimGNN and discuss how it can improve the efficiency by graph partitioning. Note that there exists a lot of variance for each model. We only use the simplest cases.

3.6.1. Embedding models

The embedding model refers to calculating the similarity between graphs by generating graph-level embeddings. Assuming the simplest case here, we only visit every edge once and deploy two computational operations on the two nodes it connects, contributing to the feature of local topology. Thus the computation cost for these cases is $O(max(E_1, E_2))$.

3.6.2. Matching models

The matching model refers to calculating the similarity between graphs by matching (graph-level interaction or node-level comparison). Assuming the simplest case here, we compute the relationship across N_1 and N_2. This part involves $N_1 \times N_2$ computational operations because we have to calculate the connection between every node in G_1 to all nodes in G_2. For the common matching models, both SimGNN and GSimCNN pad fake nodes to the smaller graph at the node-level comparison to emphasize their size difference. GMN also has the interaction of nodes within each graph, so the final computation cost is $O(max(N_1, N_2)^2)$.

3.6.3. PSimGNN

The computation cost of PSimGNN can be divided into three parts to analyze. (1) Graph Partitioning. In our model, we choose *FluidC* as the graph partitioning method. As analyzed in Section 3.1, it updates node information based on neighbor nodes or the connected edges of nodes, so it belongs to the fastest and most scalable family of algorithms in the literature with a linear computation cost of $O(E)$ [34]. Notice that the partitioned subgraphs can be

pre-computed and stored. In the setting of graph similarity search, the unseen query graph only needs to be partitioned once to obtain its subgraphs. (2) Subgraph-level Embedding Interaction. The computation cost associated with the generation of node-level and subgraph-level embeddings is $O(E)$ [23]. Assuming that each graph is partitioned into k subgraphs and the embedding dimension at the subgraph-level is D, we use *cosine* to measure the similarity between embeddings. The computation cost in the subgraph interaction part is $O(Dk^2)$. As mentioned above, the sub-graph level and node-level embedding can also be saved in advance, which significantly saves graph similarity query cost. (3) Node-level Comparison. According to the k^2 similarity scores obtained by subgraph-level interaction, we select top m subgraph pairs with the highest similarity scores for node-level comparison. After partitioning, the average number of nodes in each subgraph is N_1/k or N_2/k. As analyzed in Section 3.5.2, the average node-level comparison computation cost of one pair of subgraphs is $O(max(N_1/k, N_2/k)^2)$. Since we choose m pairs, the total computation cost of this part is $O(m \times max(N_1/k, N_2/k)^2)$ or $O(m/k^2 \times max(N_1, N_2)^2)$, where the range of m belongs to $\{0, 1, 2, \ldots, k^2\}$. The parameter m can be used as a hyperparameter to adjust the relationship between accuracy and time. When m is set to zero, our model only calculates the coarse-grained subgraph similarity. At this time, the model's computation cost is $O(E)$, where E is the number of edges in the large graph. When m is set to k^2, our model performs fine-grained similarity calculation for each pair of subgraphs, and the computation cost of the model is $O(N^2)$, where N is the number of nodes in the large graph. For occasions with time requirements, we can only perform coarse-grained matching between subgraphs. For occasions where accuracy requirements are relatively high, we can perform a fine-grained node-level comparison to improve model performance. Therefore, according to specific application scenarios, trade-offs between time and accuracy can be made to choose the best solution.

3.6.4. Similarity search

In the similarity search problem, we assume that we have a database consisted of K graphs, each of which has N nodes and E edges, for simplicity. We need to finish computing the similarity between all the graphs in the database and an incoming new graph (also with N nodes and E edges). In embedding models, we can compute all the feature vectors for graphs in the database at the very beginning. And then, when the new graph comes, we encode it to its feature vector and only compute similarity based on the feature vectors. Thus the computation cost is $O(E \times K)$. We can only forward pairs of graphs in matching models every time because of the computation across graphs. Thus the computation cost is extremely high $O(N^2 \times K)$. And obviously, the computation cost for our framework is $O(m/k^2 \times N^2 \times K + E \times K)$. When m is small, the computation cost becomes $O(E \times K)$; when m is large, the computation cost becomes $O(m/k^2 \times N^2 \times K)$. This also reflects the adjustability of our model. It is worth mentioning that our model is not suitable for very dense graphs because it is challenging to get subgraphs that can better reflect local information. In our discussion, $E \ll N^2$.

4. Experiments

4.1. Datasets

In this section, we first introduce a graph similarity computation dataset based on Barab ási-Albert preferential attachment model (BA-model) [17], which consists of three sub-datasets: BA-60, BA-100, BA-200, named according to the average number of nodes per graph. A real-world dataset is also introduced to better demonstrate our framework's effectiveness: The Internet Movie Database (IMDB).[1] The details about these datasets are as following, and we compare these with other well-known datasets used for graph similarity computation.

4.1.1. Barabási-Albert model dataset

Here we introduce the Barabási-Albert model (BA-model) concept, the rules for generating a Barabási-Albert graph (BA-graph), and how our datasets are produced. The BA-model [17] is an algorithm for generating random scale-free networks using a preferential attachment mechanism. Several natural and human-made systems, including the Internet, the world wide web, citation networks, and some social networks, are thought to be approximately scale-free and contain few nodes (called hubs) with unusually high degrees compared to the other nodes of the network. The BA-model tries to explain such nodes in real networks and incorporates two important general concepts: growth and preferential attachment, which exist widely in real networks. Growth means that the number of nodes in the network increases over time and preferential attachment indicates that the more connected a node is, the more likely it is to receive new links. Nodes with a higher degree have a more vital ability to grab links added to the network.

The BA-model begins with an initial connected network of m_0 nodes. New nodes are added to the network one at a time. Each new node is connected to $m \leqslant m_0$ existing nodes with a probability proportional to the number of links that the existing nodes already have. Formally, the probability p_i that the new node is connected to node i is $p_i = \frac{k_i}{\sum_j k_j}$ [1], where k_i is the degree of node i and the sum is made overall pre-existing nodes j (i.e. the denominator results in twice the current number of edges in the network). Heavily linked nodes (“hubs”) tend to quickly accumulate even more links, while nodes with only a few links are unlikely to be chosen as the destination for a new link. The new nodes have a “preference” to attach themselves to the already heavily linked nodes.

Our datasets are made up of some basic graphs and derivative graphs that have been trimmed, which solve several problems:

- When generating a graph with a large number of nodes randomly, there is a high probability that the generated graphs are dissimilar between each other, which results in an uneven similarity distribution after normalization.
- Due to a large number of nodes in each graph, the approximate GED algorithm cannot guarantee that the calculated similarity can fully reflect the graph pairs' similarity. We trim and generate derivative graphs while recording the number of trimming steps. These steps and the values calculated by the approximation algorithm take the minimum value as the GED with the basic graph, thereby obtaining a more accurate similarity.
- By trimming different steps, we can generate graphs with different similarities, which is more conducive to the experiment of graph similarity query.

There are three trimming methods: delete a leaf node, add a node, and add an edge. Since deleting an edge may have a greater impact on the generated graph, we will not consider this method. We try to trim the base graph without changing the base graph's global features to generate more similar graph pairs. In this way, we get three datasets according to the following generation rule.

A BA-graph of n nodes is grown by attaching new nodes, each with m edges that are preferentially attached to existing nodes with a high degree. We set n to be 60, 100, and 200, respectively, and m is fixed to 1 to generate basic graphs. Each sub-dataset gen-

[1] http://www.imdb.com/interfaces#plain

erates two basic graphs, and each base graph is trimmed with different GEDs. For each basic graph, generate 99 trimmed graphs in the range of GED 1 to 10. So each sub-dataset consists of two basic graphs and 198 trimmed graphs.

4.1.2. The internet movie database

The Internet Movie Database (IMDB) consists of the entities of movies, actors, producers, and their relationships. We filter the original IMDB dataset based on two principles: (1) graphs that have 15 or more nodes, (2) graphs where the ratio of the number of edges to the number of nodes is less than 5. The graphs filtered in this way have a sufficient number of nodes, and they are not too dense, which are suitable for partitioning in the task of graph similarity computation. In this paper, we call the new dataset IMDB-X.

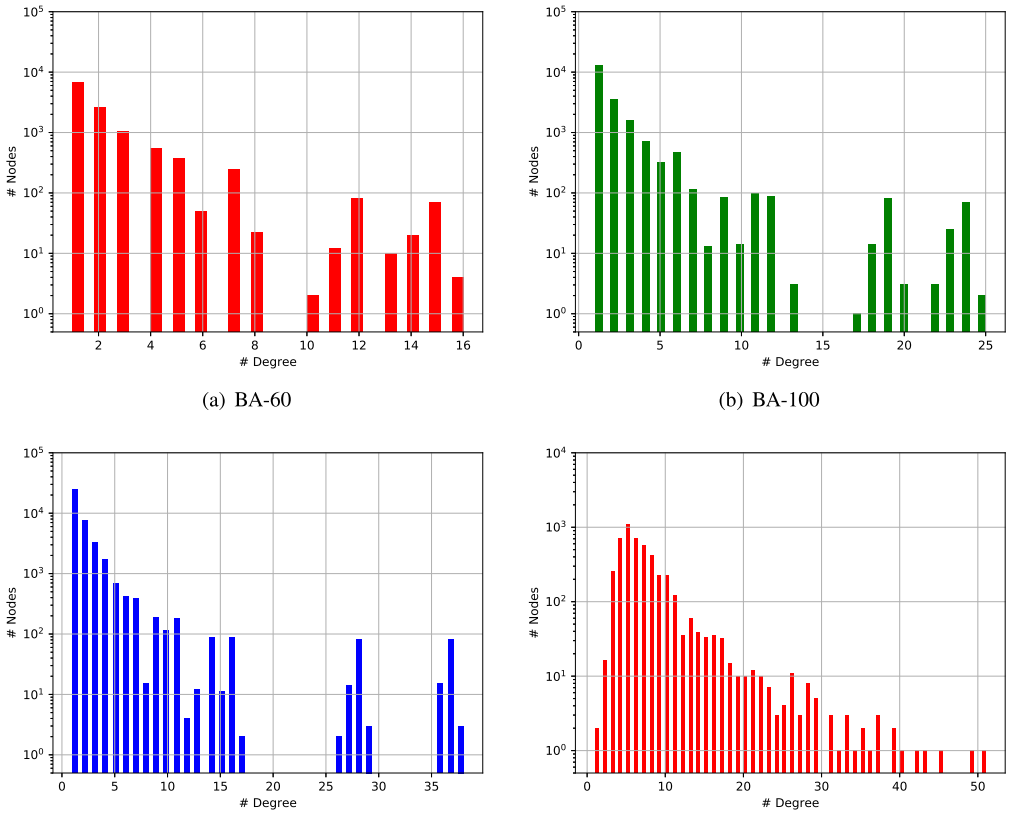
4.1.3. Comparison with other datasets

Because in other public datasets used in [2–4], such as AIDS [28] and LINUX [41], the number of nodes in each graph is relatively small and local structures are not obvious, the characteristics of the entire graph can be easily represented. As for IMDB [44], (named "IMDB-MULTI") there are some graphs with a large number of nodes. However, these graphs are relatively dense, and too many edges between nodes will make the local structures less obvious. So we filter the IMDB as IMDB-X and focus on the BA datasets and IMDB-X.

In view of this, we artificially made three BA-datasets, which have a large number of nodes and have graphs with obvious local structures by using the BA-model characteristics. Table 1 shows the comparison of different datasets for graph similarity computation. Fig. 5 shows the nodes degree distribution of BA-model datasets and IMDB-X. From these charts, we can see that the average number of nodes to the number of edges in the BA-model datasets is approximately equal to 1. Graphs are relatively sparse and suitable for extracting local structural features by graph partitioning. The degree distribution indicates that most nodes have relatively low degrees, and only a few have high degrees. These nodes have

Table 1
Statistics of datasets.

Dataset	Graph Meaning	#Graphs	#Pairs	Min #Nodes	Max #Nodes	Avg #Nodes	Min #Edges	Max #Edges	Avg #Edges
AIDS	Chemical Compounds	700	490 K	2	10	8.90	1	14	8.80
LINUX	Program Dependency Graphs	1000	1 M	4	10	7.58	3	13	6.94
IMDB	Actor/Actress Ego-Networks	1500	2.25 M	7	89	13.00	12	1467	65.95
BA-60	Barabási-Albert graph with 60 nodes	200	40 K	54	65	59.50	54	66	60.06
BA-100	Barabási-Albert graph with 100 nodes	200	40 K	96	105	100.01	96	107	100.56
BA-200	Barabási-Albert graph with 200 nodes	200	40 K	192	205	199.63	193	206	200.16
IMDB-X	Filtered Actor/Actress Ego-Networks	220	48.4 K	15	52	21.35	33	186	74.20



(a) BA-60 (b) BA-100

(c) BA-200 (d) IMDB-X

Fig. 5. Nodes degree distribution of BA-model datasets.

a greater probability of becoming the center node of the subgraph. The partitioning results of the two graphs in the BA-60 dataset and two graphs in IMDB-X are shown in Fig. 6. Through graph partitioning, obvious local structural features can be extracted, which is also a characteristic of our BA-model datasets and IMDB-X.

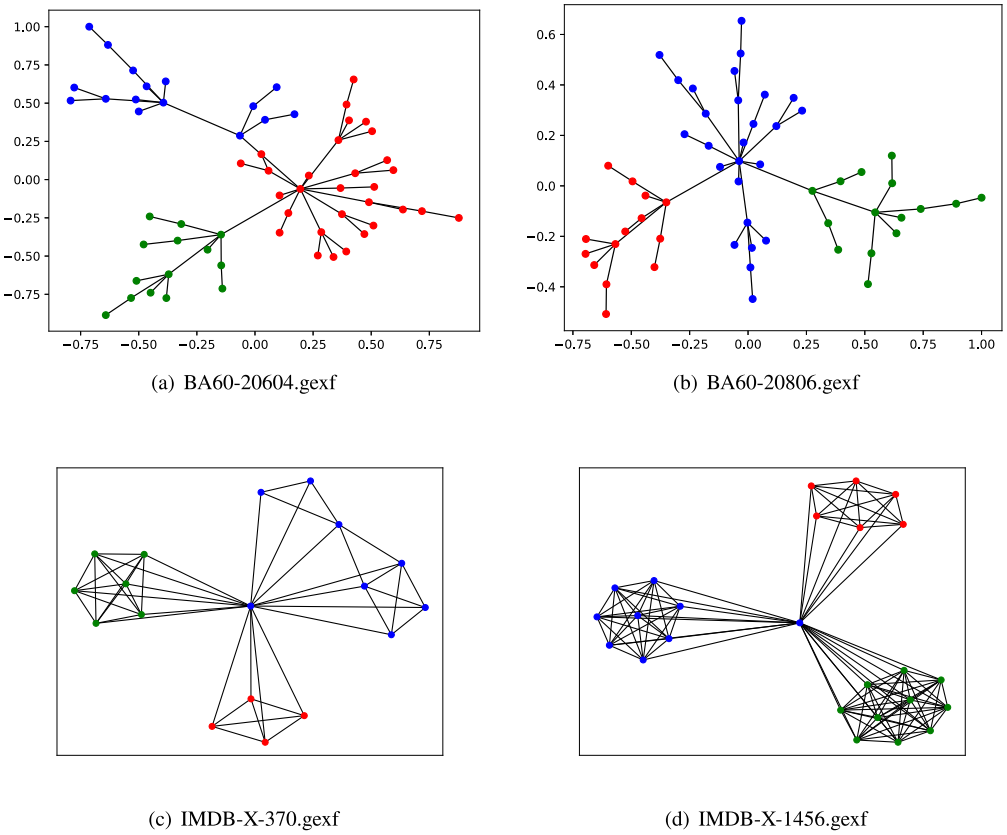
4.2. Data preprocessing and ground-truth generation

For each dataset, we randomly split 60%, 20%, and 20% of all graphs as the training set, validation set, and test set, respectively.

Due to the large number of nodes in our data set, A^* [37] algorithm cannot be used to calculate the GED. We used the smallest distance calculated by three well-known approximation algorithms, *Hungarian* [25,36] and *VJ* [12,18], and *Beam* [31]. However, these algorithms are also difficult to ensure a certain accuracy in this case. So we also added the GED value generated when trimming the graph as another evaluation indicator. When each graph is trimmed, we will get a GED value to record the number of trimming steps. Every time a leaf node is deleted in this experiment, the edge it connects will also be deleted. In this case, the GED between the derived graph and the basic graph increases by 2; and every time an edge is added, GED increases by 1.

As shown in Fig. 7, when generating a derivative graph with a specific GED from the basic graph, we randomly select among the above three methods (randomly delete a leaf node, add a leaf node or add an edge) to generate a set of operations, and the sum of GED accumulated by all operations is the specific GED value. We take the minimum value of the trimming GED and the calculated GED with these three algorithms as the final GED value. Here, the minimum value is taken instead of the average value because GED is the upper bound. The real GED value must be less than or equal to the GED PSimGNN-up only uses the subgraph-level embeddings and achieves the same level of evaluation results as other matching models, which proves the effectiveness of introducing subgraphs to help the large graphs similarity computation. PSimGNN-k, which uses k subgraph pairs for node-level comparison, achieves better results than PSimGNN-up on all evaluation metrics. Our model, PSimGNN, consistently achieves the best or second-best under most evaluation metrics across the three datasets within the neural network-based methods. In some ranking indicators (ρ, τ, and $p@10$) of BA-100 and BA-200, although PSimGNN is not optimal, which may be caused by the randomness of graph partitioning, it still gets close to GSimCNN and GMN in performance. This implies that our model introduces a more flexible framework and performs the same level of accuracy as other neural network-based models. And as the number of subgraph pairs using node-level comparison increases, the model contains more information. The corresponding evaluation results become better, which is also in line with our expectations. As mentioned before, for graphs in IMDB-X, there is no trimming steps. The Beam algorithm's accuracy is higher than the other two, so most of the ground truth is the approximate GEDs calculated by Beam. In this case, Beam shows the best performance in the Table 6, which also meets our expectations. At the same time, *PSimGNN* also shows the best performance compared with other deep learning methods on the IMDB dataset, which proves that the idea of graph partitioning is effective in the task of graph similarity computation.

As shown in Table 7, we recorded the running time for different models on test datasets. It is worth noting here that these models are implemented in different ways, and the framework itself may cause the time issue. In this case, it isn't easy to judge the computation cost directly by the running time is consumed. For example, GCN-Mean and GCN-Max, although their computation cost is $O(N)$, to get the best performance, there are several layers of node



(a) BA60-20604.gexf (b) BA60-20806.gexf

(c) IMDB-X-370.gexf (d) IMDB-X-1456.gexf

Fig. 6. Examples of graph partition from BA-60 and IMDB-X datasets. Different colors represent different subgraphs.

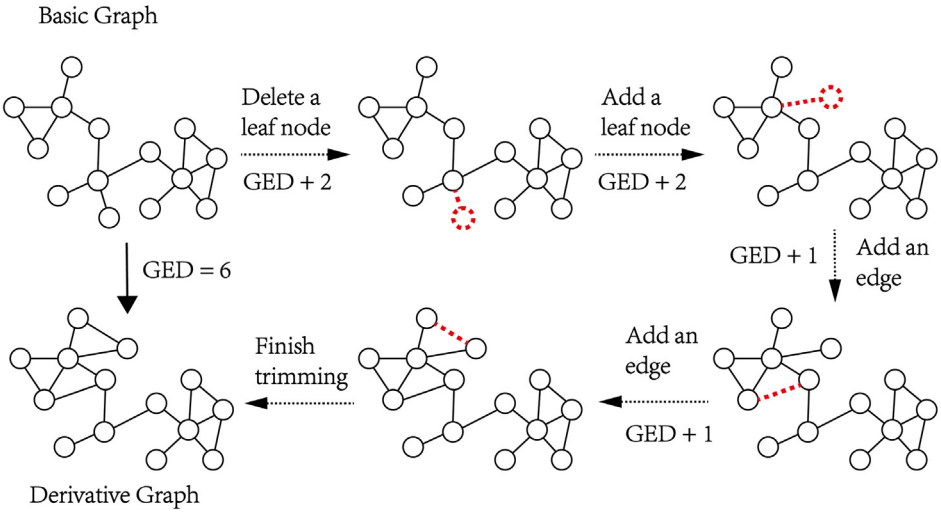


Fig. 7. Generate a derivative graph with a GED of 6 from the basic graph. It is not necessary to use all three methods in real trimming. Here is only one case.

embedding, so the time consumed is not the lowest. And GSimCNN, with $O(N^2)$ computation cost, has the fastest running time due to the optimization of CNN in the framework used.

However, since we used the GMN code for reference when implementing node-level interaction in PSimGNN, the two still have a comparative value. By observing the running time of these two methods on three datasets of different scales, we can find that when the number of graph nodes is small, PSimGNN, which partitions first and then calculates the similarity, takes a long time due to a large number of steps. When the number of nodes is large, GMN directly interacts with the large graph at the node level with quadratic computation cost, which takes more time. Simultaneously, PSimGNN performs node interaction with the smaller graph, thus shortening the time, which proves our previous analysis of computation cost. At the same time, the experimental results of PSimGNN-up, PSimGNN-k, and PSimGNN also indicate that the more subgraphs involved in the computation, the longer it takes. d here. There are no trimming steps for graphs in IMDB-X, so we only these three well-known approximation algorithms to get the ground truth GED.

In order to convert the calculated GED into the similarity score required by our model, we first normalize the GED by $nGED(G_1, G_2) = \frac{GED(G_1,G_2)}{(|G_1|+|G_2|)/2}$, where $|G_i|$ represents the total number of nodes in graph G_i. Then use the exponential function $f(x) = e^{-x}$ to map the normalized GED to between 0 and 1 to represent the pair's graph similarity. Here we can see that the more similar the graph, the smaller the GED, and the more similarity tends to 1.

4.3. Baseline methods

Our baseline includes three categories of methods, fast approximate GED calculation algorithms, graph embedding based models, and graph matching network-based models.

- The first category of baseline includes three classic algorithms for GED calculation. (1) *Hungarian* [25,36] is a cubic-time algorithms based on the Hungarian Algorithm for bipartite graph matching. (2) *VJ* [12,18] is also a cubic-time algorithms based on the algorithm of Volgenant and Jonker. (3) *Beam search* (*Beam*) [31]. The equivalent variable of the A^* algorithm is sub-exponential time.
- The second category of baseline includes two graph embedding based models, GCN-Mean and GCN-Max [11]. They all embed graphs into vectors using GCN and then use the similarities calculated by these vectors as the similarities of these graph pairs.
- The third category of baseline includes three graph matching network-based models. (1) SimGNN [2] and (2) GSimCNN [4] combine the embedding of the whole graph and node-level comparison. (3) GMN [27] uses the comparison node information within and between graphs to calculate similarity.

Our method also belongs to the third category of methods, using graph matching based networks to calculate the similarities of graph pairs.

4.4. Parameter settings

For the architecture of our model, PSimGNN, we partition each large graph into k (here $k = 3$) subgraphs. Among the nine subgraph pairs, 0, 3, and 9 subgraph pairs with the highest similarity scores are selected for node-level comparison, respectively. Here we call them PSimGNN-up (only subgraph-level interactions are involved in the computation), PSimGNN-k (k or three pairs of subgraphs participate in the node-level comparison), and PSimGNN (all or nine pairs of subgraphs participate in the node-level comparison). It is worth mentioning that we do not perform graph partition for graphs with very few nodes, such as the graphs in the AIDS and LINUX datasets.

We set the number of GIN [43] layer to 3, and use Parametric Rectified Linear Unit (PReLU) [14] as the activation function. For the initial node representations, we adopt the constant encoding scheme for BA-datasets since their nodes are unlabeled, as mentioned in Section 3.2.1. The dimensions of the 1st, 2nd, and 3rd layer of GIN's output are 64, 32, and 16, respectively. We use a fully connected layer to reduce the similarity vectors' dimension obtained at the subgraph-level interaction from 9 to 8, and another fully connected layer to change the dimension of the similarity vector after the node-level comparison from 3 to 8. Finally, four fully connected layers are used to reduce the dimension of the concatenated results from the subgraph-level interaction and the node-level comparison module, from 16 to 8, 8 to 4, 4 to 2, and 2 to 1.

For training, we set the batch size to 128, use the Adam algorithm [22] for optimization, and set the initial learning rate to 0.001. We set the number of training iterations to 2000 and choose the best model based on the lowest validation loss.

4.5. Evaluation metrics

We used two metrics to evaluate the similarity computation results of this model. *Mean Squared Error (MSE)*. MSE measures the average squared difference between all the calculated similarities and the ground-truth similarities. *Mean Absolute Error (MAE)*. MAE measure the averaged value of the absolute deviation of all the calculated similarities from the ground-truth similarities.

For the ranking results, we also use *Spearman's Rank Correlation Coefficient* (ρ) [38] and *Kendall's Rank Correlation Coefficient* (τ) [21] to evaluate how well the predicted ranking results match the true ranking results. *Precision at k* ($p@k$) is computed by taking the intersection of the predicted top k results and the ground truth top k results divided by k. Compared with $p@k$, ρ and τ can better reflect the global ranking results instead of focusing on the top k results.

4.6. Results and analysis

The experimental results on these datasets can be found in Tables 2–6. Table 2 shows that on small graphs where the local features are not obvious enough, PSimGNN can show comparable performance to other matching models. This proves that the framework is suitable for small graphs, with great scalability. The ranking results of *VJ* on the BA-60 dataset is extremely poor, and these three traditional methods also have very high *MSE* and *MAE*. These results show the limitations of traditional methods for graphs with a large number of nodes. However, as far as the index τ is concerned, the optimal value of *beam* continues to exceed the neural network methods. This may be due to the way that *beam* directly acts on edges and nodes in the BA graph can better distinguish the distance between query graphs and graphs in the database, thus having advantages in the ranking. As for the BA-100 dataset, $p@20$ is 100%. This is because when randomly dividing the test dataset of 40 graphs, there are exactly 20 graphs from the basic graph 1, and the other 20 graphs are from the basic graph 2. Being able to distinguish these graphs correctly also proves the excellent performance of the neural network-based models.

For all datasets, the GCN-Mean and GCN-Max results are worse than any matching model in terms of most evaluation indicators. When the number of nodes per graph increases, the limitation of using one vector to characterize the entire graph is more obvious, and the results are worse, which also confirms our previous analysis in Section 3.2.

PSimGNN-up only uses the subgraph-level embeddings and achieves the same level of evaluation results as other matching models, which proves the effectiveness of introducing subgraphs to help the large graphs similarity computation. PSimGNN-k, which uses k subgraph pairs for node-level comparison, achieves better results than PSimGNN-up on all evaluation metrics. Our model, PSimGNN, consistently achieves the best or second-best under most evaluation metrics across the three datasets within the neural network-based methods. In some ranking indicators

Table 3
Results on BA-60 dataset ($\mathbf{10^{-2}}$).. The best results of the neural network-based models, as well as the traditional methods that exceed these results are bolded.

Method	MSE	MAE	ρ	τ	**p@10**	**p@20**
hungarian	18.62	33.22	75.98	57.72	74.25	84.75
vj	25.87	39.48	3.29	2.29	35.00	50.50
beam	5.88	12.93	**85.80**	**74.34**	67.75	90.00
GCN-Mean	0.58	5.39	75.64	53.29	58.00	86.88
GCN-Max	1.37	9.14	74.61	52.30	54.50	86.62
SimGNN	0.78	6.58	77.30	56.78	71.00	88.87
GSimCNN	0.60	5.61	80.78	60.47	67.75	90.50
GMN	0.27	3.82	76.36	54.67	60.00	89.00
PSimGNN-up	0.44	4.80	78.92	57.63	59.50	88.37
PSimGNN-k	0.32	4.07	80.43	60.31	70.50	88.00
PSimGNN	**0.20**	**3.39**	**84.49**	**66.15**	**78.50**	**91.87**

Table 4
Results on BA-100 dataset ($\mathbf{10^{-2}}$).

Method	MSE	MAE	ρ	τ	**p@10**	**p@20**
hungarian	20.54	34.38	81.10	60.36	61.00	99.00
vj	27.39	40.46	58.37	41.56	46.25	82.62
beam	11.40	20.68	78.67	**62.83**	62.75	90.00
GCN-Mean	1.25	9.09	76.39	53.38	56.50	**100**
GCN-Max	1.20	8.54	76.17	53.04	52.50	99.88
SimGNN	0.80	6.93	76.37	53.83	58.00	**100.00**
GSimCNN	0.23	3.25	**82.33**	**61.69**	**67.00**	**100.00**
GMN	0.15	2.71	77.22	54.50	53.25	**100.00**
PSimGNN-up	0.50	4.24	77.71	55.33	53.50	**100.00**
PSimGNN-k	0.12	2.51	79.65	57.81	57.75	**100.00**
PSimGNN	**0.11**	**2.41**	80.14	58.44	61.25	**100.00**

Table 5
Results on BA-200 dataset ($\mathbf{10^{-2}}$).

Method	MSE	MAE	ρ	τ	**p@10**	**p@20**
hungarian	25.91	37.94	79.38	58.10	**64.25**	94.00
vj	31.44	42.68	61.91	43.10	48.50	80.38
beam	18.60	28.79	77.24	**65.21**	56.00	83.50
GCN-Mean	2.37	12.78	73.47	49.46	50.00	95.00
GCN-Max	2.28	10.76	74.99	51.69	53.75	94.25
SimGNN	0.84	6.19	73.47	48.89	52.75	95.13
GSimCNN	0.32	3.58	**79.68**	56.82	59.00	95.00
GMN	0.12	2.66	79.58	**57.87**	**60.25**	95.00
PSimGNN-up	0.08	4.53	74.95	51.58	46.75	95.13
PSimGNN-k	0.07	2.14	76.36	53.29	52.50	96.00
PSimGNN	**0.06**	**1.96**	79.16	57.24	55.75	**97.63**

Table 2
Results on three common datasets ($\mathbf{10^{-2}}$). The best results are bolded.

Method	AIDS			lINUX			IMDB		
	mse	τ	p@10	mse	τ	p@10	mse	τ	p@10
hungarian	2.53	37.80	36.00	2.98	51.70	91.30	0.18	87.20	82.50
vj	2.92	38.30	31.00	6.39	45.00	28.70	0.18	87.40	81.50
beam	1.21	46.30	48.10	0.93	71.40	97.30	0.24	83.70	80.30
GCN-Mean	0.34	50.10	18.60	0.85	42.40	14.10	0.69	30.70	20.00
GCN-Max	0.37	48.00	19.50	0.64	49.50	43.70	0.51	34.20	42.50
SimGNN	0.12	69.00	42.10	0.15	83.00	94.20	0.13	77.00	75.90
GSimCNN	0.08	72.40	52.10	0.10	**96.20**	**99.20**	0.08	**84.70**	82.80
GMN	**0.07**	**73.20**	52.30	**0.08**	95.70	96.80	0.08	81.80	82.30
PSimGNN	0.11	70.10	**53.40**	0.12	91.50	97.70	**0.07**	82.20	**83.10**

Table 6
Results on IMDB-X dataset ($\mathbf{10^{-2}}$).

Method	MSE	MAE	ρ	τ	**p@10**	**p@20**
hungarian	0.27	1.66	93.09	83.18	74.09	80.91
vj	0.77	2.27	93.32	83.34	74.50	81.48
beam	**0.04**	**0.49**	**96.01**	**90.40**	**90.91**	**90.68**
GCN-Mean	2.22	5.54	46.64	36.20	48.64	70.22
GCN-Max	4.71	12.32	24.62	17.36	39.09	44.55
SimGNN	0.74	3.37	52.70	39.35	55.68	61.47
GSimCNN	0.50	3.04	66.26	49.87	62.05	64.09
GMN	0.38	2.73	69.59	55.38	65.68	71.82
PSimGNN-up	0.82	4.41	53.74	42.38	59.00	61.70
PSimGNN-*k*	0.42	3.01	68.23	51.42	62.50	64.43
PSimGNN	**0.31**	**2.51**	**72.34**	**60.31**	**68.18**	**73.86**

Table 7
Results for average time consumption on one pair of graphs in milliseconds.

Method	BA-60	BA-100	BA-200
hungarian	229	331	1192
vj	221	308	1010
beam	177	254	737
GCN-Mean	6.4	8.7	10.2
GCN-Max	7.1	9.2	11.3
SimGNN	4.4	5.6	6.9
GSimCNN	2.5	3.1	5.6
GMN	9.4	13.8	37.5
PSimGNN-up	3.8	5.0	9.4
PSimGNN-*k*	8.1	10.6	15.6
PSimGNN	15.6	20.0	30.0

(ρ, τ, and p@10) of BA-100 and BA-200, although PSimGNN is not optimal, which may be caused by the randomness of graph partitioning, it still gets close to GSimCNN and GMN in performance. This implies that our model introduces a more flexible framework and performs the same level of accuracy as other neural network-based models. And as the number of subgraph pairs using node-level comparison increases, the model contains more information. The corresponding evaluation results become better, which is also in line with our expectations. As mentioned before, for graphs in IMDB-X, there is no trimming steps. The Beam algorithm's accuracy is higher than the other two, so most of the ground truth is the approximate GEDs calculated by Beam. In this case, Beam shows the best performance in the Table 6, which also meets our expectations. At the same time, *PSimGNN* also shows the best performance compared with other deep learning methods on the IMDB dataset, which proves that the idea of graph partitioning is effective in the task of graph similarity computation.

As shown in Table 7, we recorded the running time for different models on test datasets. It is worth noting here that these models are implemented in different ways, and the framework itself may cause the time issue. In this case, it isn't easy to judge the computation cost directly by the running time is consumed. For example, GCN-Mean and GCN-Max, although their computation cost is $O(N)$, to get the best performance, there are several layers of node embedding, so the time consumed is not the lowest. And GSimCNN, with $O(N^2)$ computation cost, has the fastest running time due to the optimization of CNN in the framework used.

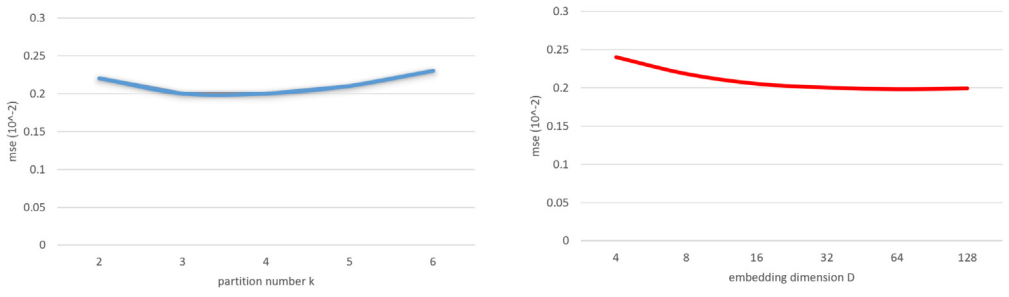
However, since we used the GMN code for reference when implementing node-level interaction in PSimGNN, the two still have a comparative value. By observing the running time of these two methods on three datasets of different scales, we can find that when the number of graph nodes is small, PSimGNN, which partitions first and then calculates the similarity, takes a long time due to a large number of steps. When the number of nodes is large, GMN directly interacts with the large graph at the node level with quadratic computation cost, which takes more time. Simultaneously, PSimGNN performs node interaction with the smaller graph, thus shortening the time, which proves our previous analysis of computation cost. At the same time, the experimental results of PSimGNN-up, PSimGNN-k, and PSimGNN also indicate that the more subgraphs involved in the computation, the longer it takes.

4.7. Ablation study

We evaluated how each of the components of PSimGNN affects the results. We report the results on the BA-60 dataset after removing a specific part. PSimGNN_sub_att represents our model without the attention mechanism in subgraph embedding. Here we use the average pooling method as an alternative. PSimGNN_cross_att indicates our model without the attention mechanism in node-level comparison. We also give each node the same weight. PSimGNN_cross and PSimGNN_within, respectively, means that our model removes the interaction within the subgraph and between subgraphs. PSimGNN is the complete model we proposed. It can be seen from Table 8 that regardless of removing each component, the performance of PSimGNN shows attenuation, and removing the attention mechanism has less impact on performance than the other two. This makes intuitive sense. Because for the latter two cases, they will lose a lot of information within or between graphs, which is not conducive to the high-quality

Table 8
Ablation Study on BA-60 dataset ($\mathbf{10^{-2}}$).. The best results are bolded.

Method	MSE	MAE	ρ	τ	**p@10**	**p@20**
PSimGNN_sub_att	0.31	3.91	83.17	62.50	72.75	90.37
PSimGNN_cross_att	0.29	3.88	82.29	61.30	73.50	90.87
PSimGNN_cross	0.61	5.72	79.83	57.82	71.50	89.37
PSimGNN_within	0.57	5.57	80.92	58.36	69.50	90.00
PSimGNN	**0.20**	**3.39**	**84.49**	**66.15**	**78.50**	**91.87**



(a) Partition Number k (b) Embedding Dimension D

Fig. 8. Mean squared error with respect to the partition number k and the dimensions D of subgraph-level embeddings.

embedding. This also further proves that the aggregation methods and attention mechanism are effective. With their joint contribution, PSimGNN can perform well in the graph similarity computation task.

4.8. Parameter sensitivity

In this part, we analyze the effect of partition number k and subgraph embedding dimension D on the performance of PSimGNN. We report the mean squared error on the BA-60 dataset. The lower the mse, the better the performance. Fig. 8(a) shows the performance comparison of the partition number k. As the number of partition subgraphs increases, the performance will first rise and then decline. This is because too few subgraphs cannot make full use of the graph's local features, and too many subgraphs will destroy the overall structure of the graph. Therefore, we need to choose a reasonable number of subgraphs to achieve optimal performance. Fig. 8(b) presents the performance comparison of the subgraph embedding dimension. In general, with the increase of the embedding dimension, the performance will increase. It makes intuitive sense since larger subgraph embedding dimension D can provide PSimGNN more capacity to represent subgraphs. However, when D reaches a certain level, around 32 here, performance growth becomes slow. Therefore, we need to find a proper length of embedding to balance the trade-off between the performance and the complexity.

5. Conclusion and future directions

We are at the intersection of graph neural network, graph similarity computation, and graph partition. We are taking the first step towards large graph similarity computation via graph partition and a novel neural network-based approach PSimGNN. The proposed method's central idea is to solve the problem of large graph similarity computation from the perspective of subgraphs, which takes any two graphs as input and outputs their similarity score. The experimental results show that PSimGNN achieves competitive accuracy and computation cost by introducing graph partitioning.

There are several directions to go for future work: (1) State-of-the-art graph partitioning methods usually consist of three main phases: coarsening, initial partitioning, and uncoarsening. It's very interesting to explore whether we can only deploy the coarsening stage on large graphs and split each large graph into some soft clusters (partitioning results is hard clusters). Then similarity computation based on these soft clusters may further reduce computational computation cost involved in the node–node similarity computation. (2) Introducing a mechanism to deal with edge attributes [46] is promising in some applications. In chemistry, atomic properties and bonds of a chemical compound are usually labeled, so it is useful to incorporate edge labels into our model. (3) Given the constraint that the exact GEDs for large graphs cannot be computed, we can only use approximate GEDs. When the number of graph nodes is further larger, the approximation algorithms become even less accurate. It would be interesting to see how the learned model generalizes to larger graphs trained only on the exact GEDs between partitioned subgraphs or other small graph datasets.

CRediT authorship contribution statement

Haoyan Xu: Conceptualization, Methodology. **Ziheng Duan:** Data curation, Software, Writing- Original draft preparation. **Yueyang Wang:** Writing- Reviewing and Editing, Writing- Original draft preparation. **Jie Feng:** Investigation, Software. **Runjian Chen:** Investigation, Software. **Qianru Zhang:** Visualization, Validation. **Zhongbin Xu:** Reviewing and Editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We thank Yunsheng Bai and Derek Xu for valuable discussions. This work is supported in part by the National Key Research and Development Program of China (No.2019YFB2102600, No.2019YFB1706101), the Fundamental Research Funds for the Central Universities (No.2020CDJQY-A005, NO.2020CDJQY-A022, No.2019CDQYRJ006), the National Natural Science Foundation of China (No.62002035, No.62072065, No.61902004), the Natural Science Foundation of Chongqing (No.cstc2020jcyj-bshX0034), Chongqing Research Program of Basic Research and Frontier Technology (No. cstc2019jcyj-msxmX0589) and the Science and Technology Innovation 2025 Major Project of Ningbo (No. 2018B10047).

References

[1] R. Albert, A.L. Barabási, Statistical mechanics of complex networks, Reviews of Modern Physics 74 (2002) 47.
[2] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, W. Wang, Simgnn: A neural network approach to fast graph similarity computation, in, in: Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, 2019, pp. 384–392.
[3] Y. Bai, H. Ding, K. Gu, Y. Sun, W. Wang, Learning-based efficient graph similarity computation via multi-scale convolutional set matching, in: AAAI, 2020, pp. 3219–3226..
[4] Y. Bai, H. Ding, Y. Sun, W. Wang, Convolutional set matching for graph similarity, 2018. arXiv preprint arXiv:1810.10866..
[5] D.B. Blumenthal, J. Gamper, On the exact computation of the graph edit distance, Pattern Recognition Letters (2018).
[6] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, C. Schulz, Recent advances in graph partitioning, in: Algorithm Engineering, Springer, 2016, pp. 117–158.
[7] H. Bunke, What is the distance between graphs, Bulletin of the EATCS 20 (1983) 35–39.
[8] H. Bunke, On a relation between graph edit distance and maximum common subgraph, Pattern Recognition Letters 18 (1997) 689–694.
[9] H. Bunke, K. Shearer, A graph distance metric based on the maximal common subgraph, Pattern Recognition Letters 19 (1998) 255–259.
[10] D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty years of graph matching in pattern recognition, International Journal of Pattern Recognition and Artificial Intelligence 18 (2004) 265–298.
[11] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, Advances in Neural Information Processing Systems (2016) 3844–3852.
[12] S. Fankhauser, K. Riesen, H. Bunke, Speeding up graph edit distance computation through fast bipartite matching, in: International Workshop on Graph-Based Representations in Pattern Recognition, Springer, 2011, pp. 102–111.
[13] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, Advances in Neural Information Processing Systems (2017) 1024–1034.
[14] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1026–1034.
[15] R. Horaud, T. Skordas, Stereo correspondence through feature grouping and maximal cliques, IEEE Transactions on Pattern Analysis and Machine Intelligence 11 (1989) 1168–1180.
[16] X. Hu, J. Xu, W. Wang, Z. Li, A. Liu, A graph embedding based model for fine-grained poi recommendation, Neurocomputing (2020).
[17] H. Jeong, Z. Néda, A.L. Barabási, Measuring preferential attachment in evolving networks, EPL (Europhysics Letters) 61 (2003) 567.
[18] R. Jonker, A. Volgenant, A shortest augmenting path algorithm for dense and sparse linear assignment problems, Computing 38 (1987) 325–340.
[19] V.G. Kaburlasos, L. Moussiades, A. Vakali, Fuzzy lattice reasoning (flr) type neural computation for weighted graph partitioning, Neurocomputing 72 (2009) 2121–2133.
[20] P. Kazienko, T. Kajdanowicz, Label-dependent node classification in the network, Neurocomputing 75 (2012) 199–209.
[21] M.G. Kendall, A new measure of rank correlation, Biometrika 30 (1938) 81–93.

[22] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014. arXiv preprint arXiv:1412.6980..
[23] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, 2016. arXiv preprint arXiv:1609.02907..
[24] H.P. Kriegel, M. Pfeifle, S. Schönauer, Similarity search in biological and engineering databases, IEEE Data Engineering Bulletin 27 (2004) 37–44.
[25] H.W. Kuhn, The hungarian method for the assignment problem, Naval Research Logistics Quarterly 2 (1955) 83–97.
[26] V.I. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, Soviet Physics Doklady (1966) 707–710.
[27] Y. Li, C. Gu, T. Dullien, O. Vinyals, P. Kohli, Graph matching networks for learning the similarity of graph structured objects, 2019. arXiv preprint arXiv:1904.12787..
[28] Y. Liang, P. Zhao, Similarity search in graph databases: A multi-layered indexing approach, in: 2017 IEEE 33rd International Conference on Data Engineering (ICDE), IEEE, 2017, pp. 783–794.
[29] T. Ma, W. Shao, Y. Hao, J. Cao, Graph classification based on graph set reconstruction and graph kernel feature reduction, Neurocomputing 296 (2018) 33–45.
[30] M. Neuhaus, H. Bunke, Edit distance-based kernel functions for structural pattern classification, Pattern Recognition 39 (2006) 1852–1863.
[31] M. Neuhaus, K. Riesen, H. Bunke, Fast suboptimal algorithms for the computation of graph edit distance, in: Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR), Springer, 2006, pp. 163–172.
[32] K. Ogaard, H. Roy, S. Kase, R. Nagi, K. Sambhoos, M. Sudit, Discovering patterns in social networks with graph matching algorithms, International Conference on Social Computing, Behavioral-Cultural Modeling, and Prediction, Springer (2013) 341–349.
[33] S.K. Pal, S. Mitra, Multilayer perceptron, fuzzy sets, classification, 1992..
[34] F. Parés, D.G. Gasulla, A. Vilalta, J. Moreno, E. Ayguadé, J. Labarta, U. Cortés, T. Suzumura, Fluid communities: a competitive, scalable and diverse community detection algorithm, in: International Conference on Complex Networks and their Applications, Springer, 2017, pp. 229–240.
[35] M. Pelillo, K. Siddiqi, S.W. Zucker, Matching hierarchical structures using association graphs, IEEE Transactions on Pattern Analysis and Machine Intelligence 21 (1999) 1105–1120.
[36] K. Riesen, H. Bunke, Approximate graph edit distance computation by means of bipartite graph matching, Image and Vision computing 27 (2009) 950–959.
[37] K. Riesen, S. Emmenegger, H. Bunke, A novel software toolkit for graph edit distance computation, in: International Workshop on Graph-Based Representations in Pattern Recognition, Springer, 2013, pp. 142–151.
[38] C. Spearman, The proof and measurement of association between two things, 1961.
[39] K. Steinhaeuser, N.V. Chawla, Community detection in a large real-world social network, in: Social Computing, Behavioral Modeling, and Prediction, Springer, 2008, pp. 168–175.
[40] Y. Tian, R.C. Mceachin, C. Santos, D.J. States, J.M. Patel, Saga: a subgraph matching tool for biological graphs, Bioinformatics 23 (2007) 232–239.
[41] X. Wang, X. Ding, A.K. Tung, S. Ying, H. Jin, An efficient graph indexing method, in: 2012 IEEE 28th International Conference on Data Engineering, IEEE, 2012, pp. 210–221.
[42] B.X. Wu, J. Xiao, J.M. Chen, Friend recommendation by user similarity graph based on interest in social tagging systems, in: International Conference on Intelligent Computing, Springer, 2015, pp. 375–386.
[43] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks?, 2018, arXiv preprint arXiv:181000826.
[44] P. Yanardag, S. Vishwanathan, Deep graph kernels, in: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2015, pp. 1365–1374.
[45] Z. Zeng, A.K. Tung, J. Wang, J. Feng, L. Zhou, Comparing stars: On approximating graph edit distance, Proceedings of the VLDB Endowment 2 (2009) 25–36.
[46] M. Zhang, H. Hu, Z. He, L. Gao, L. Sun, A comprehensive structural-based similarity measure in directed graphs, Neurocomputing 167 (2015) 147–157.

Haoyan Xu received his B.S. in July, 2020 at Zhejiang University, College of Control Science and Engineering. His research interests lie in the area of Graph Representation Learning, Time Series Analysis, Robot Learning and Microfluidics. He is particular interested in graph neural networks, with their applications in language processing, graph mining, etc. What's more, he likes studying micro-scale robotics and bioinspired robotics and exploring fundamental questions related with them.

Ziheng Duan received his B.S. in July, 2020 at Zhejiang University, College of Control Science and Engineering. His research interests lie in the area of Machine Learning, Graph Representation Learning, Time Series Analysis, espeically the interaction of them.

Yueyang Wang received the B.E. degree from the Software Institute, Nanjing University, Nanjing, China, and the Ph.D. degree from Zhejiang University, Hangzhou, China. She is currently a Lecturer with the School of Big Data and Software Engineering, Chongqing University. Her research interests include social network analysis and data mining.

Jie Feng is currently a senior student at Zhejiang University, College of Control Science and Engineering, who will receive his B.S. in June. 2021. His research interests include artificial intelligence and robotics.

Runjian Chen received the B.S. degree from the Department of Control Science and Engineering, Zhejiang University, Hangzhou, China, in 2020. His latest research interests include machine learning and its application in robotics and computer vision.

Qianru Zhang is studying at Harbin Institute of Technology and will receive her bachelor degree in 2021. Her research interests lie in Machine Learning, Time Series Analysis and Graph Representation Learning.

Zhongbin Xu is a professor in Energy Engineering College at Zhejiang University. He received his PhD from South China University of Technology in 2001. He was a visiting scholar at the University of Cambridge (2008–2009) and Harvard University (2014–2015). His current research interests include intelligent microstructure manufacturing, polymer processing engineering and microfluidics. His group has done lots of work in the application of industrial intelligence.