Soft Computing (2024) 28:61–76
https://doi.org/10.1007/s00500-023-09288-1

FOUNDATION, ALGEBRAIC, AND ANALYTICAL METHODS IN SOFT COMPUTING

DeepSIM: a novel deep learning method for graph similarity computation

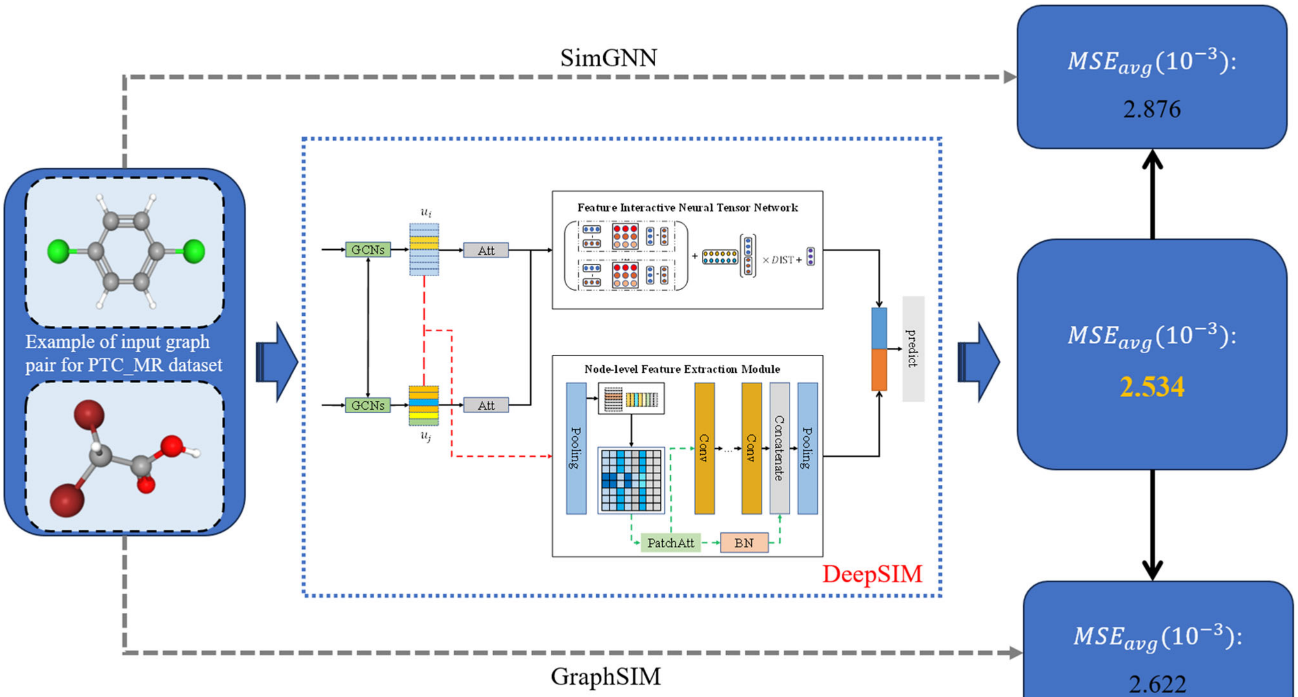
Bo Liu[1] · Zhihan Wang[2] · Jidong Zhang[2] · Jiahui Wu[2] · Guangzhi Qu[3]

Accepted: 16 September 2023 / Published online: 12 October 2023

Abstract

Graphs are widely used to model real-life information, where graph similarity computation is one of the most significant applications, such as inferring the properties of a compound based on similarity to a known group. Definition methods (e.g., graph edit distance and maximum common subgraph) have extremely high computational cost, and the existing efficient deep learning methods suffer from the problem of inadequate feature extraction which would have a bad effect on similarity computation. In this paper, a double-branch model called DeepSIM was raised to deeply mine graph-level and node-level features to address the above problems. On the graph-level branch, a novel embedding relational reasoning network was presented to obtain interaction between pairwise inputs. Meanwhile, a new local-to-global attention mechanism is designed to improve the capability of CNN-based node-level feature extraction module on another path. In DeepSIM, double-branch outputs will be concatenated as the final feature. The experimental results demonstrate that our methods perform well on several datasets compared to the state-of-the-art deep learning models in related fields.

Graphical abstract


SimGNN
$MSE_{avg}(10^{-3})$: 2.876
Example of input graph pair for PTC_MR dataset
u_i
GCNs
Att
Feature Interactive Neural Tensor Network
× DIST +
GCNs
Att
u_j
Node-level Feature Extraction Module
Pooling
Conv
Conv
Concatenate
Pooling
PatchAtt
BN
predict
DeepSIM
$MSE_{avg}(10^{-3})$: 2.534
$MSE_{avg}(10^{-3})$: 2.622
GraphSIM

Keywords Deep learning · Graph similarity · Embedding relational reasoning · Double branch

1 Introduction

Graphs are called non-Euclidean data (Bronstein et al. 2017), which is different from the grid structured types such as pictures, text, audio, video and so on. In real life, many data (e.g., traffic flow, social network, compounds, brain structure, web pages and many more) are graphically structured or can be easily transformed into graphical representations. Graph-based analysis method has been applied to a wide range of industries and achieved ideal results. Among them, calculating the similarity of graph pairs is an important application. For instance, in multi-subject brain network analysis, graph similarity calculation is helpful to clinical investigation and disease diagnosis (Ma et al. 2019); in web pages monitoring, graph similarity is necessary to get reliable PageRank values in the evolving process (Papadimitriou et al. 2010); in the field of protein related bioinformatics, the similarity is significant for pairwise interaction (Zaslavskiy et al. 2009); recently, similarity calculation combined with approximate pruning algorithm has been used for fast similarity search of graph databases (Qin et al. 2020).

Graph Edit Distance (GED) (Bunke and Allermann 1983) and Maximum Common Subgraph (MCS) (Bunke and Shearer 1998) are the most commonly used in the definition of graph similarity. The former (GED) is inspired by string edit distance which takes the minimum value of operation cost function as distance metric, and the latter (MCS) focuses on the number of maximum common subgraph nodes of the input graph pair. However, these two algorithms have been proved to be NP-hard, which bring unacceptable computational burden in use (Zeng et al. 2009; Gao et al. 2010).

Many improved methods of graph similarity measurement based on the above two ideas are presented to add feature details or reduce the computational cost, such as applying kernel function to structural pattern (Neuhaus and Bunke 2006), approximate calculation method (Riesen and Bunke 2009). Furthermore, EMD and PM algorithms (Nikolentzos et al. 2017) describe the similarity of graph pairs from the perspective of vertex embedding comparison, the former (EMD) using SVM to solve the non-semi-positive definite matrix problem and the latter (PM) introducing the pyramid matching function to compare the vertex set differences of graph pairs, but both have high computational complexity. With the development of graph signal processing, different from the traditional approach, Graph Fourier Distance (GFD) based on spectral decomposition of Laplacian matrix provides a new way to describe the differences between graph pairs in frequency domain (Lagunas et al. 2018).

Due to the need of massive and large-scale graph data analysis, data-driven approach, especially deep learning method has become the mainstream of current research in this field. Deep learning method can cover the potential features (or hidden patterns) from a large number of data and only care about the gap between prediction and target value in the training process. Under the influence of such thoughts, 2D CNN (Tixier et al. 2019) obtains node embedding for a graph and then uses PCA to process feature results which are regarded as images, finally, the high-order features are captured by CNNs for classification. It is different from traditional CNN which can't deal with graph structure data directly, graph neural network (GNN) captures whole graph features to obtain its vectorized representation, which is helpful for this research (Ktena et al. 2017; Bai et al. 2018). After training, the similarity score between the input graph pairs can be predicted quickly with acceptable error rate. GNN can get reliable network representation (Wu et al. 2020), however, the difference of graphs usually exists in the small subgraphs and the graph-level embedding may ignore this part of information (Bai et al. 2019; Ma et al. 2021; Ling et al. 2022). In addition, HS-GCN (Ma et al. 2019) adds higher-order information by random walking on the graph to improve the model computational performance. MGMN (Ling et al. 2020) proposes a node-graph matching layer to add cross-level interactions between graph pairs and then combines siamese graph neural networks for similarity computation and graph matching. GraphSIM (Bai et al. 2020), which achieves good performance in similarity computation task, proposes a multi-scale GCN to mitigate the feature loss from multiple information aggregation and uses the BFS algorithm to order the nodes to construct a similarity matrix.

Inspired by SimGNN (Bai et al. 2019), we developed a novel deep learning model for graph similarity calculation called DeepSIM. The proposed approach was designed in three stages: (1) embedding; (2) double-branch feature extraction module; (3) score prediction (i.e., output module). GCN was used to get the node-level graph representation and feed it to Stage 2 just like SimGNN. We have assumed that in some practical applications, when the node-level features have a great impact, the non-learnable method (strategy 2 of SimGNN) or the method only considering graph-level branch will have a bad influence on the performance of the model, so CNN-based node feature extraction module with a new attention layer to replace the histogram method in SimGNN and the conjecture mentioned above is proved in the experiment. At the same time, a feature-feature interaction approach called Feature Interactive Neural Tensor Network (FINTN), inspired by Neural Tensor Network (NTN) (Socher

✉ Bo Liu
b.liu@massey.ac.nz

[1] School of Mathematical and Computational Sciences, Massey University, Auckland 0745, New Zealand

[2] Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China

[3] Computer Science and Engineering Department, Oakland University, Rochester, MI, USA

et al. 2013), was raised which achieved good results in the ablation experiment. Node-level method and graph-level module constitute two branches and the features of two different concerns are fused before the Stage 3. Similar to the traditional deep learning prediction task, fully connected network is used to predict the score of Stage 2 results and calculate the difference with the ground-truth which is generated by an optimized method called DF-GED (Abu-Aisheh et al. 2015) implemented in NetworkX with normalization. DF-GED algorithm accelerates the calculation by building a search tree, and records the editing operation cost as a node operation matrix to obtain the variance metric of the graph pair.

The main contributions of our work are as follows:

- A new graph similarity calculation model with two branches was proposed. The two paths complement each other, which ensures the robustness of our approach. The model has stable and good performance on multiple datasets.
- On the graph-level branch, the distance and direction attributes of features in high-dimensional space are considered, which improves the feature interaction in traditional NTN intuitively. Our method (FINTN) has proven effective in experiments which gives a new guidance for the construction of relational inference models.
- Global average pooling was used to unify the node embedding dimensions instead of inserting fake nodes. CNN-based node-level feature extraction module with a new attention mechanism was raised to emphasize the local interaction and global impact of nodes.

The rest of this paper is organized as follows. In Sect. 2, the previous related methods are summarized and problem definitions will be given. In Sect. 3, the proposed approach (DeepSIM) will be described in detail. In Sect. 4, results of comparative experiments with correlation analysis will be shown. In Sect. 5, we will summarize this work and present some ideas for the future research.

2 Definitions and related work

Graph data G_i is composed of node set V_i and edge set E_i, that is $G_i = (V_i, E_i, A_i)$, where $v_a \subseteq V_i$ represents a node of G_i.If there is a connection between v_a and v_b,then there is an edge $e_{ab} \subseteq E_i$, A_i is the adjacency matrix of G_i, the number of nodes in V_i denoted as $N = |V_i|$. For a graph dataset with n samples $\mathbb{G} = \{G_1, G_2, \ldots, G_n\}$, some related definitions can be used (as shown in below section) to get the similarity between graph pairs as ground-truth for the learning task. In this study, a model based on undirected and unweighted graph was build and it could be easily applied to any graph structure data by changing the graph embedding method.

2.1 Graph similarity definition

Just as mentioned above, GED and MCS are the two most commonly used definitions. Suppose there are two graphs, G_1 and G_2, exact GED (as shown in Fig. 1) is the minimum number of operations to transform G_1 to G_2 (including node and edge addition, insertion and deletion), approximate GED reduces the acceptable accuracy to improve the computational efficiency significantly (Xu et al. 2018).

MCS (as shown in Fig. 2) aims to find the maximum common subgraph of two graphs, the formula is as follows:

$$d(G_1, G_2) = 1 - \frac{|\text{mcs}(G_1, G_2)|}{\max(|G_1|, |G_2|)} \qquad (1)$$

in the formula, $|\text{mcs}(G_1, G_2)|$ is the number of maximum common subgraph nodes between G_1 and G_2, $|G_1|$ and $|G_2|$ represent the number of nodes in G_1 and G_2 respectively.

The definition method to calculate similarity is generally applicable to paired input graph data, but usually has a high computational complexity which is difficult to cope with large-scale and massive computing requirements.

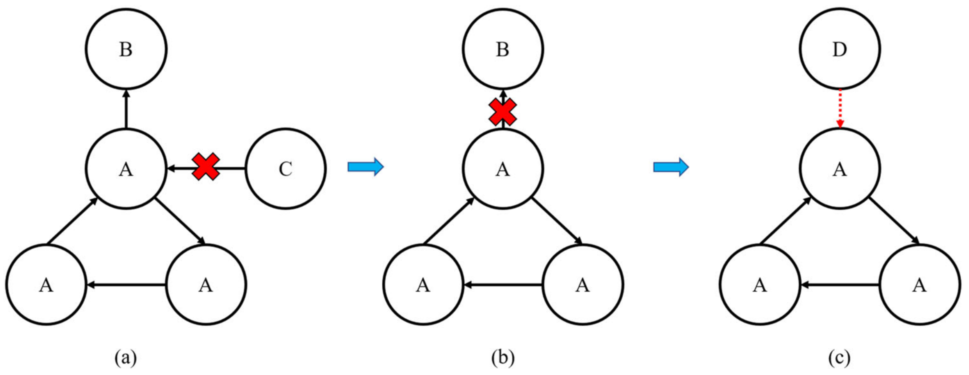


Fig. 1 An example of GED. **a** delete node C and corresponding edge, **b** delete node B and corresponding edge, **c** add node D and corresponding edge. The GED is 3

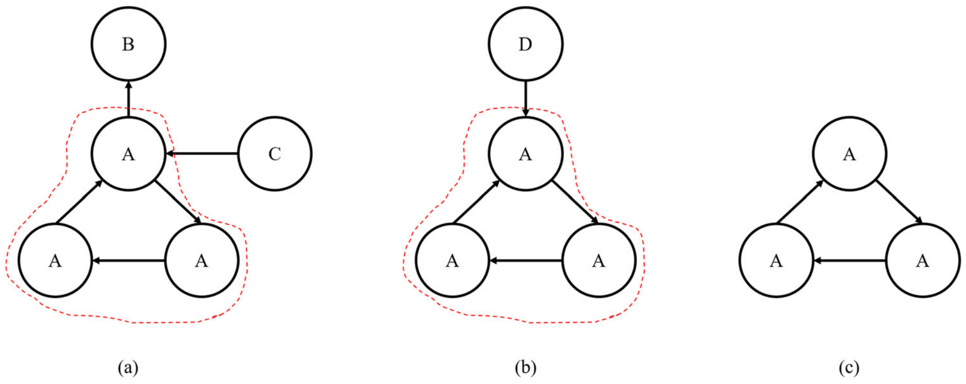


Fig. 2 An example of MCS. **a** a graph G_1, **b** another graph G_2, **c** the maximum common subgraph (G_s) of G_1 and G_2. The number of nodes in G_1 and G_2 are 5 and 4, respectively. And the node number of G_s is 3. Here, we have $MCS\,(G_1, G_2) = 0.4$

2.2 Graph embedding and GCN

Similarity calculation method based on graph embedding (network embedding) needs reliable algorithm to map graph data to vector space, and then use measurement (such as Euclidean Distance, Cosine Similarity and Minkowski Distance) to calculate feature distance. Therefore, graph representation learning has received increasing attention in this field due to its practical significance (Chen et al. 2020). DeepWalk (Perozzi et al. 2014) and node2vec (Grover et al. 2016) are based on random walk algorithm, which randomly select initial vertex as an entrance to traverse, the paths and relationships between the connected nodes are obtained to describe the graph from the node-level perspective. Where the latter (node2vec) algorithm introduces two parameters p (return parameter) and q (in-out parameter) to balance the impact of both BFS and DFS policies, and when p equals q, node2vec is equivalent to DeepWalk. Inspired by Doc2Vec (Le and Mikolov 2014), Graph2Vec (Narayanan et al. 2017) is different from the previous two node-level embedding algorithms, which directly generates the embedding of the entire graph, that is, an output of this method represents a certain graph. The execution efficiency of the above graph embedding algorithm is very sensitive to the input data size and scale. In recent years, data-driven deep learning method has also been developed in the field of graph embedding/graph representation, for example, GCN (Kipf et al. 2016), GraphSAGE (Hamilton et al. 2017) and GAT (Veličković et al. 2017). Transformation to frequency space with the help of the Laplacian matrix of the graph to define the graph convolution layer to build GCN, and GraphSAGE, on the other hand, achieves information fusion within node neighborhoods by designing reliable aggregation functions. GAT uses a multi-headed attention mechanism to assign power to each node to emphasize the different influences in the information aggregation process. Each embedding characterizes one piece of input graph data and describes the differences by a distance metric to reflect the similarity between the original graph pairs.

GCN is used as a case study to look into how to obtain reliable embeddings, we assumed that there is a graph $G_1 = (V_1, E_1, A_1)$ with its degree matrix D_1 which is a diagonal matrix,and the elements on the diagonal are the degrees of each vertex. The Laplacian matrix L_1 of G_1 can be easily obtained, i.e., $L_1 = D_1 - A_1$ and its normalized form $L_1 = I_N - D_1^{-\frac{1}{2}} A_1 D_1^{-\frac{1}{2}}$, where I_N is identity matrix. The eigenvalue decomposition of normalized L_1 is expressed as $L_1 = U_1 \wedge_1 U_1^T$ where the lth column of U_1 is eigenvector and $\wedge_1(l, l)$ is the corresponding eigenvalue (Ortega et al. 2018; Zhang et al. 2019). Graph Fourier transform (GFT) is an extension of Fourier transform in non-Euclidean domain which is defined as: $\widetilde{x_1} = {U_1}^T x_1$ where x_1 represents a set of signals defined on V_1 and inverse graph Fourier transform (IGFT) can be written as: $x_1 = U_1 \widetilde{x_1}$. Motivated by CNN, graph convolution in frequency domain can be defined as:

$$x_{1_1} * x_{1_2} = \text{IGFT}\left(\text{GFT}\left(x_{1_1}\right) \bigodot GFT\left(x_{1_2}\right)\right) = H_{\widetilde{x_{1_1}}} x_{1_2} \tag{2}$$

where $H_{\widetilde{x_{1_1}}} = U_1 \text{diag}(\widetilde{x_{1_1}}) U_1^T$.

According to previous work (Narayanan et al. 2017), considering frequency domain representation $y_{\text{output}} = \sigma(g_\theta * x_1) = \sigma(U_1 g_\theta U_1^T x_1)$ where x_1 denotes a scalar for every node and σ represents sigmoid function, Chebyshev polynomials are used to reduce the computational cost of g_θ:

$$y'_{\text{output}} = \sigma\left(g'_\theta * x_1\right) \approx \sigma\left(\sum_{k=0}^{K} \theta'_k T_k(\widetilde{L_1}) x_1\right) \tag{3}$$

where $\widetilde{L_1} = \frac{2}{\lambda_{\max}} L_1 - I_N$, $\lambda_{\max}$ denotes the largest eigenvalue of L_1 and K is the max steps away from the central node. With the above definition of K-hop convolution GCN can be built.

After the graph embedding phase, a suitable distance metric is used to describe the differences between the paired graph representations and is transformed into a similarity score by a normalization function. The data-driven GNN approach, represented by GCN, transforms the graph representation into an end-to-end graph deep learning task, which helps to explore the potential associations of the original graph pairs and is beneficial for fine-grained similarity computation tasks.

2.3 Kernel for graph representation

Different from the graph embedding, the graph kernel function pays more attention to map the original structured information into Hilbert space ($\mathbb{H}$) and avoids the loss of spatial information in the process of vectorization (Kriege et al. 2020). Gärtner et al. proposed a random walk graph kernel by recording common steps (Gärtner et al. 2003) for high-dimensional representation of the graph. In addition, Shervashidze et al. constructed graph kernel functions by subgraph segmentation combined with the Weisfeiler-Lehman isomorphism detection algorithm to improve the capability of topological and label information capture (Shervashidze et al. 2011; Weisfeiler and Leman 1968). Inspired by the Weisfeiler–Lehman kernel, Weisfeiler–Lehman similarity (WLS) was raised to describe the distance between two graphs by adding the similarity between neighbors' attributes into node feature update process and further build the WLS neural network (Ok 2020). Although graph kernel functions have gained widespread attention in the field of graph representation, they are difficult to define and may weaken the generalization ability of the model when the amount of training data is small due to their focus on raw data information. Therefore, in this paper, we focus more on graph embedding methods.

2.4 Siamese neural network

Siamese structure was first proposed for signature verification (Bromley et al. 1993), and later combined with a variety of shared-weight neural network modules for metric learning, gaining widespread attention in the fields of face recognition (Chopra et al. 2005), image patches comparison (Zagoruyko et al. 2015) and one-shot image recognition (Koch et al. 2015). The structure of siamese neural network is shown in Fig. 3.

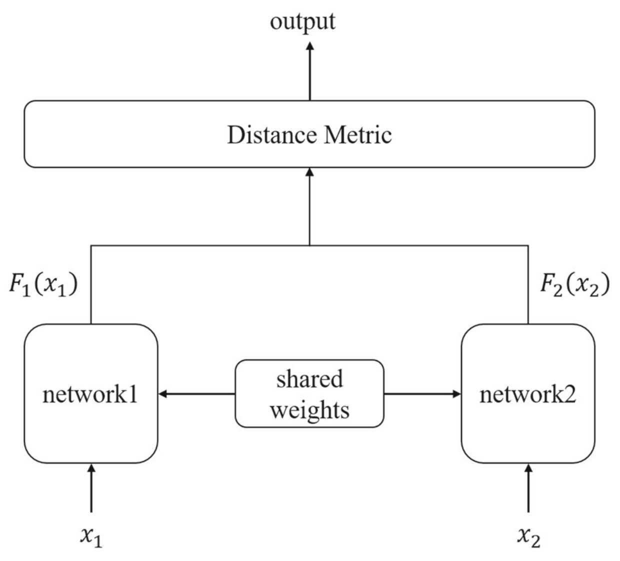


Fig. 3 An overview of siamese neural network

In Fig. 3, x_1 and x_2 denote a pair of input data, network 1 and network 2 are called two sub-networks and are usually used for feature extraction. The siamese structure is implemented by sharing weights while ensuring that two sub-networks process pairs of input in the same way. The output of the two branches is aggregated with information via a distance metric (also known as similarity estimation). The aggregated feature vector is used as the output of this network.

The choice of sub-modules for siamese neural networks is very flexible, thus Siamese GCN (S-GCN) (Ktena et al. 2018) uses spectral GCN as a sub-network proposed for graph similarity calculation. Just as mentioned above, HS-GCN (Ma et al. 2019) is an extension of S-GCN that adds higher order information by applying the random walk algorithm before the graph pair is fed into the siamese graph convolution layer and achieves good performance in similarity learning task. Not just popular in this particular field, a variant of S-GCN, Context Attended-SGCN (Chaudhuri et al. 2022), is used for remote sensing image retrieval by region-adjacency graph (RAGs) generation and adding node attention.

Siamese neural network is good at the task of pattern recognition with paired inputs, and is a good fit for the task of graph similarity computation because of its high tolerance for error samples (Chen et al. 2020), which becomes the basic framework of this study.

2.5 Neural tensor network

In Neural Tensor Network, for the sake of finding the relationship score of entity vector pairs across multiple dimensions, bi-linear tensor product is used to replace the linear form in traditional neural network. The definition is as follows:

$$g\left(e_1, R, e_2\right) = U_R^T f(e_1^T W_R^{[1:k]} e2 + V_R \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} + b_R) \quad (4)$$

where $g\left(e_1, R, e_2\right)$ is the score, $f(x) = \tanh(x)$, e_1 and e_2 are vector representations of input entities, $W_R^{[1:k]}$ is the ith slice of tensor $W(i = 1, 2, 3, \ldots, k)$, U_R^T is used to obtain a certain relation score. $V_R \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$ is called standard layer which is a traditional deep learning embedding relational inference method based on splicing.

In previous work (SimGNN), NTN without U_R^T is used to infer the relationship feature vector between two graph-level embeddings. For each graph embedding, it is obtained by aggregating the node-level embedding of the GCN output through the node attentive layer, which is defined as follows:

$$\begin{aligned} h &= \sum_{n=1}^{N} f_2\left(u_n^T c\right) u_n \\ &= \sum_{n=1}^{N} f_2\left(u_n^T \tanh\left(\left(\frac{1}{N}\sum_{m=1}^{N} u_m\right) W_2\right)\right) u_n \end{aligned} \tag{5}$$

where h denotes a graph-level embedding, $f_2(\bullet)$ means sigmoid function, u_n represents the embedding of the n-th node and W_2 is the weight matrix. It is not difficult to find that h is the weighted sum of node-level embeddings for a certain input graph.

Although in the field of embeddings relation score calculation, NTN significantly outperforms the traditional methods, its performance relies only on parameters update within the module during the training process, to a certain degree, ignoring the inherent association between the input vector pairs which may be harmful for the fine-grained similarity calculation task.

3 The proposed method

Inspired by related work, we raised a graph **sim**ilarity calculation model based on **deep** learning to address the shortcomings in some previous methods, which can be abbreviated as **DeepSIM**.

DeepSIM (as shown in Algorithm 1) consists of three stages: (1) embedding with GCN; (2) double-branch feature extraction; (3) prediction module. In Stage 1, GCN-based method transforms the original input data into vector representation in feature space. In Stage 2, there are two paths, one to capture graph-level features, the other to extract node-level association. In Stage 3, combined output of Stage 2 is fed into the last module (fully connected network) to get the prediction of paired input. The overview of DeepSIM is shown in Fig. 4 and we will present the proposed improvement strategies in detail in the subsections of Stage 2.

Algorithm 1 DeepSIM

```
Input: ${(x_n, z_n)}_{n=1}^{N_{data}}$, a dataset of graph pairs.
Output: Similarity between input graph pairs.
Hyperparameters: $\mathcal{N}_{epochs} = 5$
1: for i=1,2, ..., $\mathcal{N}_{epochs}$ do
2:     Get the respective embedding $x_e$ and $z_e$ of data $x_n$ and $z_n$ by Siamese GCN(Stage 1).
3:     Feed $x_e$ and $z_e$ into the Double-branch Feature Extraction Module that focuses on graph-level features and node-level features.
4:     To obtain node-level features $F_n$, $x_e$ and $z_e$ construct the feature map $F$ by multiplying the vectors and input $F$ in Node-level Feature Extraction Module.
5:     FINTN is used to get graph-level interaction features($F_g$).
6:     Connect $F_n$ and $F_g$ to form fusion feature and put it into a fully connected neural network to calculate the similarity score.
7: end for
8: return similarity score
```

3.1 Stage 1: Siamese GCN for node embedding

In Stage 1, we use traditional graph classification method based on deep learning, and GCN was chosen to get the node-level embedding of graph data. For pairwise comparison task, siamese structure constructed by sharing weights can better evaluate the similarity of two inputs by generating their respective high-dimensional vector representation. However, having only two embeddings are not enough to cover the information of the graph pair more completely because, like CNN, GCN lose some of the low-dimensional information when capturing features. Therefore, further analysis of the output of this stage is urgently needed.

3.2 Stage 2: Double-branch feature extraction module

The difference of graph structure data is not only reflected in the distance of the whole graph embedding, but also in the local substructure. In DeepSIM, double-branch feature extraction module was proposed to get a reliable pairwise associated features. Node-level embeddings of input graphs can be obtained by siamese GCN just as mentioned above. Node attentive mechanism, which was proposed in SimGNN, combined with a novel reasoning module was applied to extract graph-level interaction. Node-level embedding after pooling operation constructs interaction matrix by vector product to reflect substructure information. In order to better capture local feature, similar to traditional computer vision methods, we designed a feature extraction module based on CNN combined with residual connection and proposed a new patch attention mechanism (PatchAtt) to emphasize the local correlation on the interaction matrix. The output features of two approaches mentioned above will be concatenated before prediction module. The introduction of our new methods will be presented in detail in the following subsections.

3.2.1 Approach 1: feature interactive neural tensor network

After the node attention mechanism, in order to obtain the graph-level interactions between two embeddings, NTN is introduced in SimGNN to reason about deeper relations. However, NTN feature interaction mechanism is not intuitive enough, i.e., it can increase the interaction between vectors in feature space rather than only relying on updating of trainable parameters. For this motivation, Feature Interactive Neural Tensor Network (FINTN) was developed, as shown in Fig. 5,

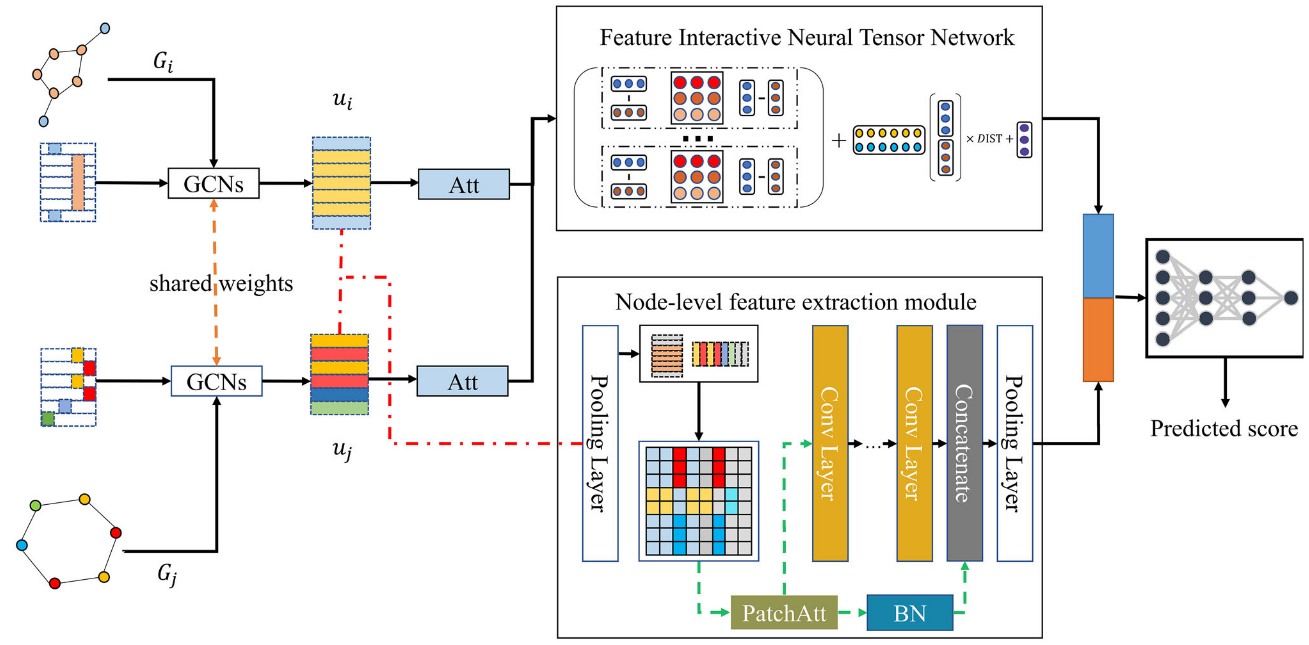
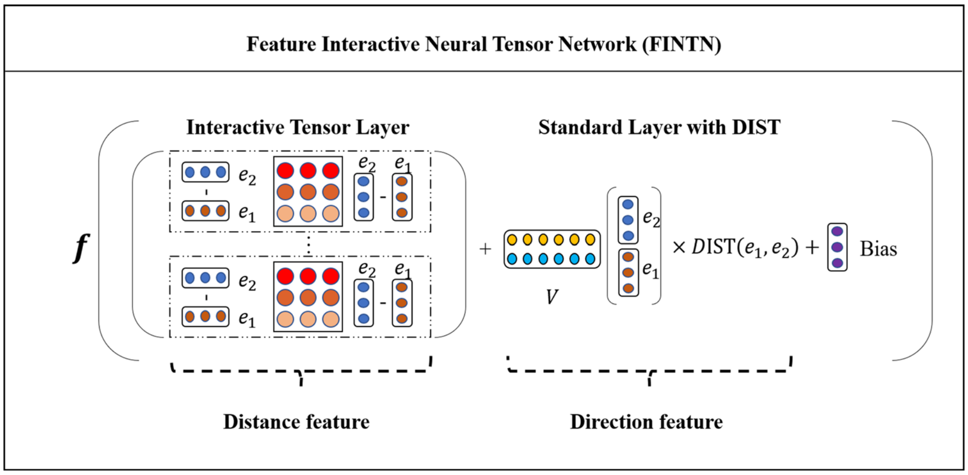


Fig. 4 An overview of DeepSIM

Fig. 5 Overview of FINTN which improved interactive method of NTN to get the relationships of input graph samples



to get graph-graph relation preferably. The input of this module, which is expressed as e_1 and e_2 in the following formula, is a pair of high-dimensional features generated by siamese GCN. Due to the characteristics of this architecture, e_1 and e_2 are in the same vector space which means the distance and direction (commonly used similarity indexes for a pair of feature vectors) of e_1 and e_2 can be easily derived. Bi-linear tensor product and standard layer of NTN were redesigned to represent the interaction intuitively, and the proposed method is defined as follows:

$$F(e_1, R, e_2) = f\left((e_2 - e_1)^T W_R^{[1:k]} (e_2 - e_1) + V_R \begin{bmatrix} e_2 \\ e_1 \end{bmatrix} * \text{DIST}(e_1, e_2) + b_R\right) \quad (6)$$

where $F(e_1, R, e_2)$ means the relation between e_1 and e_2, $f(\bullet)$ denotes tanh, $W_R^{[1:k]}$ is the slice of tensor and b_R represents bias just like NTN, $V_R \begin{bmatrix} e_2 \\ e_1 \end{bmatrix}$ is standard layer, $\begin{bmatrix} e_2 \\ e_1 \end{bmatrix}$ denotes the concatenation operation of pairwise input e_1 and e_2, DIST is the cosine similarity of two vectors:

$$\text{DIST}(e_1, e_2) = \text{cosine}(e_1, e_2) = \frac{e_1 \bullet e_2}{e_1 \times e_2} \quad (7)$$

In FINTN, by simply computing the distance and direction between two same-dimensional embeddings in vector space, there is no significant increase in computational cost compared to NTN, i.e., both have the same time complexity. However, FINTN is superior in the graph embedding relation inference task, which proves the above conjecture that NTN

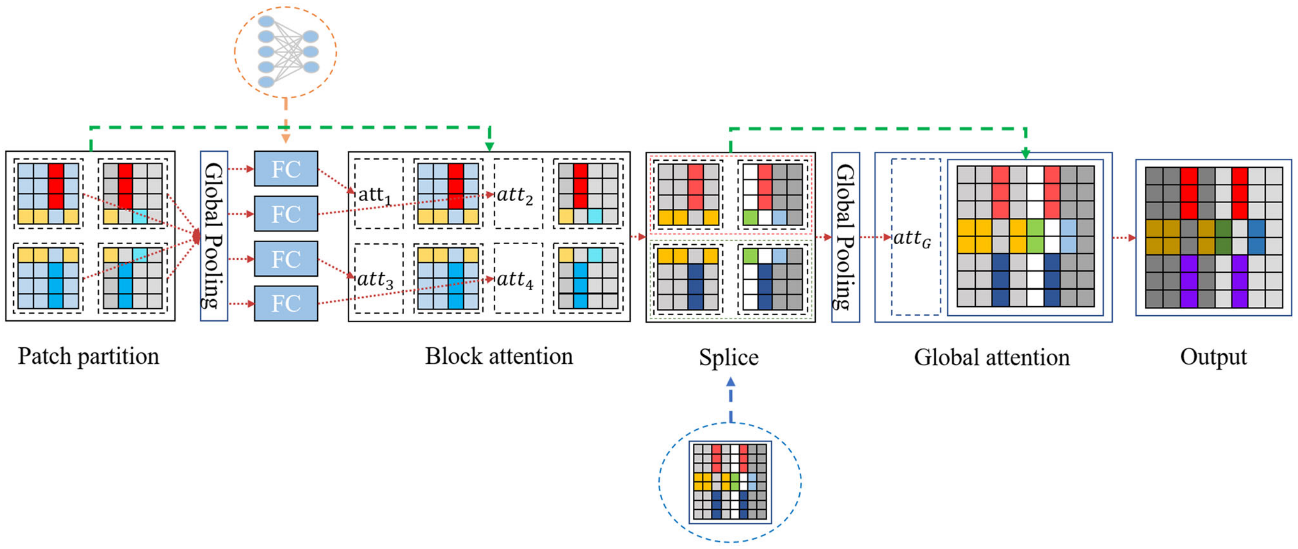


Fig. 6 Overview of PatchAtt mechanism which calculates the attention score to guide the model to select more important blocks in training to emphasize fine-grained node-level interaction

inference method ignores the inherent association between embeddings, and the statement will be covered in next section through comparison experiments.

3.2.2 Approach 2: Node-level feature extraction module based on CNNs

CNN-based method for computer vision task has been proved to be effective and widely used in various related work. After stage 1, the embeddings of graph pairs are obtained and after pooling operation (to unify the feature vector dimension) the node-level interaction matrix $M \in \mathbb{R}^D$ can be generated by vector product. In this work, the matrix can be regarded as gray scale image so that CNNs can be used to analyze it. For the sake of mining the more concerned interaction between nodes, inspired by SEnet (Hu et al. 2018), patch attention mechanism (PatchAtt just as shown in Fig. 6) was proposed to improve acuity of this branch which divided interaction matrix into h patches with equal size and fully-connected network was used to set different weights for each block after pooling. Each patch was multiplied by the attention score and spliced into a new interaction matrix $M_{\text{att}} \in \mathbb{R}^D$ according to the original position. The influence score of each block on the overall matrix is directly calculated by global pooling and the output of PatchAtt can be defined as follows:

$$M_{\text{output}} = \mathcal{G}\left(M_{\text{att}}\right) * M_{\text{att}} \tag{8}$$

where $M_{\text{att}} = \text{concat}(\text{patch}_{\text{att}_i}),\ i = 1, 2, 3, \ldots, h,\ \mathcal{G}$ means global average pooling and concat in this definition denotes block splicing method related to original position, $\text{patch}_{\text{att}_i}$ can be calculated as follows:

$$\text{patch}_{\text{att}_i} = \sigma\left(\text{FCs}\left(\mathcal{G}\left(\text{patch}_i\right)\right)\right) * \text{patch}_i,\ i = 1, 2, 3, \ldots, h \tag{9}$$

where FCs means fully connected layers, σ represents sigmoid function and patch_i denotes i-th partitioned patch of original matrix M.

ResNet (He et al. 2016) has been widely used in computer vision related tasks, such as image classification, object recognition and so on. The skip-connection in ResNet improves the convergence speed of the model during the training process and alleviates the possible degradation of the deep network architecture. Driven by this helpful approach, in the branch of our proposed model, a residual block after PatchAtt was raised to obtain node-level interaction vector by directly connecting the low-dimensional signal to the representation after 3 convolution layers. Unlike the histogram of SimGNN and the explicit node ordering in MGMN and GraphSIM, DeepSIM designs a trainable feature extraction module based on CNNs combined with PatchAtt to implicitly capture the spatial relationships between node embeddings. Finally, the mixed representation will be fed into the pooling layer to get the final output of this branch.

3.3 Stage 3: Prediction based on fully-connected network

Before prediction, the outputs of the above two branches which was described in Sect. 3.2 will be concatenated together. At this stage, fully-connected network with ReLU as activation function was chosen to be a predictive module just like traditional learning-based prediction and classifica-

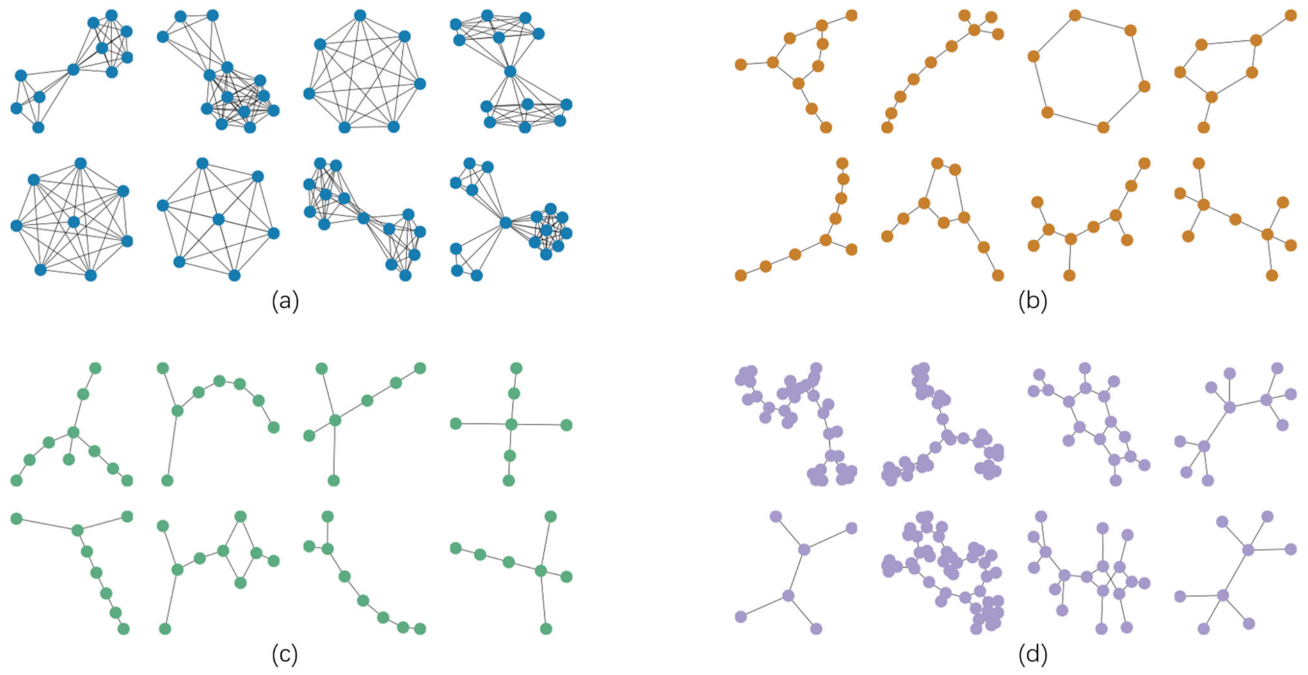


Fig. 7 Visualization of four datasets. **a** IMDB, **b** AIDS, **c** LINUX, **d** PTC_MR

Table 1 Descriptions of datasets

Dataset	Description	Graphs
AIDS	Compounds	700
PTC_MR	Carcinogenic compounds	344
LINUX	Program dependency	1000
IMDB	Actors/Actress ego-network	1500

tion tasks and its output is the final result of our model. This result will be compared with ground-truth value.

4 Experiments and results

4.1 Datasets

In order to test the effectiveness of the proposed method, we evaluated it on four real world datasets and compared with other state-of-the-art methods. The descriptions of the experimental datasets are shown in Fig. 7 and Table 1.

AIDS. AIDS is an antiviral screening dataset for NCI/NIH development and treatment, containing 42,390 chemical compounds. It is a large graphic dataset, which is usually used in the field of graphic similarity search. As with SimGNN, 700 compounds with a number of nodes less than or equal to 10 were selected for the experiment.

PTC_MR. The purpose of PTC (Predictive Toxicology Challenge) dataset (Toivonen et al. 2003) is to model the rodent carcinogenicity of chemical compounds. The PTC_MR dataset is the male rat dataset of PTC, with a total of 344 graphs and PTC_MR dataset was fully used for this experiment.

LINUX. LINUX dataset (Wang et al. 2012) is generated from Program Dependence Graph (PDG) of Linux kernel, which contains 48,747 graphs in total with an average of 45 vertices. Each graph in this dataset represents a function and a node represents a statement and an edge denotes a dependency between two statements. Like SimGNN, 1000 graphs with a number of nodes less than or equal to 10 were randomly selected as experimental data.

IMDB. IMDB dataset (Yanardag et al. 2015) is composed of 1500 ego-networks of actors or actresses. An ego-network is one in which the central node must be connected to other nodes, except for the central node, there is an edge connecting the two vertices if there is a correlation among them. For this dataset, if two people star in the same film, there is an edge connecting two nodes.

4.2 Data preprocessing

The initial datasets only contained graph structure and the corresponding information without pairwise ground-truth values which means the similarity labels should be calculated in this stage. An efficient and reliable method called DF-GED (Abu-Aisheh et al. 2015) is directly used for ground-truth calculation of four datasets.

After generating the labels, there are four generated datasets, each one contains the original information of two graphs with the corresponding metric value. Then, for the sake of testing the generalization ability of the model, 30,000 samples were randomly selected from AIDS and Linux as experimental data respectively. We divided 30,000 samples of each generated dataset into train and test set according to the proportion of 70% and 30%. At the same time, in order to test the performance of the model on the entire dataset, the full small-scale PTC_MR dataset without any selection was applied. On the large-scale IMDB dataset, 100,000 samples were randomly selected from the generated 2.25 million items with labels. In the same way as the previous division, 70% of PTC_MR and IMDB are used as the training set and 30% for model testing. Box-plot for each training set will be shown in appendix.

Before feeding the data into proposed models, the result of DF-GED need to be transformed into similarity scores. According to SimGNN, normalized-GED (nGED) combined with exponential function $\lambda(x) = e^{-x}$ was applied to convert GED values into scores in the range of (0,1]. The calculation formula of similarity score of graph pair is as follows:

$$\text{score}_{(G_1,G_2)} = \lambda\left(\text{nGED}\left(G_1, G_2\right)\right) \tag{10}$$

where $\text{nGED}(G_1, G_2) = \frac{(\text{DF-GED}(G_1,G_2))}{(|G_1|+|G_2|)/2}$, $|G_1|$ and $|G_2|$ represent the number of nodes of G_1 and G_2 respectively.

4.3 Baselines and metrics

Baselines involved in the comparison are all based on GNN, including S-GCN, HS-GCN, SingleLayer, SimGNN, GraphSIM and MGMN. SingleLayer (mentioned in Socher et al. (2013)) is a special case of NTN, which does not consider bi-linear tensor and only uses standard layer to reason about vector relations.

SimGNN pointed out that when the number of nodes is large, Strategy 1 can be used independently to speed up the model training process with good performance, so an ablation experiment is very necessary. In order to compare our proposed Approach 1 (DeepSIM_A1) with Strategy 1 (SimGNN_S1) in SimGNN, the double-branch structure of two models were decomposed to construct the ablation experiment. In other words, compare the reasoning capabilities of NTN and FINTN for graph-level interactive features.

For preventing the possible contingency of a single experiment, we conducted several comparative experiments for these methods on four real world datasets mentioned above to demonstrate the superiority of our proposed approach.

MSE (mean squared error), R-square (R^2), and Spearman's correlation coefficient (τ) are used to describe the overall level of each model, with precision at k (p@k) is used to measure the local performance. At the same time, to describe the stability of the prediction results, the standard deviation (σ_t) is applied in this process, where t denotes the number of experiments. For each model in t experiments on a specific dataset, the minimum and average values of MSE and the mean values of R^2 and p will be recorded, where MSE_{min} is used to compare the best results of the models, MSE_{avg}, $R^2{}_{\text{avg}}$ and p_{avg} are used to describe the average performance of several methods. In multiple experiments, models with high stability perform better when the MSE_{avg} values are close and the p@k values shown are all mean values.

4.4 Experimental setup

Obviously, the node embedding block and the prediction module are the same for the above models, so the parameter settings of these two modules will be shared among all methods. In the experiment, three GCN layers were stacked to build the node embedding module and ReLU was chosen to be the activation function. The output dimensions of each GCN layer are 64, 32, and 16, respectively. For prediction module, we adopted four fully connected layers to get the final output (i.e., predicted similarity score). The first three layers have 32, 16 and 4 neurons, respectively, and the number of nodes in the last layer is 1. For NTN and FINTN, the input and output dimensions were set to 16 invariably to correctly connect the front and rear modules. For the pairwise node comparison strategy in SimGNN, the default settings of previous work were used. For the parameter h in PatchAtt, we set it to 4, which means the node-level matrix was divided into 4 patches. And for each patch, global average pooling combined with two full-connected layers was used for calculating its attention score.

DeepSIM is built by TensorFlow + Keras and all methods were tested on a single machine with an Intel Xeon Silver 4110 CPU, NVIDIA GeForce GTX 1080 and NVIDIA Quadro RTX 5000 GPU. We set the batch size to 128 and used Adam (Kingma et al. 2014) as the optimizer with the initial learning rate to 0.001 and weight decay to 0.0005 during the training process. For each training, epoch is set by default to 5 and test set was used to get the corresponding evaluation metrics.

4.5 Results

The experimental results are shown in Tables 2, 3, 4 and 5. Our proposed two methods (DeepSIM and DeepSIM_A1) have achieved good performance in comparison experiments with good stability and we will show the experimental results and give a brief analysis in this subsection.

Among all the methods involved in the experiment, DeepSIM_A1 based on FINTN achieves a surprising performance compared to previous deep learning methods, which proves that our proposed interaction mechanism is of great benefit for vectors relationship mining in this domain. Note

Table 2 Results of five experiments on AIDS

Method	$\mathrm{MSE}_{\mathrm{min}}(10^{-3})$	$MSE_{avg}(10^{-3})$	R^2_{avg}	τ_{avg}	p@10	p@20	σ_5
S-GCN	6.578	6.663	0.360	0.679	0.334	0.484	0.062
HS-GCN	6.326	6.452	0.377	0.681	0.337	0.491	0.114
SingleLayer	6.476	6.763	0.350	0.670	0.333	0.482	0.347
SimGNN_S1	6.446	6.581	0.368	0.604	0.312	0.466	0.124
DeepSIM_A1	6.323	6.433	0.382	0.683	0.340	0.492	0.068
MGMN	6.341	6.469	0.372	0.680	0.332	0.486	0.075
SimGNN	7.163	7.413	0.288	0.604	0.308	0.457	0.220
GraphSIM	**6.217**	**6.340**	0.390	0.681	0.341	0.494	0.078
DeepSIM	6.370	6.411	0.384	0.674	0.342	0.492	0.040

Bold values indicate the optimal results in the experiment

Table 3 Results of five experiments on LINUX

Method	$\mathrm{MSE}_{\mathrm{min}}(10^{-3})$	$\mathrm{MSE}_{\mathrm{avg}}(10^{-3})$	R^2_{avg}	τ_{avg}	p@10	p@20	σ_5
S-GCN	14.621	16.726	0.660	0.796	0.682	0.721	1.590
HS-GCN	15.911	16.566	0.663	0.809	0.679	0.691	0.560
SingleLayer	13.941	15.433	0.681	0.818	0.677	0.713	1.434
SimGNN_S1	14.861	15.332	0.682	0.854	0.681	0.723	0.567
DeepSIM_A1	13.812	14.132	0.704	0.846	0.706	0.728	0.299
MGMN	14.088	14.221	0.693	0.824	0.697	0.724	0.155
SimGNN	14.324	14.901	0.699	0.831	0.691	0.723	0.416
GraphSIM	13.551	14.897	0.693	0.822	0.708	0.732	0.960
DeepSIM	**11.990**	**12.580**	0.738	0.861	0.713	0.747	0.946

Bold values indicate the optimal results in the experiment

Table 4 Results of five experiments on IMDB

Method	$\mathrm{MSE}_{\mathrm{min}}(10^{-3})$	$\mathrm{MSE}_{\mathrm{avg}}(10^{-3})$	R^2_{avg}	τ_{avg}	p@10	p@20	σ_5
S-GCN	2.751	3.581	0.939	0.932	0.869	0.849	0.773
HS-GCN	2.912	3.074	0.951	0.919	0.874	0.866	0.166
SingleLayer	3.291	3.469	0.942	0.913	0.887	0.856	0.253
SimGNN_S1	2.936	3.249	0.945	0.872	0.857	0.827	0.198
DeepSIM_A1	2.118	2.325	0.965	0.914	0.889	0.879	0.267
MGMN	2.216	2.341	0.962	0.908	0.876	0.863	0.087
SimGNN	2.218	2.637	0.958	0.911	0.868	0.856	0.241
GraphSIM	**1.967**	2.165	0.965	0.915	0.882	0.874	0.161
DeepSIM	2.042	**2.163**	0.966	0.921	0.882	0.859	0.114

Bold values indicate the optimal results in the experiment

that while FINTN and NTN have the same time complexity, our Approach 1 outperforms other single-branch feature inference methods (SimGNN_S1 and SingleLayer) in all comparisons.

In addition to effectiveness, we also focus on the efficiency of the proposed methods. Considering that all the methods involved in the comparison use the same graph convolution module in the process of embedding graph structured data with the number of nodes N, the time complexity of the embedding relation inference module for all methods will be discussed in Table 6.

SimGNN and SimGNN_S1 perform poorly on small dataset (AIDS), probably because the features of small datasets are not obvious enough, which means the number of training sets is also insufficient to make the models adequately trained. Also, on the AIDS dataset, SimGNN's histogram feature supplementation approach did not achieve good results, but instead greatly reduced the performance of its strategy 1 (SimGNN_S1). This phenomenon proves that the untrained supplementation method in node-level branch is not reliable enough. However, GraphSIM achieves better performance compared to other methods, including our

Table 5 Results of five experiments on PTC_MR

Method	$\text{MSE}_{\text{min}}(10^{-3})$	$\text{MSE}_{\text{avg}}(10^{-3})$	R^2_{avg}	τ_{avg}	p@10	p@20	σ_5
S-GCN	2.966	3.147	0.542	0.939	0.657	0.764	0.147
HS-GCN	2.913	2.943	0.569	0.951	0.653	0.758	0.023
SingleLayer	2.641	2.774	0.596	0.940	0.652	0.753	0.133
SimGNN_S1	2.764	2.987	0.564	0.948	0.658	0.761	0.204
DeepSIM_A1	2.642	2.723	0.603	0.911	0.636	0.746	0.068
MGMN	2.712	2.780	0.597	0.932	0.652	0.751	0.051
SimGNN	2.876	2.952	0.568	0.903	0.651	0.754	0.080
GraphSIM	2.622	2.746	0.602	0.914	0.653	0.754	0.077
DeepSIM	**2.534**	**2.672**	0.603	0.937	0.633	0.744	0.107

Bold values indicate the optimal results in the experiment

two proposed models. This result demonstrates the effectiveness of its node ordering and multi-scale structure design and inspires our future work. On the LINUX dataset, DeepSIM significantly achieves optimal and stable performance.

On the IMDB dataset, the p@10 values are greater than the p@20 values for each model. This indicates that the uneven label distribution on this dataset leads to large fluctuations in the local best-fit results for each method. GraphSIM wins in terms of optimal performance, while DeepSIM is better in average performance comparison.

Generally, methods with excellent overall metrics are also better in local metrics comparisons. However, on the PTC_MR dataset, DeepSIM and DeepSIM_A1 achieve good performance in the overall metric comparison, but the local indicator (precision at k) is slightly worse. This result demonstrates that the DeepSIM and DeepSIM_A1 methods prefer to find a balance between easier-to-fit data and hard-to-fit outlier points during the training process to improve the overall model performance while other methods may focus more on the prediction result for a single time.

In terms of efficiency (as in Table 6), the time complexity of all single-branch methods is $O(N)$, i.e., there is no complex interaction between graph pair embeddings in such methods, which are mostly simple vector stitching. In this case, SimGNN only uses the histogram to obtain the direct embedding interaction of graph pairs as an information supplement and its extra time overhead is mainly in the histogram feature generation algorithm (hist). MGMN, GraphSIM and DeepSIM all transform graph pair embeddings into two-dimensional matrices and perform feature extraction by convolution in order to mine potential associations. Such methods achieve better results in experiments, but require high time overhead. Our proposed FINTN (applied in DeepSIM_A1) achieves better predictions in comparison experiments with $O(N)$ time complexity, which proves its effectiveness and good performance.

Table 6 Time complexity comparison of relational reasoning modules

Method	Time complexity
S-GCN	$O(N)$
HS-GCN	$O(N)$
SingleLayer	$O(N)$
SimGNN_S1	$O(N)$
DeepSIM_A1	$O(N)$
MGMN	$O(N^2)$
SimGNN	$O(N)$
GraphSIM	$O(N^2)$
DeepSIM	$O(N^2)$

5 Conclusion and discussion

In this paper, a novel two-branch deep learning model for computing graph similarity is proposed and performs well in comparative experiments. The improved neural tensor network (FINTN) significantly outperforms other reasoning methods in this area.

In several comparisons, DeepSIM_A1 achieves good results with low time complexity, which proves that NTN is inadequate for inference of graph-level embedding relations and also shows that considering distance and direction metrics of vectors on the feature space is beneficial for graph similarity calculation. Thus, we believe that when the scale of dataset or the average number of nodes in the data is large, the node-level method of DeepSIM can be discarded to improve the training speed with a tolerable performance loss.

PatchAtt emphasizes the weight between local blocks and considers the relationship with the whole interaction matrix. The node-level CNN module based on this attention mechanism performs well in the experiments and achieves the best results on some datasets, which proves that considering node-level features is beneficial for similarity computation.However, focusing on fine-grained information introduces additional computational cost and may cause overfitting.

Despite the good results of DeepSIM, there are still many aspects that deserve further study in the future:

- DeepSIM performs well in the similarity comparison task on multiple datasets, but still suffers from slow training due to the high time complexity of node branching. Just like SimGNN, designing a fast and reliable node-level feature extraction module will be an important direction for our future work.
- FINTN excels in the field of similarity calculation, but the order of nodes on the two graphs is not considered, i.e., whether the vector difference and cosine similarity on the feature space are reasonable or not depends entirely on the performance of siamese GCN module. Some deep learning methods have been proposed for graph structure space alignment, which we believe is very useful for graph comparison tasks.
- The node interaction matrix in this study consists of the output of three siamese GCN layers, but a large amount of detail information may be lost in the process (i.e., only the high-dimensional information describing the graph structure is retained), which is detrimental to fine-grained node-level interaction feature extraction. A new interaction matrix generation method to retain more details will be one of the future research directions.

Acknowledgements This work is supported by National Natural Science Foundation of China (62076015).

Funding Open Access funding enabled and organized by CAUL and its Member Institutions This work is supported by National Natural Science Foundation of China (62076015) and REaDI Fund from Massey University.

Data Availability Enquiries about data availability should be directed to the authors.

Declarations

Conflict of interest The authors confirm that the content of this article has no conflicts of interest.

Research involving human and animal participants This article does not contain any studies with human participants or animals performed by any of the authors.

Appendix A

See Figs. 8, 9, 10.

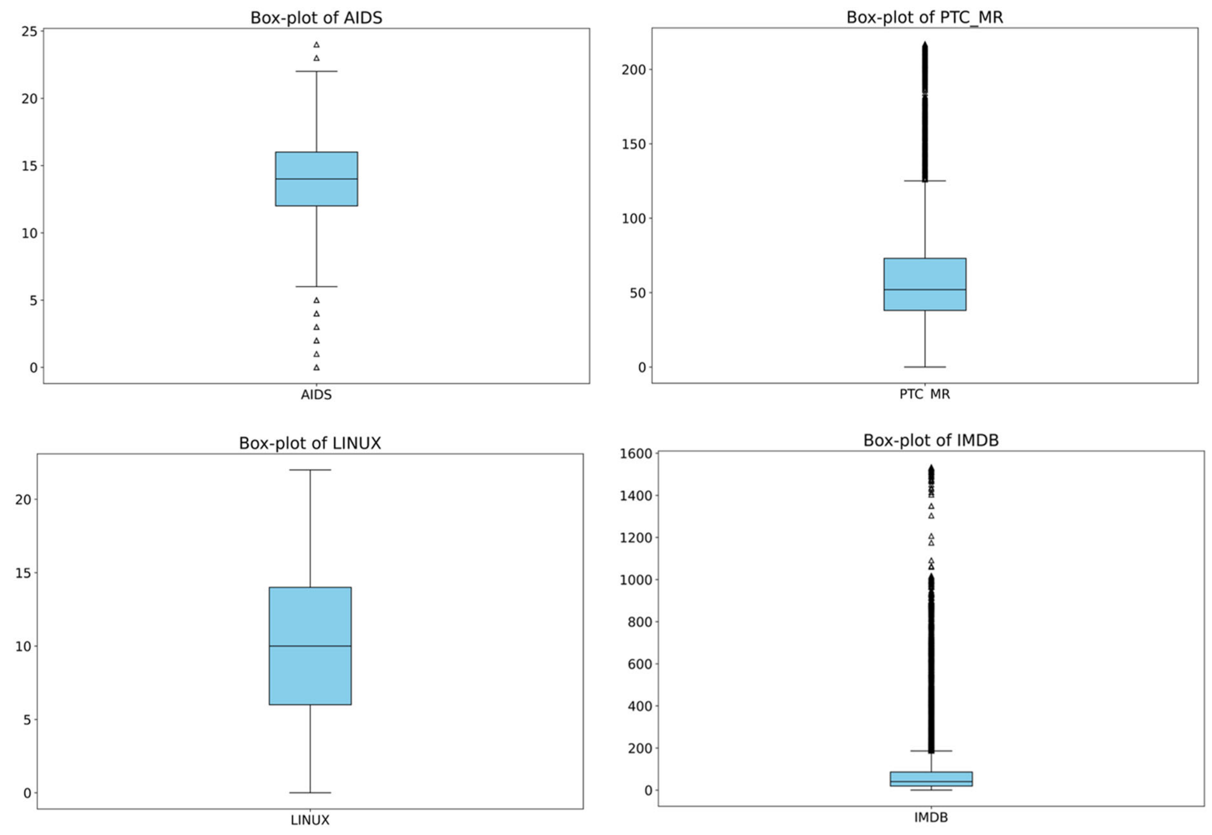


Fig. 8 Box plots of GED values for the four datasets

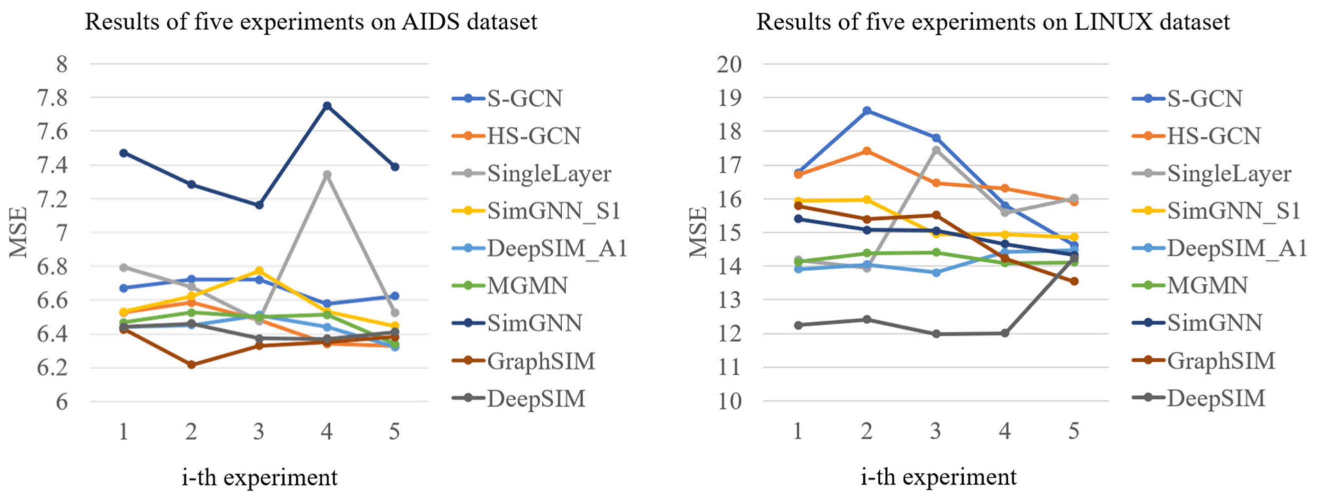
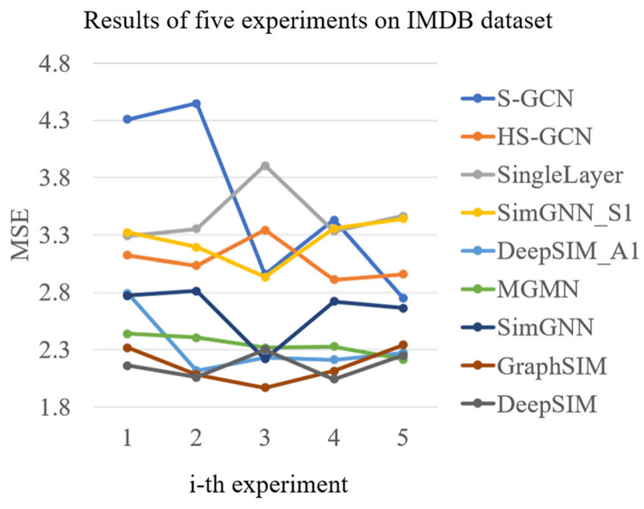


Fig. 9 Result of experiments



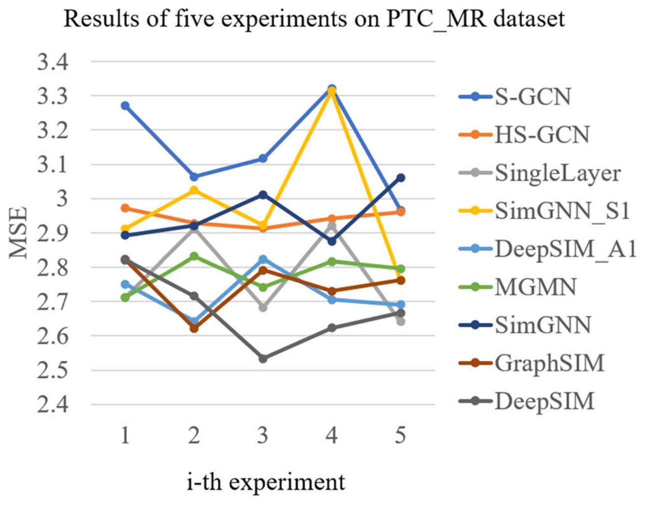


Fig. 10 Result of experiments

References

Abu-Aisheh Z et al (2015) An exact graph edit distance algorithm for solving pattern recognition problems. In: 4th International conference on pattern recognition applications and methods

Bai Y et al (2018) Convolutional set matching for graph similarity. arXiv preprint arXiv:1810.10866

Bai Y et al (2019) Simgnn: a neural network approach to fast graph similarity computation. In: Proceedings of the Twelfth ACM international conference on web search and data mining

Bai Y et al (2020) Learning-based efficient graph similarity computation via multi-scale convolutional set matching. In: Proceedings of the AAAI conference on artificial intelligence, vol 34, No. 04

Bromley J et al (1993) Signature verification using a "siamese" time delay neural network. Int J Pattern Recogn Artif Intell 7(04):669–688

Bronstein MM et al (2017) Geometric deep learning: going beyond Euclidean data. IEEE Signal Process Mag 34(4):18–42

Bunke H, Allermann G (1983) Inexact graph matching for structural pattern recognition. Pattern Recogn Lett 1(4):245–253

Bunke H, Shearer K (1998) A graph distance metric based on the maximal common subgraph. Pattern Recogn Lett 19(3–4):255–259

Chaudhuri U, Banerjee B, Bhattacharya A, Datcu M (2022) Attention-driven graph convolution network for remote sensing image retrieval. IEEE Geosci Remote Sens Lett 19:1–5. https://doi.org/10.1109/LGRS.2021.3105448

Chen X et al (2020) One-shot adversarial attacks on visual tracking with dual attention. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition

Chen F et al (2020) Graph representation learning: a survey. APSIPA Trans Signal Inf Process 9:e15

Chopra S, Hadsell R, LeCun Y (2005) Learning a similarity metric discriminatively, with application to face verification. In: 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05), vol 1. IEEE

Gao X et al (2010) A survey of graph edit distance. Pattern Anal Appl 13(1):113–129

Grover A, Leskovec J (2016) "node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining

Hamilton WL, Ying R, Leskovec J (2017) Inductive representation learning on large graphs. In: Proceedings of the 31st international conference on neural information processing systems

He K et al (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition

Hu J, Shen L, Sun G (2018) Squeeze-and-excitation networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition

Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980

Kipf TN, Welling M (2016) Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907

Koch G, Zemel R, Salakhutdinov R (2015) Siamese neural networks for one-shot image recognition. ICML deep learning workshop, vol 2

Kriege NM, Johansson FD, Morris C (2020) A survey on graph kernels. Appl Netw Sci 5(1):1–42

Ktena SI et al (2017) Distance metric learning using graph convolutional networks: application to functional brain networks. In: International conference on medical image computing and computer-assisted intervention. Springer, Cham

Ktena SI et al (2018) Metric learning with spectral graph convolutions on brain connectivity networks.". NeuroImage 169:431–442

Lagunas E et al (2018) Graph similarity based on graph Fourier distances. In: 2018 26th European signal processing conference (EUSIPCO). IEEE

Le Quoc MT (2014) Distributed representations of sentences and documents. In: International conference on machine learning, PMLR

Ling X et al (2020) Multi-level graph matching networks for deep and robust graph similarity learning

Ling X, Wu L, Wu C, Ji S (2022) Graph neural networks: graph matching. In: Wu L, Cui P, Pei J, Zhao L (eds) Graph Neural Networks: Foundations, Frontiers, and Applications. Springer, Singapore. https://doi.org/10.1007/978-981-16-6054-2_13

Ma G et al (2019) Deep graph similarity learning for brain data analysis. In: Proceedings of the 28th ACM international conference on information and knowledge management

Ma G et al (2021) Deep graph similarity learning: a survey. Data Min Knowl Discov 35:1–38

Narayanan A et al (2017) graph2vec: learning distributed representations of graphs. arXiv preprint arXiv:1707.05005 (2017)

Neuhaus M, Bunke H (2006) Edit distance-based kernel functions for structural pattern classification. Pattern Recogn 39(10):1852–1863

Nikolentzos G, Meladianos P, Vazirgiannis M (2017) Matching node embeddings for graph similarity. In: Thirty-first AAAI conference on artificial intelligence

Ok S (2020) A graph similarity for deep learning. Adv Neural Inf Process Syst 33:1–12

Ortega A et al (2018) Graph signal processing: overview, challenges, and applications. Proc IEEE 106(5):808–828

Papadimitriou P, Dasdan A, Garcia-Molina H (2010) Web graph similarity for anomaly detection. J Internet Serv Appl 1(1):19–30

Perozzi B, Al-Rfou R, Skiena S (2014) Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining

Qin Z, Bai Y, Sun Y (2020) Ghashing: semantic graph hashing for approximate similarity search in graph databases. In: Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining

Riesen K, Bunke H (2009) Approximate graph edit distance computation by means of bipartite graph matching. Image Vis Comput 27(7):950–959

Shervashidze N et al (2011) Weisfeiler-lehman graph kernels. J Mach Learn Res 12(9)

Socher R et al (2013) Reasoning with neural tensor networks for knowledge base completion. Adv Neural Inf Process Syst

Thomas G, Peter F, Stefan W (2003) On graph kernels: hardness results and efficient alternatives. Learning theory and kernel machines. Springer, Berlin, pp 129–143

Tixier AJ-P et al (2019) Graph classification with 2d convolutional neural networks. In: International conference on artificial neural networks. Springer, Cham

Toivonen H et al (2003) Statistical evaluation of the predictive toxicology challenge 2000–2001. Bioinformatics 19(10):1183–1193

Veličković P et al (2017) Graph attention networks. arXiv preprint arXiv:1710.10903

Wang X et al (2012) An efficient graph indexing method. In: 2012 IEEE 28th international conference on data engineering. IEEE

Weisfeiler B, Leman A (1968) The reduction of a graph to canonical form and the algebra which appears therein. NTI Ser 2(9):12–16

Wu Z et al (2020) A comprehensive survey on graph neural networks. IEEE Trans Neural Netw Learn Syst 32(1):4–24

Xu Zhou-bo ZK et al (2018) Summary of graph edit distance. Comput Sci

Yanardag P, Vishwanathan SVN (2015) Deep graph kernels. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining

Zagoruyko S, Komodakis N (2015) Learning to compare image patches via convolutional neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition

Zaslavskiy M, Bach F, Vert J-P (2009) Global alignment of protein-protein interaction networks by graph matching methods. Bioinformatics 25(12):i259-1267

Zeng Z et al (2009) Comparing stars: on approximating graph edit distance. Proc VLDB Endow 2(1):25–36

Zhang S et al (2019) Graph convolutional networks: a comprehensive review. Comput Soc Netw 6(1):1–23