Hierarchical Large-scale Graph Similarity Computation via Graph Coarsening and Matching

Haoyan Xu[1]*, Runjian Chen[1]*, Yunsheng Bai[2]*
Ziheng Duan[1], Jie Feng[1], Ke Luo[3], Yizhou Sun[2], Wei Wang[2]†
[1]College of Control Science and Engineering, Zhejiang University
[2]Department of Computer Science, University of California, Los Angeles
[3]College of Life Sciences, Zhejiang University
haoyanxu@zju.edu.cn, rjchen@zju.edu.cn, yba@cs.ucla.edu
{duanziheng, zjucse_fj, luoke}@zju.edu.cn, {yzsun, weiwang}@cs.ucla.edu

Abstract

In this work, we focus on large graph similarity computation problem and propose a novel "embedding-coarsening-matching" learning framework, which outperforms state-of-the-art methods in this task and has significant improvement in time efficiency. Graph similarity computation for metrics such as Graph Edit Distance (GED) is typically NP-hard, and existing heuristics-based algorithms usually achieves a unsatisfactory trade-off between accuracy and efficiency. Recently the development of deep learning techniques provides a promising solution for this problem by a data-driven approach which trains a network to encode graphs to their own feature vectors and computes similarity based on feature vectors. These deep-learning methods can be classified to two categories, embedding models and matching models. Embedding models such as GCN-MEAN and GCN-MAX, which directly map graphs to respective feature vectors, run faster but the performance is usually poor due to the lack of interactions across graphs. Matching models such as GMN, whose encoding process involves interaction across the two graphs, are more accurate but interaction between whole graphs brings a significant increase in time consumption (at least quadratic time complexity over number of nodes). Inspired by large biological molecular identification where the whole molecular is first mapped to functional groups and then identified based on these functional groups, our "embedding-coarsening-matching" learning framework first embeds and coarsens large graphs to coarsened graphs with denser local topology and then matching mechanism is deployed on the coarsened graphs for the final similarity scores. Detailed experiments on both synthetic and real datasets have been conducted and the results demonstrate the efficiency and effectiveness of our proposed framework in both similarity regression and classification tasks.

Introduction

With flexible representative abilities, graphs have a broad range of applications in various fields, including social network study, computational chemistry (Gilmer et al. 2017), and biomedical image analysis (Ktena et al. 2017). Recently, especially with the development of deep learning techniques, there aroused surging interests on graph-related problems. Here in this paper, we focus on large graph similarity prediction problem, which is one of the fundamental challenges and appears in various world applications including biological molecular similarity search (Kriegel, Pfeifle, and Schönauer 2004)(Tian et al. 2007) and social group network similarity identification(Steinhaeuser and Chawla 2008)(Ogaard et al. 2013).

Traditionally, there are various evaluation metrics including Graph Edit Distance (GED) (Sanfeliu and Fu 1983) developed for graph similarity computation problem. Existing algorithms for these metrics can be divided into two classes. The first one includes algorithms like (Riesen, Emmenegger, and Bunke 2013) and (McCreesh, Prosser, and Trimble 2017) that calculate the exact values. While the exact similarity scores for sure help us better understand the relationship between graphs, it is indeed an NP-hard problem and requires exponential time complexity in the worst case. The second class, which includes algorithms such as (Neuhaus, Riesen, and Bunke 2006), (Jonker and Volgenant 1987), (Fankhauser, Riesen, and Bunke 2011), (Kuhn 1955) and (Riesen and Bunke 2009), only computes the approximate values and saves time in return. However, these algorithms still run with polynomial or even sub-exponential time complexity.

To date, graph neural networks (GNNs) have been proved to be a potential data-driven solution for graph-related tasks with relatively high accuracy and far lower time cost in inference. Many graph convolution layers, such as Graph Convolution Network (GCN) (Defferrard, Bresson, and Vandergheynst 2016), Edge Convolution Network (EDGECONV) (Wang et al. 2019) and Graph Isomorphism Network (GIN) (Xu et al. 2018), are proposed and demonstrated powerful in embedding node features with original labels and local topology. Also, as the structure of graph is quite different from that of image, advanced pooling techniques to downsample the graphs are developed, including TOPKPOOLING (Gao and Ji 2019), SAGPOOL (Lee, Lee, and Kang 2019) and MEMPOOL (Khasahmadi et al. 2020).

When we talk about graph similarity learning problem, there always involves two stages: (1) embedding, which maps each graph to its representation feature vector and

*Equal contribution with order determined by rolling the dice.

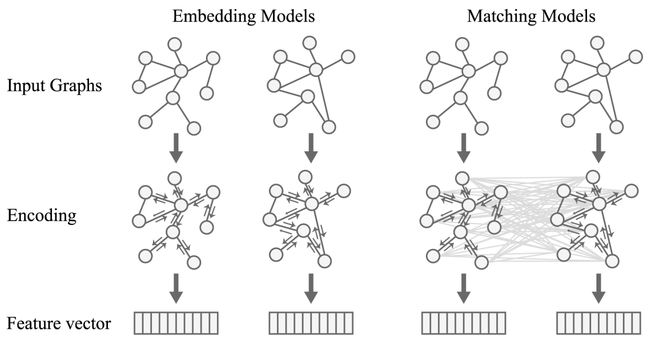

Figure 1: Two classes of existing models

makes similar graph close in that feature space (2) similarity computation based on these feature vectors. Existing deep learning techniques for graph similarity computation problem can be clearly classified into two categories with the different methods in first stage, as illustrated in fig. 1. The first category, exemplified by HIERARCHICALLY MEAN (GCN-MEAN) and HIERARCHICAL MAX (GCN-MAX) (Defferrard, Bresson, and Vandergheynst 2016), directly maps graphs to feature vectors by hierarchically coarsening the graphs while the second category embraced by GRAPH MATCHING NETWORKS (GMN) (Li et al. 2019) embeds pair of graphs at the same time with a cross-graph matching mechanism. Other techniques like GSIMCNN (Bai et al. 2018b) and GRAPHSIM (Bai et al.) are similar to the second category except that they do not generate two separate feature vectors but instead directly embed graph pair to similarity score with cross graph mechanism. Meanwhile SIMGNN (Bai et al. 2018a) deploy the two categories above parallelly. For simplicity, in the remaining content of this paper we will call the first category embedding models and the second matching models.

On the one hand, embedding models can be quite faster than matching models at inference stage. We here take similarity search as an example where we have K graphs in the database and we want to compute the similarity between a new graph with all the graphs in the database. Because embedding models can embed all the graphs in the database to feature vectors in advance, these models only need to forward the new graph to its feature vector and compute similarity K times based on feature vectors. Meanwhile, when there comes a new graph, matching models have to forward it respectively with all the graphs in the database (across graph mechanism can not be finished in advance), which is absolutely time-consuming. Details about time complexity analysis will be shown in Section 3. On the other hand, as the feature vectors generated by embedding models do not involve interaction between graphs, the performance of embedding models are far worse than matching models.

Let's consider the process of biological or chemical molecules identification, we always map them to molecular groups and evaluate their similarity based on these groups with denser local topology instead of directly identifying the whole molecules. Inspired this process as well as the success of pooling methods on graph classification problem and matching mechanism on graph similarity prediction problem, we propose our "embedding-coarsening-matching" graph similarity computation framework COSIM-GNN, i.e., *Co*arsening-based *Sim*ilarity Computation via *G*raph *N*eural *N*etworks. As illustrated in fig. 2, graph embedding layers and pooling layers are first applied to encode and coarsen the graphs and then matching mechanism is deployed on the coarsened graph pair to compute similarity between graphs. In the first two stages of our model, there is no interaction across graphs so this part can be finished in advance in inference time or similarity search problem in order to save time. Meanwhile the intra-attention brought by coarsening and inter-attention introduced in matching part guarantee the performance, which is even better than to directly apply the matching models.

Many existing pooling methods can be incorporated in our framework and achieve relatively good performance. However, we notice that these state-of-the-art pooling mechanisms always involve a process of generating centroids for pooling but the generation process of centroids never rely on the input graphs, which does not make sense intuitively because the centroids for different graph should be different. Thus we propose a novel pooling layer **ADAPTIVE POOLING**. The generation process of centroids in ADAPTIVE POOLING is **input-related** and still keep the property of **permutation invariance**, leading to better performance than state-of-the-art pooling techniques. We highlight our main contributions as follows:

- We propose a novel framework, which first hierarchically encodes and coarsens graphs and then deploys matching mechanism on the coarsened graph pairs, to address the challenging problem of similarity computation between large graphs.
- We propose a novel pooling layer ADAPTIVE POOLING. The generation of centroids in this layer is based on the input graph, which leads to better performance while maintaining permutation invariance.
- Our framework shows significant improvement in time complexity as compared to matching models and outperforms matching models (thus far better than embedding models) on both similarity regression and classification problems.
- We conduct extensive experiments on both real graph datasets and synthetic datasets consisted of large graphs, which will be detailed in Section 4 to demonstrate the scalability, effectiveness and efficiency of our proposed framework.
- Our framework is able to learn from graph pairs with relatively small number of nodes (approximate 100) and then be deployed to infer similarity between very large graphs with thousands of nodes.

Proposed Framework

In order to reduce the high time consumption brought by interaction across whole large graphs and take the advantage of the combination of intra- and inter- attention mechanism, we propose our novel COSIM-GNN framework, as shown in Figure 2. In this section, we respectively introduce how each stage works in details. We use bold font for matrices and tilt font for function in this section. The right superscript of a matrix stands for graph number indicator and the right subscript stands for the stage of the matrix. For example, $\mathbf{X}_{in}^{(1)}$ means the node feature matrix for the first input graph and $\mathbf{A}_{encode}^{(2)}$ means the adjacency matrices for the second graph after encoding.

Starting with two input node sets denoted as $\mathbf{X}_{in}^{(1)} \in \mathbb{R}^{n_{in}^{(1)} \times d_{in}^{(1)}}$ and $\mathbf{X}_{in}^{(2)} \in \mathbb{R}^{n_{in}^{(2)} \times d_{in}^{(2)}}$ as well as the adjacency matrix $\mathbf{A}_{in}^{(1)} \in \mathbb{R}^{n_{in}^{(1)} \times n_{in}^{(1)}}$ and $\mathbf{A}_{in}^{(2)} \in \mathbb{R}^{n_{in}^{(2)} \times n_{in}^{(2)}}$, we in each part first give the overall equation on how the stage works and then make detailed explanations on the overall equation.

Encoding

In the first stage, we employ k encoding layers $(F_1, F_2, ... F_k)$ on the two input graphs respectively for embedding feature of nodes and transform the feature dimension of nodes to d_{encode}, just as shown in (1) below:

$$\mathbf{X}_{encode}^{(i)}, \mathbf{A}_{encode}^{(i)} = F_k(F_{k-1}(...F_1(\mathbf{X}_{in}^{(i)}, \mathbf{A}_{in}^{(i)}))) \quad (1)$$

where $i = 1, 2$, $\mathbf{X}_{encode}^{(i)} \in \mathbb{R}^{n_{in}^{(i)} \times d_{encode}}$, $\mathbf{A}_{encode}^{(i)} \in \mathbb{R}^{n_{in}^{(i)} \times n_{in}^{(i)}}$. (2) shows that F_j $(j = 1, 2, ..., k)$ is consisted of a graph convolution layer GC, a non-linear activation σ and a batchnorm layer bn.

$$F(\mathbf{X}, \mathbf{A}) = bn\left(\sigma(GC(\mathbf{X}, \mathbf{A}))\right) \quad (2)$$

We use ReLU for σ here and graph embedding layers GC can be GCN (Defferrard, Bresson, and Vandergheynst 2016), GAT(Veličković et al. 2017) or GIN (Xu et al. 2018).

Coarsening

What this stage does is to pool the encoded graphs, $\mathbf{X}_{encode}^{(i)}$ and $\mathbf{A}_{encode}^{(i)}$, to coarsened graphs, $\mathbf{X}_{pool}^{(i)} \in \mathbb{R}^{n_{pool} \times d_{pool}}$ and $\mathbf{A}_{pool}^{(i)} \in \mathbb{R}^{n_{pool} \times n_{pool}}$. The overall transformation is shown in equation 3, where σ is a non-linear activation function, $\mathbf{W} \in \mathbb{R}^{d_{encode} \times d_{pool}}$ is a trainable parameter matrix standing for a linear transformation and $\mathbf{C}^{(i)} \in \mathbb{R}^{n_{pool} \times n_{in}^{(i)}}$ is an assignment matrix representing a projection from the original node number to pooled node number. As $\mathbf{C}^{(i)}$ assigns weights for nodes in the input graph to nodes in the coarsened graph, it indeed stands for an intra-attention mechanism.

$$\begin{aligned} \mathbf{X}_{pool}^{(i)} &= \sigma\left(\mathbf{C}^{(i)} \mathbf{X}_{encode}^{(i)} \mathbf{W}\right) \\ \mathbf{A}_{pool}^{(i)} &= \sigma\left(\mathbf{C}^{(i)} \mathbf{A}_{encode} (\mathbf{C}^{(i)})^{\mathsf{T}}\right) \end{aligned} \quad (3)$$

Different approaches are adopted to compute the assignment matrix $\mathbf{C}^i$ in different pooling layers and when we look at one of the most representative pooling layer so far, MEMPOOL (Khasahmadi et al. 2020), we notice that it generates memory heads, which stands for the new centroids in the space of pooled graphs, without involvement of the input graph. However, intuitively, the new centroids for different graphs should be different. Thus here we propose a new method: ADAPTIVE POOLING, to calculate $\mathbf{C}$. We first generate h batches of centroids $\mathbf{K}^i \in \mathbb{R}^{h \times n_{pool} \times d_{encode}}$ based on the encoded graph (**permutation invariance** remains) and then compute and aggregate the relationship between every batch of centroids and the encoded graph, leading to the final assignment matrix $\mathbf{C}$. Detailed ablation study about pooling layer is conducted in Section 4, which demonstrates the effectiveness of ADAPTIVE POOLING.

As shown in equation 4, an average aggregation F_{avg} over the encoded graph is deployed, transforming $\mathbf{X}_{encode} \in \mathbb{R}^{n_{in} \times d_{encode}}$ to $\mathbf{X}_{avg} \in \mathbb{R}^{1 \times d_{encode}}$. Then an Multiple Layer Perceptron (MLP) is applied to map $\mathbf{X}_{avg}$ to $\mathbf{K} \in \mathbb{R}^{(h \times n_{pool}) \times d_{encode}}$, after which $\mathbf{K}$ is reshaped to $\mathbb{R}^{h \times n_{pool} \times d_{encode}}$. Note that the generation of batches of centroids here is dependent on the encoded graph (input of this layer), which makes sense because centroids for different input graphs in training and testing set should be different. Besides, due to the mean aggregation we apply here, we keep the property of permutation invariance in our proposed pooling method, one of the most important properties for graph-related deep learning architecture.

$$\mathbf{K} = MLP\left(F_{avg}\left(X_{encode}\right)\right) \quad (4)$$

Then we compute the relationship $\mathbf{C}_p \in \mathbb{R}^{n_{pool} \times n_{in}}$ $(p = 1, 2, ..., h)$ between every batch of centroids $\mathbf{K}_p \in \mathbb{R}^{n_{pool} \times d_{encode}}$ $(p = 1, 2, ..., h)$ and $\mathbf{X}_{encode}^i \in \mathbb{R}^{n_{encode} \times d_{encode}}$. We empirically find that a cosine similarity leads to satisfactory pooling performance, as described in equation 5, where a row normalization is deployed in the resulting similarity matrix.

$$\begin{aligned} \mathbf{C}_p &= \text{cosine}\left(\mathbf{K}_p, \mathbf{X}_{encode}\right) \\ \mathbf{C}_p &= \text{normalize}\left(\mathbf{C}_p\right) \end{aligned} \quad (5)$$

We finally aggregate the information of h relationship $\mathbf{C}_p \in \mathbb{R}^{n_{pool} \times n_{in}}$. In (6), we concatenate $\mathbf{C}_p$ $(p = 1, 2, ..., h)$ and perform a trainable weighted sum Γ_ϕ to the concatenated matrix, leading to our assignment matrix $\mathbf{C}$.

$$\mathbf{C} = \Gamma_\phi \left(\mathop{\Big\|}_{p=0}^{|h|} \mathbf{C}_p \right) \quad (6)$$

Matching

As shown in (7), this part aims to compute similarity based on the interaction of two coarsened graphs. f_{match} takes the two coarsened graphs as input and generates feature vectors $\mathbf{X}_{final}^{(i)}$ for respective graph. A cosine similarity is deployed for the similarity score. Finally we compute the Mean Squared Loss between the score and the ground truth

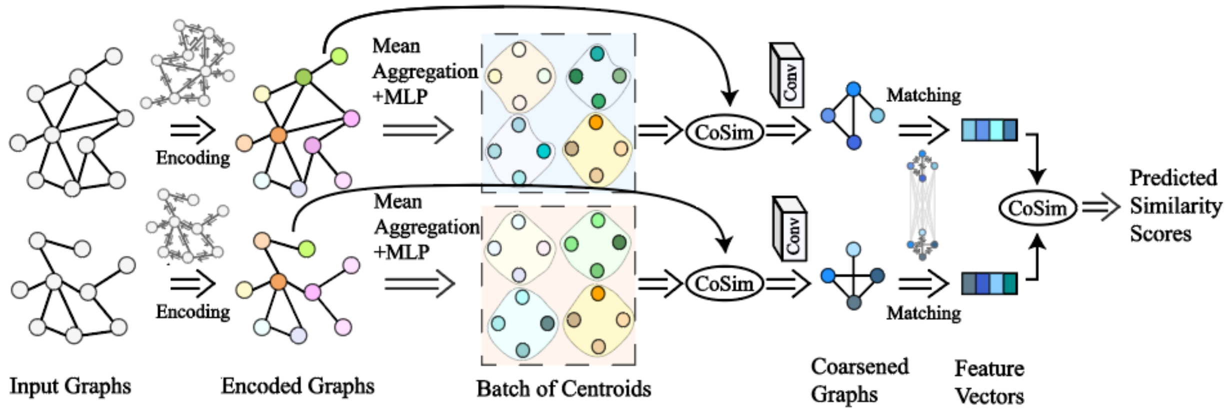

Figure 2: An overview of our "embedding-coarsening-matching" framework CoSim-GNN. Encoding part aggregates features respectively inside the two graphs. Then coarsening stage transforms the encoded graphs to batches of centroids and these centroids jointly coarsen the graphs. Finally matching-based feature aggregation is deployed on the coarsened graph pair and similarity score is computed based on the two graph-level feature vectors.

similarity GT for backward update.

$$\begin{aligned}\mathbf{X}_{final}^{(i)} &= f_{match}\left(\mathbf{X}_{pool}^{(i)}, \mathbf{A}_{pool}^{(i)}, \mathbf{X}_{pool}^{(j)}, \mathbf{A}_{pool}^{(j)}\right) \\ \textbf{Score} &= cosine\left(\mathbf{X}_{final}^{(1)}, \mathbf{X}_{final}^{(2)}\right) \\ \textbf{Loss} &= \frac{1}{batchsize}\sum_{i=1}^{batchsize}\left(Score(i) - GT(i)\right)^2\end{aligned} \tag{7}$$

Several matching mechanisms such as Graph Matching Network (Li et al. 2019) and GSimCNN (Bai et al. 2018b), can be deployed for f_{match} and here we give a simple example. As shown in (8), there are three propagators f_P and one aggregator f_A to transform the pooled graphs to respective feature vector.

$$\mathbf{X}_{final}^{(i)} = f_A\left(f_P\left(\mathbf{X}_{pool}^{(i)}, \mathbf{A}_{pool}^{(i)}, \mathbf{X}_{pool}^{(j)}, \mathbf{A}_{pool}^{(j)}\right)\right) \tag{8}$$

In the propagators f_P, we aggregate both inside features ($\mathbf{I}^{(i)}$) and external features ($\mathbf{M}^{(i)}$). As shown in (9), we first propagate features inside respective graph with a GAT (Veličković et al. 2017) and add up the features for all the neighbors of every node to form $\mathbf{I}^{(i)} \in \mathbb{R}^{n_{pool} \times d_{pool}}$.

$$\begin{aligned}\mathbf{X}_{gat}^{(i)} &= \mathrm{GAT}\left(X^{(i)}, A^{(i)}\right) \\ \mathbf{I}^{(i)}(k) &= \sum_{\forall j,\ A^{(i)}(k,j)=1} \mathbf{X}_{gat}^{(i)}(j), \quad k = 1, 2, ..., n_{pool}\end{aligned} \tag{9}$$

As for the inter-attention, we compute $\mathbf{M}^{(i)} \in \mathbb{R}^{n_{pool} \times d_{pool}}$ with f_{cross} in (10). First a relationship mask is generated across the graph pair, which involves matrix multiply for the normalized graphs and a softmax activation (σ). Then we apply the mask to the graph $\mathbf{X}^j$ and subtract $\mathbf{X}^i$ with the masked $\mathbf{X}^j$.

$$\begin{aligned}\mathbf{M}^{(i)} &= f_{cross}\left(\mathbf{X}^{(i)}, \mathbf{X}^{(j)}\right) \\ &= \mathbf{X}^{(i)} - \sigma\left(\frac{\mathbf{X}^{(i)}}{\|\mathbf{X}^{(i)}\|_{L2}} \cdot \frac{(\mathbf{X}^{(j)})^{\mathsf{T}}}{\|\mathbf{X}^{(j)}\|_{L2}}\right) \cdot \mathbf{X}^{(j)}\end{aligned} \tag{10}$$

After the above two steps, we apply f_{node} to embrace all these features including the original input by concatenating all these features and deploying an MLP on the concatenated matrix.

$$\begin{aligned}\mathbf{X}_P^{(i)} &= f_{\text{node}}\left(\mathbf{X}_{pool}^{(i)}, \mathbf{I}^{(i)}, \mathbf{M}^{(i)}\right) \\ &= MLP\left(\|\left(\mathbf{X}_{pool}^{(i)}, \mathbf{I}^{(i)}, \mathbf{M}^{(i)}\right)\right)\end{aligned} \tag{11}$$

As for the aggregator, we use the following module proposed in (Li et al., 2015), where L_G, L_{gate} and L are simply implemented with Multiple Layer Perceptron and the nonlinear activation σ is the softmax function.

$$\begin{aligned}\mathbf{X}_{final}^{(i)} &= f_A\left(\mathbf{X}_P^{(i)}\right) \\ &= \mathrm{L}_G\left(\sigma\left(\mathrm{L}_{\text{gate}}\left(\mathbf{X}_P^{(i)}\right)\right) \cdot \mathrm{L}\left(\mathbf{X}_P^{(i)}\right)\right)\end{aligned} \tag{12}$$

Time Complexity Analysis in Graph Similarity Search Problem

For a pair of input graphs $\mathbf{X}_{in}^{(1)} \in \mathbb{R}^{n_{in}^{(1)} \times d_{in}^{(1)}}$ and $\mathbf{X}_{in}^{(2)} \in \mathbb{R}^{n_{in}^{(2)} \times d_{in}^{(2)}}$, we can assume that they separately have $m^{(1)}$, $m^{(2)}$ edges and the final embedded feature vectors for both graphs are $\mathbf{X}_{final}^{(1)} \in \mathbb{R}^{d_{final}}$ and $\mathbf{X}_{final}^{(2)} \in \mathbb{R}^{d_{final}}$. Then

Table 1: Time complexity comparison in similarity search problem.

Categories	Time Complexity
Embedding models	$O\left(2 \times m + K \times d_{final}\right)$
Matching models	$O\left(\left(2 \times m \times 2 + n \times n + d_{final}\right) \times K\right)$
CoSim-GNN	$O\left(2 \times m + n \times n_{pool} + \left(2 \times m_{pool} \times 2 + n_{pool} \times n_{pool} + d_{final}\right) \times K\right)$

we can analyse the time complexity over $n_{in}^{(i)}$, $m^{(i)}$ and d_{final}. Note that due to the fact that there exists a lot of variance for each model, we here analyse the simplest cases for each category and the real time consumption will be presented later in the Section 4. Table 1 shows the time complexity comparison for embedding models, matching models and our proposed framework, which will be discussed in details in the three parts below. In our settings, n_{pool} is far less than n, thus our framework costs far less time than matching models do, especially when the node number of the graphs is very large.

Embedding models

Considering the simplest case here for embedding models, we only visit every edge once and deploy two computational operational on the two nodes it connect, which contributes to the feature of local topology. Thus the computation complexity for these cases is $O\left(2 \times m^{(i)}\right),\ i = 1, 2$.

Matching models

Assuming the simplest case here for matching models, we first compute the relationship across $\mathbf{X}_{in}^{(1)}$ and $\mathbf{X}_{in}^{(2)}$. This part involves $n_{in}^{(1)} \times n_{in}^{(2)}$ computational operations because we have to calculate the connection between every node in $\mathbf{X}_{in}^{(1)}$ to all nodes in $\mathbf{X}_{in}^{(2)}$. Then for each input graph, we also visit every edge once and deploy two computational operational on the two nodes it connect, which also makes contribution to the feature of local topology. Thus the computation complexity for these cases is $O\left(2 \times m^{(i)} + n_{in}^{(1)} \times n_{in}^{(2)}\right),\ i = 1, 2$.

Our framework

For our framework, we take the denotation in Section 2, that is the number of nodes in coarsened graphs is n_{pool}. In the embedding stage, we similarly have to visit every edge once and compute twice. Then in the pooling part, we make an transformation from $n_{in}^{(i)}$ dimensional space to n_{pool} dimensional space, which costs us $n_{in}^{(i)} \times n_{pool}$ computational operations. Finally in the matching stage, like what have been analysed in the previous section, we need $2 \times m_{pool}^{(i)} + n_{pool} \times n_{pool},\ i = 1, 2$ operations. Thus the resulting complexity is $O\left(2 \times m^{(i)} + n_{in}^{(i)} \times n_{pool} + 2 \times m_{pool}^{(i)} + n_{pool} \times n_{pool}\right),\ i = 1, 2$.

Experiments and Results

Dataset Information

In most of the existing graph datasets, such as AIDS and LINUX, the number of nodes is relatively small in each graph. Thus the characteristics of the entire graphs can be easily characterized and there is no need to coarsen the graphs for better similarity computation results. From this point of view, we here use two types of datasets for evaluating the performance of our framework. The first one is relatively large graphs (with 15 or more nodes) in IMDB dataset. The second one is synthetic graphs with more nodes. We randomly divide each dataset to three sub-sets containing 60%, 20% and 20% of all graphs for training, validating and testing.

Processing IMDB Dataset In this paper, we focus on similarity computation of large graphs, so we filter the original IMDB dataset(Yanardag and Vishwanathan 2015) and choose all the graphs that have 15 or more nodes. The new dataset is called **IMDB-L**.

Synthetic Dataset Information To generate an synthetic dataset, we need to generate graphs and give ground truth similarity for graph pairs. A small number of basic graphs is first generated and then we prune the basic graphs to generate derived graphs with two different categories of pruning rules: BarabsiAlbert preferential attachment model (BA model) (Jeong, Néda, and Barabási 2003) and Erds-Rnyi graph (ER model) (Erdos 1959)(Bollobás and Béla 2001). Details about why we adopt this process and how these two models work will be presented in appendix B. We generate 4 datasets: **BA-60**, **BA-100**, **BA-200** and **ER-100**, where the first two characters stand for generation rule and behind is the number of nodes in the basic graphs for that dataset.

As for the ground truth calculation, A* algorithm cannot be used to calculate the ged distance because of the large number of nodes in our dataset. We propose to use the minimum among four evaluation indicators: the three values calculate respectively by HUNGARIAN,VJ, and BEAM and the GED value we obtain while generating the graph. Then we convert this minimum indicator to the similarity score with a normalization of the minimums followed by a exponential function, resulting in a value among $[0, 1]$. The details about the ground truth generation can be found in appendix C.

Experiment Settings

Evaluation Metrics We apply six metrics to evaluate all the models: *TIME*, *Mean Squared Error (MSE)*, *Mean Absolute Error (MAE)*, *Spearman's Rank Correlation Coefficient (ρ)* (Spearman 1961), *Kendall's Rank Correlation Coefficient (τ)* (Kendall 1938) and *Precision at k (p@k)*. *TIME*

Table 2: Setting of different models in our framework.

Model Names	**Embedding**	**Coarsening**	**Matching**
CoSim-ATT	GIN	SimAtt (Bai et al. 2018a) ($n_{pool} = 1$)	OurMatch
CoSim-CNN	GIN	AdaptivePool ($n_{pool} = 10$)	GSimCNN (Bai et al. 2018b)
CoSim-SAG	GIN	SAGPool (Lee, Lee, and Kang 2019) ($n_{pool} = 10$)	OurMatch
CoSim-TopK	GIN	TopKPool (Gao and Ji 2019) ($n_{pool} = 10$)	OurMatch
CoSim-Mem	GIN	MemPool (Khasahmadi et al. 2020) ($n_{pool} = 10$)	OurMatch
CoSim-GNN10	GIN	AdaptivePool ($n_{pool} = 10$)	OurMatch
CoSim-GNN1	GIN	AdaptivePool ($n_{pool} = 1$)	OurMatch

Table 3: Results for *MSE* and *MAE* in 10^{-3}. The three traditional method are involved in the ground truth computation and thus these values are labeled with superscript *.

Method	**BA-60**		**BA-100**		**BA-200**		**ER-100**		**IMDB-L**	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Hungarian	186.19*	332.20*	205.38*	343.83*	259.06*	379.38*	236.15*	421.66*	2.67*	16.60*
vj	258.73*	294.83*	273.94*	404.61*	314.45*	426.86*	275.22*	463.53*	7.68*	22.73*
Beam	59.14*	129.26*	114.02*	206.76*	186.03*	287.88*	104.73*	226.42*	0.39*	3.93*
GCN-Mean	5.85	53.92	12.53	90.88	23.66	127.82	16.58	92.98	22.17	55.35
GCN-Max	13.66	91.38	3.14	42.24	22.77	107.58	79.09	211.07	47.14	123.16
SimGNN	8.57	63.80	6.23	47.80	3.06	32.77	6.37	45.30	7.42	33.74
GSimCNN	5.97	56.05	1.86	30.18	2.35	32.64	2.93	34.41	5.01	30.43
GMN	2.82	38.38	4.14	34.17	1.16	26.60	1.59	28.68	3.82	27.28
CoSim-CNN	2.50	35.53	1.49	27.20	0.53	18.44	2.78	33.36	10.37	38.03
CoSim-ATT	2.04	33.41	0.97	22.95	0.73	16.26	1.39	27.27	**1.53**	16.57
CoSim-SAG	3.26	38.85	3.30	33.14	1.91	35.48	1.55	29.84	1.62	**16.08**
CoSim-TopK	3.44	40.87	1.24	25.61	0.88	20.63	2.04	34.28	1.98	20.02
CoSim-Mem	5.45	48.07	1.11	24.59	**0.32**	**14.82**	1.74	26.78	1.57	17.02
CoSim-GNN10	2.04	33.04	1.01	23.53	0.40	16.43	1.38	27.43	1.68	17.57
CoSim-GNN1	**1.84**	**32.36**	**0.95**	**22.06**	0.36	15.42	**1.17**	**25.73**	2.00	18.62

is the least average time a model need to compute similarity over one graph pair with the strategy similar to what we discuss in Section 3. *MSE* and *MAE* measure the average squared/absolute difference between the predicted similarities and the ground-truth similarities. Details and results about τ, ρ and $p@k$ will be shown in the appendix A.

Baseline There are three types of baselines. The first category consists of traditional methods for GED computation, where we include *A*-Beamsearch (Beam)* (Neuhaus, Riesen, and Bunke 2006), *Hungarian* (Kuhn 1955) (Riesen and Bunke 2009), and *Vj* (Jonker and Volgenant 1987) (Fankhauser, Riesen, and Bunke 2011). *Beam* is one of the variants of A* algorithm and its time complexity is subexponential. *Hungarian* based on the Hungarian Algorithm for bipartite graph matching, and *Vj* based on the Volgenant and Jonker algorithm, are two algorithms in cubic-time. The second category is made up of embedding models, including *GCN-Mean* and *GCN-Max* (Defferrard, Bresson, and Vandergheynst 2016). The third category consists of matching models and we here involve *GMN* (Li et al. 2019) and two matching-based models: *SimGNN* (Bai et al. 2018a) and *GSimCNN* (Bai et al. 2018b).

Setting in Our Proposed Framework We provide experiments on seven kinds of variants of our framework to demonstrate its scalability and the effectiveness of ADAPTIVE POOLING. The names and settings of these seven variants are shown in Table 2. The parameters in our framework are as below: $k = 3$, $n_{encode} = 64$ (same in baselines), $h = 5$, $m = 5$, $l_{ins} = 2$, $l_{node} = 2$, $batchsize = 128$ (same in baselines).

We train models for 2000 iterations for BA datasets, 5000 iterations for IMDB-L and 10000 iterations on ER. The model that performs the best on validation sets is selected for testing. All experiments are conducted on the same device and details about this device are presented in appendix D.

Results and Analysis

Effectiveness and Efficiency Statistic results are shown respectively in Table 3 and 4. For better understanding the results, we highlight best *MSE* and *MAE* results among all the models in Table 3 and highlight the least time consumption respectively among the four different categories (traditional methods, embedding models, matching models and our proposed models) in Table 4. It is shown in the statistic that models in our framework outperform matching models in every dataset and our proposed pooling layer achieves the best performance among all the tested pooling method. Here we emphasize three aspects. Firstly, it can be found

Table 4: Results for average time consumption on one pair of graphs in milliseconds.

Method	**BA-60**	**BA-100**	**BA-200**	**ER-100**	**IMDB-L**
HUNGARIAN	>100	>100	>100	>100	>100
VJ	>100	>100	>100	>100	>100
BEAM	>100	>100	>100	>100	>100
GCN-MEAN	**1.31**	1.48	**1.79**	1.51	1.68
GCN-MAX	1.32	**1.46**	1.85	**1.49**	**1.58**
SIMGNN	2.92	3.24	5.26	3.46	4.34
GSIMCNN	**2.07**	**2.73**	**5.16**	**2.86**	**3.16**
GMN	5.77	9.36	28.57	10.00	9.89
COSIM-CNN	1.99	2.16	2.67	2.19	2.21
COSIM-ATT	1.85	2.02	2.34	1.99	2.12
COSIM-SAG	2.18	2.59	3.97	2.65	2.33
COSIM-TOPK	2.08	2.28	2.51	2.28	2.83
COSIM-MEM	3.30	3.56	3.95	3.50	3.58
COSIM-GNN10	3.29	3.51	4.03	3.63	3.09
COSIM-GNN1	**1.83**	**2.01**	**2.31**	**1.97**	**2.08**

Table 5: Results on Aminer dataset to show models' generalization ability. MSE and MAE are in 10^{-3} and time is in second.

Method	GCN-MAX	GCN-MEAN	GSIMCNN	COSIM-GNN
MSE	7.76	8.43	48.37	**3.64**
MAE	59.79	73.45	176.44	**50.37**
Time	6.72	6.96	32.00	**6.46**

that for all the datasets, our framework outperforms all the traditional methods, embedding models and matching models on both *MSE* and *DEV*. Secondly, it is shown that our COSIM-GNN1 runs faster than any matching model. When we take comparisons among BA datasets, it is clear that our time consumption keeps low as the node number increases and the gaps on time consumption between our models and matching models keep increasing. Especially when we look at COSIM-GNN1 and GMN, our model requires $\frac{1}{3}$, $\frac{1}{5}$, $\frac{1}{14}$, $\frac{1}{5}$ and $\frac{1}{5}$ as the time consumption of GMN respectively on **BA-60**, **BA-100**, **BA-200**, **ER-100** and **IMDB-L**. Thirdly, when we look at the results among all the models under our framework, it can be found that in four out of the all 5 datasets, our pooling layer performs better than all other pooling layers, which demonstrates the effectiveness of our proposed pooling layer.

Generalization Sometimes, due to the limitation of computing resources, we may not be able to train models on very large graph datasets (graphs that have thousands of nodes) efficiently. Therefore, it's interesting and meaningful to explore whether our model can be efficiently trained on small graph dataset and then be effectively applied to infer similarity on large graph pairs.

In this paper, we derive two sub-networks from the academic network of AMiner [1]. The first sub-network includes 1,397 authors and 1446 edges representing the coauthor relationship and the second sub-network includes 1324 papers and 1977 edges representing the citation relationship. We use these two as the basic graphs, and then use the previously mentioned trimming method (the trimming steps are 10, 20, ..., 100), to get 198 trimming graphs, which with the two basic graphs constitute our Aminer dataset. Experimental results are in Table 5. In our experiments, GMN (Li et al. 2019) can not be deployed to this dataset even with a 32GB RAM (error in memory overflow). It is found that CoSim-GNN outperform the other techniques in both MSE and MAE on the different dataset where graphs have much more nodes. Also the time consumption of CoSim-GNN is even lower than embedding models. This results are very promising in many other application including analysis on social network and very large biological molecular.

[1]https://aminer.org/aminernetwork

Table 6: Ablation study in matching part.

	X&I&M	X&M	M	X&I
MSE	0.95	1.02	1.11	2.56
MAE	22.06	23.49	23.76	38.19

Table 7: Accuracy on classification task in %.

Method	BA60	BA100	BA200	OpenSSL	COIL
GCN-Mean	92.56	91.43	91.38	78.93	73.94
GCN-Max	92.19	91.38	90.94	80.72	72.10
GSimCNN	93.78	98.75	98.94	89.50	80.44
CoSim-Mem	93.22	95.00	99.63	90.45	83.85
CoSim-GNN	**97.50**	**99.38**	**99.75**	**94.66**	**88.30**

Ablation Study

We conduct an ablation study on BA-100 dataset to validate the key components that contribute to the improvement of our COSIM-GNN.

We focus on equation (11) and select different combination of $\mathbf{X}$, $\mathbf{I}$ and $\mathbf{M}$ in the concatenation operation. Results are shown in Table 6. We can see that after removing $\mathbf{X}$ or $\mathbf{I}$ or $\mathbf{M}$ from COSIM-GNN, the performance will drop, which proves that all modules are useful for similarity computation. What's more, it can be seen that if $\mathbf{M}$ is removed from the framework, the performance will drop the most. The conclusion here is that all the modules contributed to the framework and the main contribution is brought by $\mathbf{M}$ with the across graph inter-attention.

Graph Pair Classification

We conduct similarity classification experiments (classify graph paris to "similar" or "dissimilar") on the four synthetic datasets we generate and two real benchmark datasets: OpenSSL (Xu et al. 2017) and COIL-DEL (COIL (Riesen and Bunke 2008)). In Table 7, it can be found that "CoSim" models achieve the best performance on both synthetic and real datasets in classification task. Experiments also show that more pooling layers lead to better GED regression performance only when the node numbers of graphs are large, i.e. on the BA200 dataset, and here we only use one coarsening layer for all graph classification tasks.

Conclusion

In this paper, we propose CoSim-GNN for large graphs similarity computation, including three stages: encoding, coarsening and matching. Noticing that the generation of pooled centroids does not rely on the input graphs among current state-of-art pooling methods, we provide a novel pooling method ADAPTIVE POOLING. Thorough experiments are conducted on various baselines, datasets and evaluation metrics to demonstrate the scalability, effectiveness and efficiency of CoSim-GNN as well as the effectiveness of ADAPTIVE POOLING. CoSim-GNN opens up the door for learning large graph similarity and the future focus will be on more efficient pooling and matching mechanisms, which are able to improve the large graph similarity regression and classification in this novel framework.

References

Bai, Y.; Ding, H.; Bian, S.; Chen, T.; Sun, Y.; and Wang, W. 2018a. SimGNN: A Neural Network Approach to Fast Graph Similarity Computation.

Bai, Y.; Ding, H.; Gu, K.; Sun, Y.; and Wang, W. ???? Learning-based Efficient Graph Similarity Computation via Multi-Scale Convolutional Set Matching.

Bai, Y.; Ding, H.; Sun, Y.; and Wang, W. 2018b. Convolutional set matching for graph similarity. *arXiv preprint arXiv:1810.10866* .

Bollobás, B.; and Béla, B. 2001. *Random graphs*. 73. Cambridge university press.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In Lee, D. D.; Sugiyama, M.; Luxburg, U. V.; Guyon, I.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 29*, 3844–3852. Curran Associates, Inc. URL http://papers.nips.cc/paper/6081-convolutional-neural-networks-on-graphs-with-fast-localized-spectral-filtering.pdf.

Erdos, P. 1959. On random graphs. *Publicationes mathematicae* 6: 290–297.

Fankhauser, S.; Riesen, K.; and Bunke, H. 2011. Speeding Up Graph Edit Distance Computation through Fast Bipartite Matching. In Jiang, X.; Ferrer, M.; and Torsello, A., eds., *Graph-Based Representations in Pattern Recognition*, 102–111. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-20844-7.

Gao, H.; and Ji, S. 2019. Graph U-Nets.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural Message Passing for Quantum Chemistry.

Jeong, H.; Néda, Z.; and Barabási, A.-L. 2003. Measuring preferential attachment in evolving networks. *EPL (Europhysics Letters)* 61(4): 567.

Jonker, R.; and Volgenant, A. 1987. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing* 38(4): 325–340.

Kendall, M. G. 1938. A new measure of rank correlation. *Biometrika* 30(1/2): 81–93.

Khasahmadi, A. H.; Hassani, K.; Moradi, P.; Lee, L.; and Morris, Q. 2020. Memory-Based Graph Networks. In *International Conference on Learning Representations*. URL https://openreview.net/forum?id=r1laNeBYPB.

Kriegel, H.-P.; Pfeifle, M.; and Schönauer, S. 2004. Similarity Search in Biological and Engineering Databases. *IEEE Data Eng. Bull.* 27(4): 37–44.

Ktena, S. I.; Parisot, S.; Ferrante, E.; Rajchl, M.; Lee, M.; Glocker, B.; and Rueckert, D. 2017. Distance Metric Learning using Graph Convolutional Networks: Application to Functional Brain Networks.

Kuhn, H. W. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly* 2(1-2): 83–97.

Lee, J.; Lee, I.; and Kang, J. 2019. Self-attention graph pooling. *arXiv preprint arXiv:1904.08082* .

Li, Y.; Gu, C.; Dullien, T.; Vinyals, O.; and Kohli, P. 2019. Graph Matching Networks for Learning the Similarity of Graph Structured Objects.

McCreesh, C.; Prosser, P.; and Trimble, J. 2017. A partitioning algorithm for maximum common subgraph problems .

Neuhaus, M.; Riesen, K.; and Bunke, H. 2006. Fast Suboptimal Algorithms for the Computation of Graph Edit Distance. In Yeung, D.-Y.; Kwok, J. T.; Fred, A.; Roli, F.; and de Ridder, D., eds., *Structural, Syntactic, and Statistical Pattern Recognition*, 163–172. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-37241-7.

Ogaard, K.; Roy, H.; Kase, S.; Nagi, R.; Sambhoos, K.; and Sudit, M. 2013. Discovering patterns in social networks with graph matching algorithms. In *International Conference on Social Computing, Behavioral-Cultural Modeling, and Prediction*, 341–349. Springer.

Riesen, K.; and Bunke, H. 2008. IAM graph database repository for graph based pattern recognition and machine learning. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, 287–297. Springer.

Riesen, K.; and Bunke, H. 2009. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision computing* 27(7): 950–959.

Riesen, K.; Emmenegger, S.; and Bunke, H. 2013. A novel software toolkit for graph edit distance computation. In *International Workshop on Graph-Based Representations in Pattern Recognition*, 142–151. Springer.

Sanfeliu, A.; and Fu, K. 1983. A distance measure between attributed relational graphs for pattern recognition.

IEEE Transactions on Systems, Man, and Cybernetics SMC-13(3): 353–362.

Spearman, C. 1961. The proof and measurement of association between two things. .

Steinhaeuser, K.; and Chawla, N. V. 2008. Community detection in a large real-world social network. In *Social computing, behavioral modeling, and prediction*, 168–175. Springer.

Tian, Y.; Mceachin, R. C.; Santos, C.; States, D. J.; and Patel, J. M. 2007. SAGA: a subgraph matching tool for biological graphs. *Bioinformatics* 23(2): 232–239.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* .

Wang, Y.; Sun, Y.; Liu, Z.; Sarma, S. E.; Bronstein, M. M.; and Solomon, J. M. 2019. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)* 38(5): 1–12.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018. How Powerful are Graph Neural Networks? *ArXiv* abs/1810.00826.

Xu, X.; Liu, C.; Feng, Q.; Yin, H.; Song, L.; and Song, D. 2017. Neural network-based graph embedding for cross-platform binary code similarity detection. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 363–376.

Yanardag, P.; and Vishwanathan, S. 2015. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1365–1374.