Graph Partitioning and Graph Neural Network based Hierarchical Graph Matching for Graph Similarity Computation

Haoyan Xu∗
haoyanxu@zju.edu.cn
College of Control Science and Engineering
Zhejiang University

Ziheng Duan∗
duanziheng@zju.edu.cn
College of Control Science and Engineering
Zhejiang University

Jie Feng∗
zjuce_fj@zju.edu.cn
College of Control Science and Engineering
Zhejiang University

Runjian Chen
rjchen@zju.edu.cn
College of Control Science and Engineering
Zhejiang University

Yida Huang
stevenhuang@zju.edu.cn
College of Computer Science and Engineering
Zhejiang University

Yueyang Wang†
yueyangw@cqu.edu.cn
School of Big Data and Software Engineering
Chongqing University

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

Motivated by this observation, we propose a graph partitioning and graph neural network based model, called PSimGNN, to effectively resolve this issue. Specifically, each of the input graphs is partitioned into a set of subgraphs to directly extract the local structural features firstly. Next, a learnable embedding function is used to map each subgraph into an embedding vector. Then, some of these subgraph pairs are selected for node-level comparison to supplement the subgraph-level embedding with fine-grained information. Finally, coarse-grained interaction information among subgraphs and fine-grained comparison information among nodes in different subgraphs are integrated to predict the final similarity score.

Using approximate Graph Edit Distance (GED) as graph similarity metric, experimental results on graph data sets of different graph size demonstrate PSimGNN outperforms state-of-the-art methods in graph similarity computation tasks. The codes will release when this paper is published.

KEYWORDS
graph deep learning, graph similarity computation, graph partition

1 INTRODUCTION
Graph similarity computation and graph matching techniques have been widely used in various fields, such as recommendation system [33], computer vision [14, 27] and so on. Graph Edit Distance (GED) [7] and Maximum Common Subgraph (MCS) [9] are two common metrics for evaluating how similar the two graphs are. The

Figure 1: Graph similarity search. Given a database $\mathcal{D} = \{g_1, g_2, g_3, \ldots\}$ consisting of a set of nodes and edges labeled graphs, graph similarity search aims to find all graphs in $\mathcal{D}$ that are similar to a user-given query graph $q$.

traditional graph similarity computation methods such as $A^*$ [29], Hungarian [20, 28], VJ [11, 16], Beam[24], perform directly from the edges and nodes characteristics of the graphs. These exact and approximate algorithms for computing the GED or MCS between two graphs have a high time-complexity, and are hard to be generalized to large graphs in real applications such as graph similarity search. As shown in Figure 1, in the task of graph similarity search, how to retrieve graphs $g$ in the graph database $\mathcal{D}$ that approximately match or similar with the query graph $q$ efficiently is challenging but meaningful.

With the rapid development of deep learning technology, Graph Neural Networks (GNN) provide a new solution for similarity computation and matching of graph structures. Graph deep learning models automatically extract the structural characteristics of the graph through GNN layers, embed each node into a low-dimension vector containing both its own feature information and its local connection relationship information. SimGNN [3], GSIMCNN[4], GMN [22] are some representative graph deep learning based models.
for graph similarity computation. During the training stage, these
models fit the GED or MCS ground truth (label) in a supervised
learning way and learn a mapping between a pair of graph inputs
and the similarity score. When testing or during actual applications,
deep learning models are time-efficient compared with traditional
graph similarity computation methods. However, these deep learn-
ing models still either contain Pairwise Node Comparison process
which need at least quadratic time with respect to the number of
nodes in two graphs, or they only directly embed the whole graph
to a graph-level vector. Embedding the whole graph to a vector and
computing the similarity between vectors as the similarity of the
corresponding graph pairs is time-efficient but will lose much node-
level information. Graph similarity computation is challenging not
only because of the demand to perceive the whole graph, but also
the need for careful comparison between the information of the
nodes.

In this paper, we focus on large graph similarity computation
and try to address the following two challenges. First, although
many graph similarity computation models based on neural net-
works have been proposed, few of them can effectively calculate the
similarity between graphs with a large number of nodes, and adjust
the models according to different tasks to achieve the trade-off
between accuracy and time complexity. Second, for a graph with a
large number of nodes, the local information of the graph, such
as the structural characteristics of the subgraph, should also be
directly concerned. However, none of the existing models consider
these challenges in detail.

Although large graph similarity computation is still a very chal-
 lenging issue, we notice that with the advent of ever larger instances
in applications such as scientific simulation, social networks, or
road networks, graph partition (GP) becomes more and more im-
portant, multifaceted, and challenging [6]. Inspired by these appli-
cations that based on graph partitioning methods, we propose an
ded-to-end model PSimGNN, i.e., Partition based Similarity Com-
putation via Graph Neural Networks, to address these challenges.
First, our model partitions each of the input graph into a set of
subgraphs to directly extract the local structural features. Second,
a learnable embedding function is used to map every subgraph
into an embedding vector. Then coarse-grained similarity com-
putation is conducted by computing the similarity of each pair of
subgraph-level embedding vectors. Third, our model can select some of these subgraph pairs for node-level comparison
to supplement the subgraph-level embedding with fine-grained
information. Finally, the model integrate coarse-grained interac-
tion information between subgraphs and fine-grained comparison
information between nodes in different subgraph pairs to predict
the final similarity score. Our model is especially effective in graph
structure similarity computation of large graphs, compared with
other state-of-the-art graph similarity computation models which
need at least quadratic time with respect to the number of nodes in
two graphs. Thus our major contributions are:

- We first propose the graph partitioning based framework to ad-
dress the challenging problem of similarity computation between
large graphs. This framework achieves a good trade-off between
accuracy and efficiency.
- We propose a novel model that extract and aggregate local infor-
mation effectively to conduct subgraph-level comparison. This
can resolve the challenges of under representation ability or high
time complexity of many graph deep learning based similarity
computation models.
- We conduct extensive experiments on a very popular graph simi-
ilarity/distance metric, GED, based on datasets of different size.
Results from these experiments and theoretical analysis demon-
strate the effectiveness and efficiency of PSimGNN model in
graph similarity computation tasks.

2 RELATED WORK

In this section, we introduce the concept of graph partitioning,
graph neural network and graph similarity that will be used in this
paper.

2.1 Graph Partitioning

Graph partitioning is an effective way for complexity reduction or
parallelization [6]. In mathematics, a graph partition is the reduc-
tion of a graph to a smaller subgraph by partitioning its set of nodes into
mutually exclusive groups and the partitioned graph may be better
suited for analysis and problem-solving than the original. Finding
a partition that simplifies graph analysis is a hard problem, but one
that has applications to scientific computing, VLSI circuit design,
and task scheduling in multiprocessor computers, among others [2].
Recently, the graph partition problem has gained importance due
to its applications for clustering and detection of cliques in social,
pathological and biological networks [6].

2.2 Graph Neural Networks

Graph Neural Networks (GNNs) are an effective framework for
representation learning of graphs, which can directly operate on the
graph structure. GNNs follow a neighborhood aggregation
scheme, where the representation vector of a node is computed by
recursively aggregating and transforming representation vectors
of its neighboring nodes. Many GNN variants have been proposed
and have achieved state-of-the-art results on both node and graph
classification tasks. However, despite GNNs revolutionizing graph
representation learning, there is limited understanding of their
representational properties. Studies [34] have shown that popular
GNN variants (such as graph convolutional networks and Graph-
SAGE [12]) have limited discriminative power, and they cannot
learn to distinguish certain simple graph structures. Therefore, we
use Graph Isomorphism Network (GIN) [34] in our experiment that
is the most expressive in the GNN class and has the same powerful
function as the Weisfeiler Lehman graph isomorphism test [30].

2.3 GED & MCS

Graph Edit Distance (GED) [7] can be considered as an extension
of the String Edit Distance [21] metric, which is defined as the
minimum cost required to convert one graph to another through a
sequence graph editing operations. Maximum Common Subgraph
(MCS) [9] is equivalent to GED under the same cost function [8].
Both are the two most common ways to calculate the similarity of
graphs or the distance between graphs, which is the core operation
of graph similarity search and many applications. However, this
core operation, computing the GED or MCS between two graphs, is known to be NP-complete [9, 36]. For a pair of graphs with more than 16 nodes, even the state-of-the-art algorithms cannot reliably compute the exact GED within reasonable time [5]. So, instead of calculating the exact similarity, there are some methods which can find approximate values in a fast and heuristic way. However, these methods usually require complicated design and the time complexity is still sub-exponential or polynomial in the number of nodes in the graphs, such as Hungarian [20, 28], VJ [11, 16], Beam[24], etc.

3 THE PROPOSED APPROACH: PSIMGNN

In this section, we formally define the problem of graph similarity computation, and then introduce the proposed method PSimGNN, i.e., Partition based Similarity Computation via Graph Neural Networks, which is an end-to-end neural network-based method to solve graph similarity computation problem. PSimGNN consists of four parts: (1) graph partitioning; (2) subgraph-level embedding interaction; (3) node-level comparison; (4) graph similarity score computation. An overview of PSimGNN is shown in the Figure 2.

3.1 Problem Definition

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

3.2 Graph Partitioning

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

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

To overcome these limitations and better reflect the concept of local structure characteristics of large graphs, we partition a graph into \( k \) subgraphs using graph partitioning method. In our experiments, the Fluid Communities algorithm (FluidC) [26] shows the best performance. The graph partitioning contains three steps:

- Step-1: Choose \( k \) nodes randomly in the graph as the initial nodes of \( k \) communities.
- Step-2: Iterate over all nodes in a random order and update the community of each node based on its own community and the communities of its neighbours.
- Step-3: Repeat step-2 until convergence.

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

3.3 Subgraph-Level Embedding Interaction

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

### 3.3.1 Part I: Subgraph Node Embedding

Among the existing multiple graph neural network methods, we choose Graph Isomorphism Network (GIN) [34] because it can not only efficiently gather information of neighboring nodes like Graph Convolutional Networks (GCN) [10, 19] and GraphSAGE [12], but also learn accurate structural information. The hidden layer can be written as follow:

$$ h_v^{(k)} = MLP^{(k)}((1 + e^{(k)}) \cdot h_v^{(k-1)} + \sum_{u \in N(v)} h_u^{(k-1)}) $$  \hspace{1cm} (1)

where $h_v^{(k)}$ is the k-th layer node embedding for the node $v$, MLP means Multilayer Perceptron [25], $e$ is a learnable parameter and $N(v)$ represents the neighbor nodes of node $v$.

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

### 3.3.2 Part II: Subgraph Embedding

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

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

Through learning the weight matrix, $z$ provides the global structure and feature information of the subgraph suitable for a given similarity measure. Based on $z$, we can calculate an attention weight for each node. For node $i$, in order to make it notice the global information, we take the inner product between its node embedding $x_i$ and $z$. That is to say, the nodes that are more capable of expressing the features of the graph should be given higher weights. The sigmoid function $\sigma(x) = 1 + \exp^{-x}$ is applied to the result to ensure that the attention weight is between (0, 1). Finally, subgraph...
embedding \( h \in \mathbb{R}^D \) is the weighted sum of node embedding:

\[
h = \sum_{j=1}^{N} \sigma(x_j \odot z)x_j = \sum_{j=1}^{N} \sigma \left( x_j \odot \tanh \left( \frac{1}{N} \sum_{i=1}^{N} u_j W_j \right) \right)x_j
\]

(2), where \( \odot \) represents the dot product between vectors.

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

\[
s(h_i, h_j) = \cos(h_i, h_j) = \frac{h_i \odot h_j}{||h_i||_2 ||h_j||_2}
\]

(3), where \( ||h||_2 \) is the 2-norm of \( h \).

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

\[
s(G_1, G_2) = \text{MLP} \left( \bigoplus_{i=1}^{k} s(G_{1,i}^j, G_{2,j}^k) \right)
\]

(4), where \( \bigoplus \) is the concatenation operation, \( s(G_1, G_2) \) represents the similarity score between the pair of large graphs and \( s(G_{1,i}^j, G_{2,j}^k) \) represents the similarity score between the \( i \)-th subgraph of \( G_1 \) and the \( j \)-th subgraph of \( G_2 \).

3.4 Node-Level Comparison

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

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

\[
v_{j \rightarrow i} = \text{MLP}(h_i^{(t)} \odot h_j^{(t)}) \forall (i, j) \in E_1^m \cup E_2^n
\]

(5)

An attention mechanism is used to give different weights to the nodes of another subgraph to indicate the importance of different nodes \( j \) to nodes \( i \):

\[
a_{j \rightarrow i} = \frac{\exp(h_i^{(t)} \odot h_j^{(t)})}{\sum_{j'} \exp(h_i^{(t)} \odot h_{j'}^{(t)})}
\]

(6)

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

\[
\mu_{j \rightarrow i} = a_{j \rightarrow i}(h_i^{(t)} - h_j^{(t)}) \forall i \in V_1, j \in V_2, \text{or} \forall i \in V_2, j \in V_1
\]

(7)

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

\[
h_i^{(t+1)} = \text{MLP}(h_i^{(t)} \odot \sum_{j} v_{j \rightarrow i} \odot \sum_{j'} \mu_{j' \rightarrow i})
\]

(8)

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

\[
h_{agg} = \text{MLP}_{agg} \left( \sum_{i \in V} \sigma(\text{MLP}_{att}(h_i^{(T)})) \odot \text{MLP}(h_i^{(T)}) \right)
\]

(9)

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

\[
s(h_{agg1}, h_{agg2}) = \cos(h_{agg1}, h_{agg2}) = \frac{h_{agg1} \odot h_{agg2}}{||h_{agg1}||_2 ||h_{agg2}||_2}
\]

(10)

3.5 Graph Similarity Score Computation

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

\[
s(G_1, G_2)_{coarse} = \text{MLP} \left( \bigoplus_{i=1}^{k} s(G_{1,i}^j, G_{2,j}^k) \right)
\]

(11a)

\[
s(G_1, G_2)_{fine} = \text{MLP} \left( \bigoplus_{i=1}^{m} s(G_{1,i}^j, G_{2,j}^k) \right)
\]

(11b)

\[
s(G_1, G_2) = \text{MLP} \left( s(G_1, G_2)_{coarse} \odot s(G_1, G_2)_{fine} \right)
\]

(11c)
After the similarity score, \( s(G_1, G_2) \in \mathbb{R} \), is predicted, it is compared with the ground truth similarity score, \( s(G_1, G_2)_{gt} \in \mathbb{R} \), using the following mean square error loss function:

\[
\mathcal{L} = \frac{1}{|T|} \sum_{(i,j) \in T} \left( s(G_1, G_2) - s(G_1, G_2)_{gt} \right)^2
\]  

where \( T \) is the training graph pairs and \(|T|\) is the total number of the training graph pairs.

### 3.6 Time Complexity Analysis

For a pair of input graphs \( G_1 \) and \( G_2 \) with \( E_1 \), \( E_2 \) edges and \( N_1 \) and \( N_2 \) nodes separately, we can evaluate the time complexity of several types of models that are commonly used in graph similarity computation. Then we analyze the time complexity of PSimGNN and discuss how it can reduce the time complexity by graph partitioning. Note that due to the fact that there exists a lot of variance for each model, we only use the simplest cases.

#### 3.6.1 Embedding models

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

#### 3.6.2 Matching models

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

#### 3.6.3 PSimGNN

The time complexity of PSimGNN can be divided into three parts to analyze. (1) Graph Partitioning. In our model, we choose FluidC as the graph partitioning method. As analyzed in Section 3.1, it updates node information based on neighbor nodes, or the connected edges of nodes, so it belongs to the fastest and most scalable family of algorithms in the literature with a linear computational complexity of \( O(E) \) [26]. Notice that the partitioned subgraphs can be pre-computed and stored, and in the setting of graph similarity search, the unseen query graph only needs to be partitioned once to obtain its subgraphs. (2) Subgraph-level Embedding Interaction. The time complexity associated with the generation of node-level and subgraph-level embeddings is \( O(E)[19] \). Assuming that each graph is partitioned into \( k \) subgraphs and the embedding dimension at the subgraph-level is \( D \), we use cosine to measure the similarity between embeddings, so the time complexity in the subgraph interaction part is \( O(Dk^2) \). As mentioned above, the subgraph level and node-level embedding can also be saved in advance, which greatly saves the time of graph similarity query. (3) Node-level Comparison. According to the \( k^2 \) similarity scores obtained by subgraph-level interaction, we select top \( m \) subgraph pairs with the highest similarity scores for node-level comparison. After partitioning, the average number of nodes in each subgraph is \( N_1/k \) or \( N_2/k \).

As analyzed in Section 3.5.2, the average node-level comparison time complexity of one pair of subgraphs is \( O(\max(N_1/k, N_2/k)^2) \). Since we choose \( m \) pairs, the total time complexity of this part is \( O(m \times \max(N_1/k, N_2/k)^2) \) or \( O(m/k^2 \times \max(N_1, N_2)^2) \), where the range of \( m \) belongs to \( \{0, 1, 2, ..., k^2\} \). The parameter \( m \) can be used as a hyperparameter to adjust the relationship between the accuracy and time. When \( m \) is set to zero, our model only calculates the coarse-grained subgraph similarity. At this time, the time complexity of the model is \( O(E) \), where \( E \) is the number of edges in the large graph. When \( m \) is set to \( k^2 \), our model performs fine-grained similarity calculation for each pair of subgraphs, and the time complexity of the model is \( O(N^2) \), where \( N \) is the number of nodes in the large graph. For occasions with time requirements, we can only perform coarse-grained matching between subgraphs, and for occasions where accuracy requirements are relatively high, we can perform fine-grained node-level comparison to improve model performance. Therefore, according to specific application scenarios, trade-offs between time and accuracy can be made to choose the best solution.

#### 3.6.4 Similarity search

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

## 4 EXPERIMENTS

### 4.1 Datasets

In this section, we first introduce a graph similarity computation dataset based on Barabási–Albert preferential attachment model (BA-model) [15], which consists of three sub-datasets: BA-60, BA-100, BA-200, named according to the average number of nodes per graph. And we compare our released BA-model dataset with other well-known datasets used for graph similarity computation.

#### 4.1.1 Barabási–Albert model dataset

Here we introduce the concept of Barabási–Albert model (BA-model), the rules for generating a Barabási–Albert graph (BA-graph), and how our datasets are produced. The BA-model [15] is an algorithm for generating random scale-free networks using a preferential attachment mechanism. Several natural and human-made systems, including the Internet, the world wide web, citation networks, and some social networks are thought to be approximately scale-free and certainly
Table 1: Statistics of datasets.

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

Figure 5: Nodes degree distribution of BA-model datasets.

- contain few nodes (called hubs) with unusually high degree as compared to the other nodes of the network. The BA-model tries to explain the existence of such nodes in real networks and it incorporates two important general concepts: growth and preferential attachment, which exist widely in real networks. Growth means that the number of nodes in the network increases over time and preferential attachment means that the more connected a node is, the more likely it is to receive new links. Nodes with a higher degree have a stronger ability to grab links added to the network.

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

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

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

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

- A BA-graph of \( n \) nodes is grown by attaching new nodes each with \( m \) edges that are preferentially attached to existing nodes with high degree. We set \( n \) to be 60, 100, and 200, respectively, and \( m \) is fixed to 1, to generate basic graphs. Each sub-dataset generates two basic graphs, and each base graph is trimmed with different GEDs. For each basic graph, generate 99 trimmed graphs in the range of GED 1 to 10. So each sub-dataset consists of two basic graphs and 198 trimmed graphs.

4.1.2 Comparison with Other Datasets. Because in public datasets, such as AIDS [23] and LINUX [32], the number of nodes in each graph is relatively small and local structures are not obvious, the characteristics of the entire graph can be easily represented. As for
IMDB [35], (named 'IMDB-MULTI') there are some graphs with a large number of nodes. However, these graphs are relatively dense and too many edges between nodes will make the local structures less obvious. So we did not choose these three datasets.

In view of this, we artificially made three BA-datasets, which not only has a large number of nodes, but also has graphs with obvious local structures by using the characteristics of the BA-model. Table 1 shows the comparison of different datasets for graph similarity computation. Figure 5 shows the nodes degree distribution of BA-model datasets. From these charts, we can see that the ratio of the average number of nodes to the number of edges in the BA-model datasets is approximately equal to 1. Graphs are relatively sparse and suitable for extracting local structural features by graph partitioning. The degree distribution indicates that most nodes have relatively low degrees, and only a few have high degrees. These nodes have a greater probability of becoming the center node of the subgraph. The partitioning results of the two graphs in the BA-60 dataset are shown in Figure 6. Through graph partitioning, obvious local structural features can be extracted, which is also a characteristic of our BA-model datasets.

4.2 Data Preprocessing

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

Due to the large number of nodes in our data set, A* [29] algorithm cannot be used to calculate the GED. We used the smallest distance calculated by three well-known approximation algorithms, Hungarian [20, 28] and VJ [11, 16], and Beam [24]. However, these algorithms are also difficult to ensure a certain accuracy in this case. So we also added the GED value generated when trimming the graph as another evaluation indicator. When each graph is trimmed, we will get a GED value to record the number of trimming steps. In this experiment, every time a leaf node is deleted, the edge it connects will also be deleted. In this case, the GED between the derived graph and the basic graph increases by 2; and every time an edge is added, GED increases by 1.

As shown in Figure 7, when generating a derivative graph with a specific GED from the basic graph, we randomly select among the above three methods (randomly delete a leaf node, add a leaf node or add an edge) to generate a set of operations, and the sum of GED accumulated by all operations is the specific GED value. We take the minimum value of the trimming GED and the calculated GED with these three algorithms as the final GED value. Here, the minimum value is taken instead of the average value because GED is the upper bound. The real GED value must be less than or equal to the GED generated here.

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

4.3 Baseline Methods

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

- The first category of baseline includes three classic algorithms for GED calculation. (1) Hungarian [20, 28] is a cubic-time algorithms based on the Hungarian Algorithm for bipartite graph matching. (2) VJ [11, 16] is also a cubic-time algorithms based on the algorithm of Volgenant and Jonker. (3) Beam search (Beam) [24]. The equivalent variable of the A* algorithm is sub-exponential time.

- The second category of baseline includes two graph embedding based models, GCN-Mean and GCN-Max. They all embed graphs into vectors using GCN, and then use the similarities calculated by these vectors as the similarities of these graph pairs.

- The third category of baseline includes three graph matching network based models. (1) SimGNN [3] and (2) GSimCNN [4]...
combine the embedding of the whole graph and node-level comparison. (3) GMN [22] uses the comparison node information within and between graphs to calculate similarity.

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

4.4 Parameter Settings

For the architecture of our model, PSimGNN, we partition each large graph into \( k \) (here \( k=3 \)) subgraphs. Among the 9 subgraph pairs, 0, 3 and 9 subgraph pairs with the highest similarity scores are selected for node-level comparison respectively. Here we call them PSimGNN-up (only subgraph-level interactions are involved in the computation), PSimGNN-k \( (k \geq 3 \) pairs of subgraphs are participated in the node-level comparison) and PSimGNN \( (k \geq 3 \) pairs of subgraphs are participated in the node-level comparison).

We set the number of GIN [34] layer to 3, and use Parametric Rectified Linear Unit (PReLU) [13] as the activation function. For the initial node representations, we adopt the constant encoding scheme for BA-datasets, since their nodes are unlabeled, as mentioned in Section 3.2.1. The dimensions of the 1st, 2nd and 3rd layer of GIN’s output are 64, 32 and 16, respectively. We use a fully connected layer to reduce dimension of the similarity vectors obtained at the subgraph-level interaction from 9 to 8, and another fully connected layer to change the dimension of the similarity vector after the node-level comparison from 8 to 4. Finally, 4 fully connected layers are used to reduce the dimension of the concatenated results from the subgraph-level interaction and the node-level comparison module, from 16 to 8, 4 to 2, and 2 to 1.

For training, we set the batch size to 128, use the Adam algorithm [18] for optimization, and set the initial learning rate to 0.001. We set the number of training iterations to 2000, and choose the best model based on the lowest validation loss.

4.5 Evaluation Metrics

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

Table 4: Results on BA-200 dataset \( (10^{-2}) \)

| Method      | MSE     | MAE     | \( \rho \) | \( \tau \) | p@10  | p@20  |
|-------------|---------|---------|------------|------------|-------|-------|
| hungarian   | 25.91   | 37.94   | 79.38      | 58.10      | 64.25 | 94.00 |
| vj          | 31.44   | 42.68   | 61.91      | 43.10      | 48.50 | 80.38 |
| beam        | 18.60   | 28.79   | 77.24      | 65.21      | 56.00 | 83.50 |
| GCN-Mean    | 2.37    | 12.78   | 73.47      | 49.46      | 50.00 | 95.00 |
| GCN-Max     | 2.28    | 10.76   | 74.99      | 51.69      | 53.75 | 94.25 |
| SimGNN      | 0.84    | 6.19    | 73.47      | 48.89      | 52.75 | 95.13 |
| GSimCNN     | 0.32    | 3.58    | 79.68      | 56.82      | 59.00 | 95.00 |
| GMN         | 0.12    | 2.66    | 79.58      | 57.87      | 60.25 | 95.00 |
| PSimGNN-up  | 0.07    | 2.14    | 76.36      | 53.29      | 52.50 | 96.00 |
| PSimGNN-k   | 0.06    | 1.96    | 79.16      | 57.24      | 55.75 | 97.63 |
| PSimGNN     | 0.11    | 2.41    | 80.14      | 58.44      | 61.25 | 100.00|

Table 3: Results on BA-100 dataset \( (10^{-2}) \)

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

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

| Method      | MSE     | MAE     | \( \rho \) | \( \tau \) | p@10  | p@20  |
|-------------|---------|---------|------------|------------|-------|-------|
| hungarian   | 18.62   | 33.22   | 75.98      | 57.72      | 74.25 | 84.75 |
| vj          | 25.87   | 39.48   | 3.29       | 2.29       | 35.00 | 50.50 |
| beam        | 5.88    | 12.93   | 83.80      | 74.34      | 67.75 | 90.00 |
| GCN-Mean    | 0.58    | 5.39    | 75.64      | 53.29      | 58.00 | 86.88 |
| GCN-Max     | 1.37    | 9.14    | 74.61      | 52.30      | 54.50 | 86.62 |
| SimGNN      | 0.78    | 6.58    | 77.30      | 56.78      | 71.00 | 88.87 |
| GSimCNN     | 0.60    | 5.61    | 80.78      | 60.47      | 67.75 | 90.50 |
| GMN         | 0.27    | 3.82    | 76.36      | 54.67      | 60.00 | 89.00 |
| PSimGNN-up  | 0.44    | 4.80    | 78.92      | 57.63      | 59.50 | 88.37 |
| PSimGNN-k   | 0.32    | 4.07    | 80.43      | 60.31      | 70.50 | 88.00 |
| PSimGNN     | 0.20    | 3.39    | 84.49      | 66.15      | 78.50 | 91.87 |

Table 4: Results on BA-200 dataset \( (10^{-2}) \)

4.6 Results and Analysis

The effectiveness results on the three datasets can be found in Tables 2, 3 and 4. The two embedding models we use, GCN-Mean and GCN-Max, have worse results than any matching model, which proves that the representation ability of a single embedding is limited. When the number of nodes per graph increases, the limitation of using a vector to characterize the entire graph is more obvious and the results are worse, which also confirms our previous analysis in Section 3.2. PSimGNN-up only uses the subgraph-level embeddings and achieves the same level of evaluation results as other
matching models, which proves the effectiveness of our framework. PSimGNN-\(k\), which uses \(k\) subgraph pairs for node-level comparison, achieves better results than PSimGNN-up on all evaluation metrics. Our model, PSimGNN, consistently achieves the best or second best under most evaluation metrics across the three datasets within the neural network based methods. This implies that our model not only introduces a more flexible framework, but also performs the same level of accuracy as other neural network based models. And as the number of subgraph pairs using node-level comparison increases, the model contain more information and the corresponding evaluation results become better, which is also in line with our expectations.

The ranking results of VJ on the BA-60 dataset is extremely poor, and these three traditional methods have very high \(MSE\) and \(MAE\). These results show the limitations of traditional methods for graphs with a large number of nodes. As for the BA-100 dataset, \(\rho@20\) is 100%. This is because when randomly dividing the test dataset of 40 graphs, there are exactly 20 graphs from the basic graph 1, and the other 20 graphs are from the basic graph 2. Being able to distinguish these graphs perfectly also proves the excellent performance of the neural network based models.

PSimGNN does not achieve the best in some metrics in BA-100 and BA-200 datasets. This may be due to the randomness of graph partition. Some subgraphs do not extract the local structural features well when they are partitioned. Improving the stability and accuracy of partition is also what we should do in the future.

5 CONCLUSION AND FUTURE DIRECTIONS

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

There are several directions to go for the future work: (1) Since the nodes in three BA datasets used in this paper are none-attribute, we also need to find suitable datasets with a large number of nodes and node attributes to verify our model; (2) We should also introduce a mechanism to deal with edge attributes. In chemistry, not only atomic properties, but also bonds of a chemical compound are usually labeled, so it is useful to incorporate edge labels into our model; (3) Given the constraint that the exact GEDS for large graphs cannot be computed, we can only use approximate GED. When the number of graph nodes is larger, the approximation algorithm becomes less accurate. It would be interesting to see how the learned model generalize to larger graphs, which is trained only on the exact GEDS between partitioned subgraphs.

ACKNOWLEDGEMENTS

We thank Yunsheng Bai and Derek Xu for valuable discussions.

REFERENCES

[1] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. Reviews of modern physics, 74(1):47, 2002.

[2] Konstantin Andreev and Harald Racke. Balanced graph partitioning. Theory of Computing Systems, 39(6):929–939, 2006.

[3] Yunsheng Bai, Hao Ding, Song Biao, Ting Chen, Yizhou Sun, and Wei Wang. Simgnn: A neural network approach to fast graph similarity computation. In Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, pages 384–392, 2019.

[4] Yunsheng Bai, Hao Ding, Yizhou Sun, and Wei Wang. Convolutional set matching for graph similarity. arXiv preprint arXiv:1810.07066, 2018.

[5] David B Blumenthal and Johann Gamper. On the exact computation of the graph edit distance. Pattern Recognition Letters, 2018.

[6] Aydin Buoı, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. In Algorithm Engineering, pages 117–158. Springer, 2016.

[7] Horst Bunke. What is the distance between graphs. Bulletin of the EATCS, 20:35–39, 1983.

[8] Horst Bunke. On a relation between graph edit distance and maximum common subgraph. Pattern Recognition Letters, 18(8):689–694, 1997.

[9] Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. Pattern recognition letters, 193(4):255–259, 1998.

[10] Michael Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In Advances in neural information processing systems, pages 3844–3852, 2016.

[11] Stefan Funkhauser, Kaspar Riesen, and Horst Bunke. Speeding up graph edit distance computation through fast bipartite matching. In International Workshop on Graph-Based Representations in Pattern Recognition, pages 102–111. Springer, 2011.

[12] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In Advances in neural information processing systems, pages 1024–1034, 2017.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision, pages 1026–1034, 2015.

[14] Radu Horaud and Thomas Skordas. Stereo correspondence through feature grouping and maximal cliques. IEEE Transactions on Pattern Analysis and Machine Intelligence, 11(11):1168–1189, 1989.

[15] Hawoong Jeong, Zoltan Néda, and Albert-László Barabási. Measuring preferential attachment in evolving networks. EPL (Europhysics Letters), 61(4):567, 2003.

[16] Roy Jonker and Anton Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. Computing, 38(4):325–340, 1987.

[17] Maurice G Kendall. A new measure of rank correlation. Biometrika, 30(1/2):81–93, 1938.

[18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[19] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907, 2016.

[20] Harold W Kuhn. The hungarian method for the assignment problem. Naval research logistics quarterly, 2(1-2):83–97, 1955.

[21] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In Soviet physics doklady, volume 10, pages 707–710, 1966.

[22] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. arXiv preprint arXiv:1904.12787, 2019.

[23] Yongjiang Liang and Peixiang Zhao. Similarity search in graph databases: A multi-layered indexing approach. In 2017 IEEE 33rd International Conference on Data Engineering (ICDE), pages 783–794. IEEE, 2017.

[24] Michel Neuhaus, Kaspar Riesen, and Horst Bunke. Fast suboptimal algorithms for the computation of graph edit distance. In Joint IAFR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR), pages 163–172. Springer, 2006.

[25] Sankar P Pal and Sushmita Mitra. Multilayer perceptron, fuzzy sets, classification. 1992.

[26] Ferran Parés, Dario Garcia Gasulla, Armand Vilalta, Jonatan Moreno, Eduard Ayguadé, Jesús Labarta, Ulises Cortés, and Toyotaro Suzumura. Fluid communities: a competitive, scalable and diverse community detection algorithm. In International Conference on Complex Networks and their Applications, pages 229–240. Springer, 2017.

[27] Marcello Pelillo, Kaleem Siddiqi, and Steven W Zucker. Matching hierarchical structures using association graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence, 21(11):1105–1120, 1999.

[28] Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. Image and Vision computing, 27(7):950–959, 2009.

[29] Kaspar Riesen, Sandro Emmenegger, and Horst Bunke. A novel software toolkit for graph edit distance computation. In International Workshop on Graph-Based Representations in Pattern Recognition, pages 142–151. Springer, 2013.

[30] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. Journal of Machine
[31] Charles Spearman. The proof and measurement of association between two things. 1961.

[32] Xiaoli Wang, Xiaofeng Ding, Anthony KH Tung, Shanshan Ying, and Hai Jin. An efficient graph indexing method. In 2012 IEEE 28th International Conference on Data Engineering, pages 210–221. IEEE, 2012.

[33] Bu-Xiao Wu, Jing Xiao, and Jie-Min Chen. Friend recommendation by user similarity graph based on interest in social tagging systems. In International Conference on Intelligent Computing, pages 375–386. Springer, 2015.

[34] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? arXiv preprint arXiv:1810.00826, 2018.

[35] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 1365–1374, 2015.

[36] Zhiping Zeng, Anthony KH Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. Proceedings of the VLDB Endowment, 2(1):25–36, 2009.