More Interpretable Graph Similarity Computation via Maximum Common Subgraph Inference

Zixun Lan, Binjie Hong, Ye Ma, and Fei Ma

Abstract—Graph similarity measurement is a fundamental task in various graph-related applications. However, recent learning-based approaches lack interpretability as they directly transform interaction information between two graphs into a hidden vector, making it difficult to understand how the similarity score is derived. To address this issue, we propose an end-to-end paradigm for graph similarity learning called Similarity Computation via Maximum Common Subgraph Inference (INFMCS), which is more interpretable. Our key insight is that the similarity score has a strong correlation with the Maximum Common Subgraph (MCS). We implicitly infer the MCS to obtain the normalized MCS size, with only the similarity score being used as supervision information during training. To capture more global information, we stack vanilla transformer encoder layers with graph convolution layers and propose a novel permutation-invariant node Positional Encoding. Our entire model is simple yet effective. Comprehensive experiments demonstrate that INFMCS consistently outperforms state-of-the-art baselines for graph-graph classification and graph regression tasks. Ablation experiments verify the effectiveness of our proposed computation paradigm and other components. Additionally, visualization and statistical analysis of results demonstrate the interpretability of INFMCS.

Index Terms—Graph neural network, graph similarity learning.

I. INTRODUCTION

T he graph similarity measurement is a fundamental problem in various real-world applications, such as graph search in graph-based databases [1], malware detection [2], and brain data analysis [3]. Graph Edit Distance (GED) [4] and Maximum Common Subgraph (MCS) [5] are two popular graph similarity metrics. However, exact computation of both metrics is known to be NP-hard [6], and no algorithm has been able to compute the exact GED between graphs with more than 16 nodes within a reasonable time [7]. This has led to an increased interest in approximation algorithms, as well as a recent surge in graph similarity learning methods [8], [9], [10], [11], [12], [13].

Recent methods have improved performance on graph similarity prediction tasks by capturing node-level or subgraph-level interactions. The initial approach is to encode each graph as a fixed-length vector at the graph-level using Graph Neural Networks (GNNs), and then combine the vectors of the two input graphs to predict similarity. However, the actual difference between two graphs often arises from very small local substructures, making it difficult for a graph-level fixed-length vector to capture local information [9]. To alleviate this problem, GMN [8], MGN [11] and H2MN [12] derive node-level and graph-level embeddings containing interaction information of different scales through the cross-graph attention (propagation), and then convert these embeddings to one hidden vector (e.g., concatenation between two graph-level embeddings). SimGNN [9] and GraphSim [10] derive the corresponding hidden vector by applying the convolution operation to the pairwise node similarity matrix or extracting its histogram features respectively. Ultimately, all models map the hidden vector to the ground-truth similarity.

Despite incorporating interaction information between graphs, previous methods lack interpretability and fail to explain the meaning of the final hidden vector or how to relate it to the ground truth. Natural language processing research has demonstrated that models with greater interpretability typically achieve higher performance [14]. Attention mechanisms have been widely used in tasks like machine translation [15], language modeling [16], and abstractive summarization [17]. Attention provides a way to interpret the model’s inner workings [18], [19], [20], [21] and can improve its performance. In general, graph similarity learning models that adopt a more reasonable and interpretable paradigm can capture more critical information and filter out interference, thus outperforming less interpretable models.

To cope with this limitation, this study proposes a more interpretable end-to-end paradigm for graph similarity learning, named Similarity Computation via Maximum Common Subgraph Inference (INFMCS). Commonly, the more significant proportion of MCS to the average size of graph pairs [10], i.e. normalized MCS size (nMCS)\(^1\), the more similar the two graphs are. According to this fact, we infer Maximum Common Subgraph (MCS) implicitly and then obtain the normalized

---

\(^{1}\text{nMCS}(G_1, G_2) = \frac{\text{MCS}(G_1, G_2)}{|G_1| + |G_2|/2} \) with \( G_1 \) be the graph with smaller size. In this paper, we always view the graph with the smaller size as \( G_1 \), because the MCS size between two graphs is less than or equal to the size of the smaller graph.
MCS size in an end-to-end fashion. First, we perform message passing from graph $\mathcal{G}_2$ to graph $\mathcal{G}_1$ by the modified cross-graph attention mechanism, thereby obtaining $|\mathcal{G}_1|$ pairs of node-level embeddings. Here $|\mathcal{G}_1|$ is the size of graph $\mathcal{G}_1$. In each pair, one embedding represents one node in $\mathcal{G}_1$ and the another embedding represents one node in $\mathcal{G}_2$ most likely matching the former. After concatenating the two embeddings in the embedding pair, we use Multilayer Perceptron (MLP) to transform the concatenation to the matching score between zero and one, where one and zero represent that two nodes are matched and not matched respectively. Finally, we add up the $|\mathcal{G}_1|$ matching scores as the predicted MCS size and then normalize it to derive the predicted similarity score. The entire process is optimized by either GED/MCS normalized similarity or graph-classification classification label end-to-end. We also stack a few vanilla transformer encoder layers [22] with graph convolution layers to capture more global information, called Graph Convolution with Transformer (GCwT). Unlike the inherent order of sentences in natural language [22], graphs are permutation-invariant, resulting in no order for nodes. Thus we propose a novel Positional Encoding based on permutation-invariant node ordering. Comprehensive experiments demonstrate that INFMCS consistently outperforms state-of-the-art baselines for graph-graph classification and regression tasks. Ablation experiments verify the effectiveness of individual components, including the proposed graph similarity learning paradigm and GCwT with novel Positional Encoding. In brief, we highlight our main contributions as follows:

- We propose a more interpretable end-to-end paradigm for graph similarity learning. The interpretability is derived from inferring MCS implicitly.

- To our best knowledge, the novel Positional Encoding based on permutation-invariant node ordering is the first being proposed and used in the graph-transformer related model for graph similarity task. Like the order of tokens in sentences, permutation-invariant node ordering is also an essential graph feature.

- We perform comprehensive experiments on six graph-regression datasets and two graph-classification datasets to verify the effectiveness of the INFMCS. Ablation experiments verify the effectiveness of individual components. Also, the case study and visualization demonstrate interpretability.

II. RELATED WORK AND PRELIMINARIES

A. Graph Similarity Learning

Initial methods, such as SMPNN [23], GCNMEAN and GCNMAX [24] directly encode each graph as a graph-level fixed-length vector via GNNs and then only use graph-level interaction to predict similarity. After that, more models were proposed to exploit node-level or subgraph-level interactions by degrees. GMN [8] uses cross-graph attention to derive node-level embeddings that contain another graph’s information. SimGNN [9] and GraphSim [10] derive the corresponding hidden vector differently by applying the convolution operation to the pairwise node similarity matrix or extracting its histogram features. [13] first partitions graphs and then conducts node-wise comparison among subgraphs. MGMN [11] designs graph matching layers by comparing each node’s representation of one graph with the other whole graph representation. After converting graph to hypergraph via random walk or K-hop neighbourhood, H2MN [12] utilizes hypergraph convolution and subgraph matching blocks to predict similarity.

Recently, one interpretable method, GOTSim [25], was proposed. It formulates the similarity between a pair of graphs as the minimal “transformation” cost from one graph to another in the learnable node-embedding space. The interpretability of GOTSim derives from all node pairs’ injective mapping between the nodes of one graph to the other. However, the matching of the maximum common subgraph’s node pairs is more valuable than the matching of all node pairs for graph similarity calculation from the perspective of the ground-truth graph similarity label. Compared with GOTSim, our proposed INFMCS can calculate the maximum common subgraph implicitly in an end-to-end fashion, in order to increase interpretability and thus improve performance.

Compared with previous methods, our proposed INFMCS is simpler. INFMCS does not need to compute node-wise interactions per layer [8], [9] but only uses the last layer of node-level embeddings to capture cross-graph interactions. Second, ours does not need to consider multi-scale matching scores of $|\mathcal{G}_1| \times |\mathcal{G}_2|$ pairs [11], only calculates $|\mathcal{G}_1|$ matching scores. Third, ours does not need to preprocess graph data, such as graph partition [13] and hypergraph construction [12]. Although our method avoids the above computational burden, it achieves good performance.

B. Graph Convolution Network

We describe the computation process of one layer in GCN($\cdot$):

$$u_{i}^{n+1} \text{ReLU} \left( \sum_{j \in \mathcal{N}(i)} \frac{1}{\sqrt{d_i d_j}} u_{j}^{n} W^{(n)} + b^{(n)} \right). \quad (1)$$

Here, $u^{(n)}_{i} \in \mathbb{R}^{D(n)}$ and $u^{(n+1)}_{i} \in \mathbb{R}^{D(n+1)}$ are representations of node $i$ at $n$-th and $n + 1$-th layer, $\mathcal{N}(i)$ is the set of the first-order neighbors of node $i$ plus itself, $d_i$ is the degree of node $i$ plus 1, $W^{(n)} \in \mathbb{R}^{D(n) \times D(n+1)}$ is the weight matrix of the $n$-th GCN layer, $b^{(n)} \in \mathbb{R}^{D(n+1)}$ is the bias, $D(n)$ denotes the dimension of embedding vector at layer $n$. ReLU$(x) = \max(0, x)$ is the activation function.

C. Transformer Encoder Layer

Each Transformer Encoder Layer has two parts: a self-attention module and a position-wise feed-forward network (FFN):

$$Q^{h} = H^{(l)} W_{Q}^{h}, \quad K^{h} = H^{(l)} W_{K}^{h}, \quad V^{h} = H^{(l)} W_{V}^{h}, \quad (2)$$

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
Graph Convolution with Transformer (GCwT)

\[ A^h = \frac{Q^h H^h \top}{\sqrt{d_K}}, \quad H^h = \text{softmax}(A^h) V^h, \]
\[ H' = W' \cdot (\|h H^h\| + H^{(l)}) \]
\[ H^{(l+1)} = \text{FFN}(\text{LN}(H')) + H'. \]

Here, \( H^{(l)} = [h_1^{(l)}, \ldots, h_n^{(l)}] \top \in \mathbb{R}^{n \times d} \) denote the input of self-attention module where \( d \) is the hidden dimension and \( h_i^{(l)} \in \mathbb{R}^{1 \times d} \) is the hidden representation at position \( i \). The input \( H \) is projected by three matrices \( W_h^h \in \mathbb{R}^{d \times d_K}, W_k^h \in \mathbb{R}^{d \times d_K} \) and \( W_v^h \in \mathbb{R}^{d \times d_V} \) to the corresponding representations \( W_h^q, W_k^q, W_v^q \) for each head \( h \). \( A \) is a matrix capturing the similarity between queries and keys for each head \( h \) and \( || \) is concatenation operation. LN is layer normalization and FFN is the feed-forward network. For simplicity, we omit bias terms and assume \( d_K = d_V = d \).

III. MODEL DESIGN

Our approach follows the hypothesis that the more similar a pair of graphs is, the greater the ratio of MCS size between graphs to the pair’s average size is. To achieve this, we first derive \( |G_1| \) pairs of node embeddings via cross-graph attention and then transform them into matching scores. Ideally, the sum of \( |G_1| \) matching scores is precisely equal to the size of MCS. Finally, we normalize the sum of these scores to predict the similarity. We also proposed a Graph Convolution with Transformer based on permutation-invariant Positional Encoding to fill a gap between the shallow GCN and the sizeable receptive field. The overall process is end-to-end and is outlined in Fig. 1. Before describing the modules, we introduce relevant preliminaries that set the background for the remainder of the paper.

Graph Similarity Learning: Given a pair of input graphs \( (G_1, G_2) \), the aim of graph similarity learning is to produce a similarity score \( y = s(G_1, G_2) \in \mathcal{Y} \). The graph \( G_1 = (V_1, E_1) \) is represented as a set of \( N \) nodes \( v_i \in V_1 \) with a feature matrix \( X_1 \in \mathbb{R}^{N \times d} \) edges \( (v_i, v_e) \in E_1 \) forming an adjacency matrix \( A_1 \in \mathbb{R}^{N \times N} \). Here \( d \) is the dimension of node feature.

Similarly, the second graph \( G_2 = (V_2, E_2) \) can be represented in the same way.

A. Graph Convolution With Transformer

Before exploiting node-wise interactions, we need to obtain node-level embeddings as in the previous method. GCN [26] is the most popular spatial graph convolution. In this study, we use it to compute node-level embeddings. For simplicity, we denote the encoding process by GCN(\cdot) and describe architectural details in the Section II. The GCN computes node representations \( \mathbb{H} \in \mathbb{R}^{|V| \times d} \) via

\[ \mathbb{H} = [h_1; h_2; \cdots; h_{|V|}], h_i = \text{GCN}(G_i, [x_i, \{x_j\}_{j \in \mathcal{N}(i)}]), \]

where \( h_i \in \mathbb{R}^{1 \times d} \) and \( \mathcal{N}(i) \) denotes the node representation and neighbors of node \( v_i \) respectively. \( d \) is the dimension of hidden layer. \( x_i \in \mathbb{R}^{1 \times F_n} \) and \( x_j \in \mathbb{R}^{1 \times F_e} \) are original feature of node \( i \) and edge \( (i, j) \) respectively. \( F_n \) and \( F_e \) are the dimension of node and edge feature respectively. Graph similarity learning requires not only node embeddings to perceive local information but also node representations to contain global information due to many subtle differences across the whole graph [10], [12]. However, over-smoothing [26], [27] constrains graph convolution from stacking multiple layers, resulting in a gap between the shallow GCN and the sizeable receptive field.

To fill this gap, we stack some vanilla transformer encoder layers [22] with graph convolution layers to capture more global information. The deep perception ability of Transformer comes from the self-attention mechanism, which internally calculates the correlation between embeddings. The attention weight matrix is equivalent to constructing a fully connected graph, and then message passing is based on this fully connected structure. For sentence representation [22], extensive experiments show the importance of Positional Encoding. Sentences with the same tokens but in different order have different semantics (Fig. 2).
which shows that order is also an inherent feature of sentences. However, graphs are permutation-invariant, resulting in no order for nodes. Thus, we propose a permutation-invariant node ordering $C \in \mathbb{R}^{[|V|]}$ based on closeness centrality [28]:

$$C = \text{argtop}(|V|)\left\{c_1, \ldots, c_{|V|}\right\}, c_i = \frac{n - 1}{|V| - 1} \sum_{j=1}^{n-1} d(j, i),$$

(7)

where $c_i$ is the closeness centrality of node $v_i$ and $d(j, i)$ is the shortest path distance from node $j$ to node $i$. It is the reciprocal of the average shortest path distance to $v_i$ over all other nodes, and higher closeness values indicate higher centrality. Besides, $n$ is the number of nodes in the connected part of the graph containing the node $v_i$ and $\frac{n-1}{|V| - 1}$ is the proportion of this connected component in the entire graph. Hence, (7) can be generalized to graphs with more than one connected component, where the size of the connected component scales each node. argtop(|V|)(·) calculates the rank of each element in the vector in descending order. For example, argtop(|V|)([0.4, 0.6, 0.1, 0.9]) = [2, 1, 3, 0]. Our model also generalizes to edge representations. We convert the original graph to a line graph and then compute the permutation-invariant node order since the edges in the original graph construct the nodes in the line graph. Here, the line graph $L(G)$ of a given graph $G$ is defined as a new graph where the nodes in $L(G)$ correspond to the edges in $G$. For a pair of edges $(u, v)$ and $(v, w)$ in $G$, the node in $L(G)$ corresponding to the edge $(u, v)$ of $G$ will have an edge connecting to the node in $L(G)$ corresponding to the edge $(v, w)$ of $G$.

We denote the vanilla transformer encoder by TransformerEncoder(·) for simplicity and describe architectural details in the Section II. Given a learnable Positional Encoding dictionary $PE \in \mathbb{R}^{m \times d}$ ($m \gg |V|$), final node representations $H \in \mathbb{R}^{[|V|] \times d}$ is derived by (8):

$$H = \text{TransformerEncoder}(\mathcal{H}), \mathcal{H} = H + PE[C],$$

(8)

where $PE[C] \in \mathbb{R}^{[|V|] \times d}$ is Positional Encoding according to the node ordering $C$.

B. Inferring MCS Implicitly

Notably, we always view the graph with the smaller size as $G_1$, since the MCS size between two graphs is less than or equal to the size of the smaller graph in this paper. Given the node representations of the last layer of the graph representation learning $H_1 = [h_1^1; h_1^2; \ldots; h_{|V_1|}^1] \in \mathbb{R}^{[|V_1|] \times d}$ for $G_1$ and $H_2 = [h_2^1; h_2^2; \ldots; h_{|V_2|}^1] \in \mathbb{R}^{[|V_2|] \times d}$ for $G_2$, we pass the message from $G_2$ to $G_1$ by modified cross-graph attention $a_{ij}$, and then obtain the representation $h_i^1 \in \mathbb{R}^{1 \times d}$ of node $v_j \in V_2$ that most likely matches node $v_i \in V_1$:

$$a_{ij} = \frac{\exp(s_h(h_i^1, h_j^1) \times \tau_s^{-1})}{\sum_j \exp(s_h(h_i^1, h_j^1) \times \tau_s^{-1})}, h_i^1 = \sum_j a_{ij} h_j^1,$$

(9)

where $s_h$ is a vector space similarity metric, like euclidean or cosine similarity. In order to discretize $a_{ij}$, we add a learnable parameter $\tau_s \in (0, 1]$. In other words, it makes the weight $a_{ij}$ of one node $v_j \in V_2$ ($j = \text{argmax}_{j} a_{ij}$) tend to one and the others tend to zero due to $\sum_j a_{ij} = 1$. Thus, $h_i^1$ represents the representation of node corresponding to node $v_j$ with the highest probability.

After concatenating $h_i^1$ with $h_j^1$, we transform the concatenation to the matching score $s_i$ by Multilayer Perceptron (MLP).

Ideally, the sum of $|G_1|$ matching scores is precisely equal to the size of MCS. Finally, we normalize the sum of these predicted scores to compute similarity $\hat{y}$:

$$\hat{y} = \frac{\sum_{i=1}^{|G_1|} s_i}{(|G_1| + |G_2|)/2}, s_i = \text{sigmoid}(\text{MLP}(h_i^1 || h_j^1)).$$

(10)
The loss functions are defined as follows:

$$\mathcal{L}_c = -\frac{1}{|D|} \sum_{i=1}^{|D|} y_i \log (\hat{y}_i) + (1 - y_i) \log (1 - \hat{y}_i),$$

$$\mathcal{L}_r = \frac{1}{|D|} \sum_{i=1}^{|D|} (y_i - \hat{y}_i)^2,$$

where \( \mathcal{L}_c \) represents the binary cross-entropy loss for the graph-graph classification task and \( \mathcal{L}_r \) is the mean square error loss for the graph-graph regression task. \( y_i \) denotes the ground-truth supervision information, \( \hat{y}_i \) is the predicted graph similarity, and \(|D|\) is the size of the dataset.

IV. EVALUATION

In this section, we evaluate the performance of our INFMCS with comparison to recently proposed state-of-the-art approaches for both the graph-graph classification and graph-graph regression tasks, and with significant goals of addressing the following questions: Q1: How effective, efficient and robust is INFMCS compared to the state-of-the-art approaches under MCS/GED metric? Q2: How does the proposed similarity computation paradigm and GCW/T with permutation-invariant Positional Encoding improve performance? Q3: Does INFMCS have stronger interpretability?

A. Data

For graph-graph classification task, we use FFmpeg\(^3\) and OpenSSL\(^4\) [11] as datasets, where each graph denotes binary function’s control flow graph (CFG). Therefore, we take two CFGs compiled from the same source code as positive samples, i.e., \( s(G_1, G_2) = 1 \), and the dissimilar CFGs compiled from different source code, i.e., \( s(G_1, G_2) = 0 \). Moreover, we split each dataset into 3 sub-datasets according to the graph size in order to investigate the impact of graph size. For graph-graph regression task, we employ three real datasets and three synthetic datasets, including AIDS(2-15), LINUX(2-15), PTC_MM(all), BA100, BA200 and BA300. We extract graphs from the original datasets (AIDS [29], LINUX [30], PTC_MM [31])\(^5\) and construct the above three real datasets, where the values in parentheses indicate the size range of extracted graph. To verify the performance on large graphs, we also use the Barabási–Albert model [32] to generate synthetic graphs. We generated three synthetic datasets with graph sizes around 100, 200, and 300 respectively, called BA100, BA200 and BA300. Detailed descriptions and statistics of both real and synthetic datasets can be found in the following.

1) Real Datasets: For AIDS(2-15), LINUX(2-15), we randomly selected 1500 graphs whose size range from 2 to 15 in the AIDS [29] and LINUX [30]. For PTC_MM(all), we use the all graphs in the PTC_MM [31]. Each dataset is randomly split 80%, 10%, and 10% of all the graphs as original training set, original validation set, and original testing set, respectively. Then, each graph in the original set is paired with another graph as a sample. The datasets statistics for the regression task is shown in Table I.

For MCS metric, we use MCSR [33] to calculate the MCS for a pair of graphs and then normalize it to similarity score: \( n_{\text{MCS}}(G_1, G_2) = \frac{\text{MCS}(G_1, G_2)}{|G_1| + |G_2|} \). For GED metric, we use A* [34] to compute GED for graphs. For graphs that can not be computed by A* in 1000 seconds, we take the minimum distance computed by BEAM [35], HUNGRAN [36], VJ [37] and HED [38], since their returned GEDs are guaranteed to be upper bounds of the true GEDs. Fig. 3 presents the proportion of exact and approximate GED-labels.

For the classification task, we split FFmpeg and OpenSSL into 3 sub-datasets (i.e., [3, 200], [20, 200], and [50, 200]) according to the size ranges of pairs of input graphs. Table II shows the statistics of these datasets.

2) Synthetic Datasets: To verify the performance on large graphs, we also use the Barabási–Albert model [32] to generate synthetic graphs. We generated three synthetic datasets with graph sizes around 100, 200, and 300 respectively, called BA100, BA200 and BA300.

MCS metric We randomly generate a core graph by BA-model for each example and take this core as the primary graph. Next, we add edges and nodes to its surroundings randomly, thereby obtaining \( g_1 \) and \( g_2 \). We regard the core as the MCS and then obtain the similarity score. For instance, the size of the core graph is 60, and the number of added nodes for \( g_1 \) is 35, for \( g_2 \) is 45, thus the final label is 0.6. Although the size of the real MCS may be more than 60, it is the lower bound for this sample. The

\(^3\)https://ffmpeg.org/

\(^4\)https://www.openssl.org/

\(^5\)https://chrismrrs.github.io/datasets/docs/datasets/

| Table I |
|-----------------|-----------------|-----------------|-----------------|-----------------|
| | | | | |
| Statistics of AIDS, LINUX and PTC_MM. B (BILLION), M (MILLION) |
| # [G] | Avg. [V| | #Labels | # Graph Pairs |
| | | | | |
| AIDS(2-15) | 1500 | 10.22 | 20.39 | 38 | 225M |
| LINUX(2-15) | 1500 | 9.40 | 15.58 | 1 | 225M |
| PTC MM(all) | 336 | 13.97 | 28.64 | 20 | -112M |

| Table II |
|-----------------|-----------------|-----------------|-----------------|-----------------|
| | | | | |
| Statistics of FFmpeg and OpenSSL. B (BILLION), M (MILLION) |
| | | | | |
| Subsets | # [G] | Avg. [V] | Avg. [E] | # Pairs |
| | | | | |
| FFmpeg | | | | |
| [3, 200] | 83008 | 18.83 | 27.02 | 6.89B |
| [20, 200] | 31696 | 51.02 | 75.88 | 1B |
| [50, 200] | 10824 | 90.93 | 136.83 | 117M |
| OpenSSL | | | | |
| [3, 200] | 73953 | 15.73 | 21.97 | 5.46B |
| [20, 200] | 15800 | 44.89 | 67.15 | 249M |
| [50, 200] | 4308 | 83.68 | 127.75 | 18.5M |
for the range of the number of added nodes : $\rho \times 1$ nodes $g$ as the results for each baseline. We also use five classical algorithms as baselines to compare running time, including A* [34], MCSPLIT [33], BEAM [35], HUNGARIAN [36], VJ [37] and HED [38].

Algorithm 1: Generate BA Graphs for MCS Metric.

**Input:** $(c_1, c_2)$ for the range of the number of core graph nodes; $(a_1, a_2)$ for the range of the number of added graph nodes; $n$ for the number of samples

**Output:** BA datasets in MCS metric

data $\leftarrow \{\}$;

for $i$ in range($n$) do
    $c \leftarrow$ random($c_1, c_2$);
    $a_1 \leftarrow$ random($a_1, a_2$);
    $a_2 \leftarrow$ random($a_1, a_2$);
    // use BA-model to generate the core graph with $c$ nodes
    core $\leftarrow$ barabasi-albert_graph($c$);
    // use BA-model to generate two graphs based on the core graph, and they have $c + a$ nodes
    $g_1' \leftarrow$ barabasi-albert_graph($c + a_1$, core);
    $g_2' \leftarrow$ barabasi-albert_graph($c + a_2$, core);
    sim $\leftarrow c/(c + 0.5 \times (a_1 + a_2));$
    $(g_1', g_2', \text{sim}) || \text{data;}
end for

range of core graph size and added nodes are shown in Table III. The pseudocode is shown in the Algorithm 1.

GED metric First, we generated two basic graphs by BA-model. To each of them, it is edited by $c$.

Algorithm 2: Search the number of Transformer layers $L$ in Table IV. The pseudocode is shown in the Algorithm 2.
We operate the training AUC [12].

First Notably [13] [9] [10] [8] [7] [6] [5] [4] [3] [2] [1]. We show the running time between different methods in Fig. 5. Figure 5 shows the running time comparisons.

Algorithm 2: Generate BA Graphs for GED Metric.

Input: b for the number of basic graph nodes; n for the number of generated samples to store in one collection

Output: BA datasets in GED metric

data ← { };
ged ← 0;
A ← { };
B ← { };

Generate two collections of graphs;
// use BA-model to generate two basic graphs with b nodes
b1 ← barabasi_albert_graph(b);
b2 ← barabasi_albert_graph(b);

for i in range(n) do
    every 10 iterations, increase ged
    if i % 10 = 0 then
        ged ← ged + 1;
    end if
    // the trimming methods are (1) deleting a leaf node and its edge (2) adding a leaf node and (3) adding an edge, which have the same possibility to operate for each time
    // trim_ged_times function(graph, num_trim). Num_trim denotes the number of trim.
    g1 ← trim_ged_times(b1, ged);
g2 ← trim_ged_times(b2, ged);

{{g1, ged}} ||| A;
{{g2, ged}} ||| B;
end for

Pack and label graphs;
C ← A ||| B

for i in range(2n) do
    for j in range(2n) do
        (g1, ged1) ← C[i];
        (g2, ged2) ← C[j];
        bench_ged ← min(BEAM, HUNGARIAN, VJ, HED)
        ged ← min(ged1 + ged2, bench_ged)
        sim ← exp(-(ged)^2/2);
        {{g1, g2, sim}} ||| data;
    end for
end for

TABLE IV
STATISTICS OF BA-GRAPH FOR GED METRIC

| BA     | # Basic Graph | # Nodes | # Train | # Valid | # Test |
|--------|---------------|---------|---------|---------|--------|
| BA100  | 2             | 32000   | 4000    | 4000    | 4000   |
| BA200  | 2             | 20000   | 4000    | 4000    | 4000   |
| BA300  | 2             | 30000   | 4000    | 4000    | 4000   |

the dimension of hidden layers \( d \in \{128, 256, 512\} \), where the dimension of all hidden layers is set to the same. More details about the hyper-parameter search can be found in the appendix, available online. We conduct all the experiments on a machine with an Intel Xeon 4114 CPU and two Nvidia Titan GPUs. As for training, we use the Adam algorithm for optimization [43] and fix the initial learning rate to 0.001. The proposed model is trained on real datasets for 100 epochs with a batch size of 128 and on synthetic datasets for 30 epochs with a batch size of 32. Checkpoints are saved for each epoch to select the best checkpoints on the evaluation set. The source code can be found in the supplementary materials.

E. Overall Performance

Graph-Graph Classification Task: We operate the training process five times and report the mean and standard deviation in AUC. Our method is straightforward and achieves state-of-the-art performance on both datasets under all settings. The graph-graph classification performance is illustrated in Table V. We have two observations. First, compared with GMN [8], SimGNN [9] and GraphSim [10], our method obtains relative gains around 5%. It indicates that our method makes better use of node-wise interactions. Second, compared with PSimGNN [13] and H2MN [12], our method obtains relative gains of around 2%. It implies that the improvement of graph representation ability can benefit experimental results. A more detailed analysis can be found in the ablation study. Notably, we only exploit node-wise interactions at the last layer, while the previous method exploits the interaction information at each layer. It shows that our computational paradigm is simpler and has less complexity. Moreover, the construction process of the classification label is inconsistent with the internal logic of our method. It demonstrates that our method is robust even if the labels are MCS-independent.

Graph-Graph Regression Task: For the graph-graph regression task, we also conduct the experiments five times and report their mean performance. The detailed performances on real and synthetic datasets are demonstrated in Table VI, VII and VIII. Our model achieves state-of-the-art performance on both MCS and GED metrics. In general, we can obtain similar conclusions as to the classification task. As for synthetic datasets, we observe that our method still outperforms other methods. It shows that our method can be adapted to bigger graphs. Notably, the results of our method on the MCS metric are about eight times better than those on the GED metric. We infer that this situation results from the model’s internal logic consistent with the MCS label. The MSE of the model prediction results is close to zero, which means we can infer a more accurate MCS based on the average classification label is consistent with the internal logic of our method. It demonstrates that our computational paradigm is simpler and has less complexity. Moreover, the construction process of the classification label is inconsistent with the internal logic of our method. It shows that our method is robust even if the labels are MCS-independent.

Efficiency: We show the running time between different methods in Fig. 5 in order to evaluate the efficiency of INFMCs. As we can see, the learning-based approaches are consistently
TABLE V
GRAPH-GRAPH CLASSIFICATION RESULTS (AUC SCORE) WITH STANDARD DEVIATION (IN PERCENTAGE)

| Datasets | FFmpeg (5, 200) | OpenSSL (5, 200) | FFmpeg (5, 200) | OpenSSL (5, 200) |
|----------|----------------|-----------------|----------------|----------------|
| SimGNN   | 95.38±0.76     | 94.32±1.01     | 93.45±0.54     | 93.96±0.31     |
| GMN      | 94.15±0.62     | 95.92±1.38     | 94.76±0.45     | 96.43±0.61     |
| GraphSim | 97.46±0.30     | 96.69±0.28     | 94.48±0.73     | 96.84±0.54     |
| MGMM     | 98.07±0.06     | 98.29±0.10     | 97.83±0.11     | 96.90±0.10     |
| PSimGNN  | 96.77±0.54     | 96.86±0.95     | 95.23±0.15     | 96.10±0.46     |
| GOTSIM   | 96.93±0.34     | 97.01±0.52     | 95.65±0.31     | 97.87±0.49     |
| H2MN     | 98.28±0.20     | 98.54±0.14     | 98.30±0.29     | 98.27±0.16     |
| INFMC   | 98.49±0.09     | 99.36±0.13     | 99.48±0.20     | 99.14±0.31     |

TABLE VI
GRAPH-GRAPH REGRESSION RESULTS ABOUT MSE (×10−2), ρ AND P@10 ON THE MCS METRIC

| Datasets | AODS(2-15) | LINUX(2-15) | PTC_MMM(all) |
|----------|------------|-------------|--------------|
| Metrics  | mse         | ρ           | p@10         | mse         | ρ           | p@10         |
| INFMC    | 0.30        | 0.3952      | 0.7976       | 0.02        | 0.9814      | 0.8870       |

TABLE VII
GRAPH-GRAPH REGRESSION RESULTS ABOUT MSE (×10−2), ρ AND P@10 ON THE GED METRIC

| Datasets | AODS(2-15) | LINUX(2-15) | PTC_MMM(all) |
|----------|------------|-------------|--------------|
| Metrics  | mse         | ρ           | p@10         | mse         | ρ           | p@10         |
| INFMC    | 0.04        | 0.4090      | 0.1854       | 0.04        | 0.4054      | 0.3445       |

TABLE VIII
GRAPH-GRAPH REGRESSION RESULTS ABOUT MSE (×10−2) ON THE SYNTHETIC DATASETS

| Datasets | BA100 | BA200 | BA300 |
|----------|-------|-------|-------|
| Metric   | mse(MCS) | mse(GED) | mse(MCS) | mse(GED) | mse(MCS) | mse(GED) |
| INFMC    | 5.49e-7 | 0.061 | 1.80e-5 | 0.011 | 0.003 | 0.005 |

 faster than the traditional methods in all datasets, especially the exact algorithm MCSPLIT. We also observe that INFMC is faster than other learning-based approaches. We attribute the efficiency gains to two points. First, unlike previous methods that exploit the interaction information of each layer, our method only needs to compute the interaction information of the last layer. Second, our method does not require building hypergraphs or graph partitions.

Hyperparameter sensitivity analysis: We fixed the number of Transformer layers $L$ to 8 and the dimension of hidden layers $d$ to 256 to explore the impact of several vital hyper-parameters on OpenSSL subsets (Fig. 6). The performance under different hyperparameters is consistent, revealing the robustness of our method. We observe that the performance of INFMC improves as the number of Transformer layers increases. We hypothesise that more Transformer layers allow node-level embeddings to
TABLE IX
NUMBER OF PARAMETERS

| Model          | BMBAVG | GMN | GraphSim | SimGNN | SMPPNN |
|----------------|--------|-----|----------|--------|--------|
| Number of Parameters | 26K    | 62K | 17K      | 2M     | 13K    |

TABLE X
RESULTS OF HYPERPARAMETER SEARCH

| FFmpeg | OpenSSL |
|--------|---------|
| t     | d      | [3, 200] | [20, 200] | [50, 200] | [3, 200] | [20, 200] | [50, 200] |
| 8   | 8   | 8   | 8   | 8   | 2   | 6   | 8   |
| 5   | 8   | 5   | 8   | 5   | 8   | 5   | 8   |
| AIDLS | LINUX | PTC-MM | BA100 | BA200 | BA300 | AIDLS | LINUX | PTC-MM | BA100 | BA200 | BA300 |
| 128  | 128  | 128  | 128  | 128  | 128  | 128  | 128  | 128  | 128  | 128  | 128  |
| 128  | 128  | 128  | 128  | 128  | 128  | 128  | 128  | 128  | 128  | 128  | 128  |

TABLE XI
SCOPE OF HYPERPARAMETER SEARCH

| Parameter          | Values                                      |
|--------------------|---------------------------------------------|
| # GCN layer        | 128, 256, 512                               |
| GCN hidden dimension | 8                                        |
| # Transformer head | 2, 4, 6, 8                                  |
| # Transformer layer | 128 (regression) / 32 (classification)      |
| Learning rate      | 0.001                                       |
| # Epoch            | 100 (regression) / 30 (classification)      |

TABLE XII
ABLATION STUDY ON THE MCS METRIC

| Model          | (msec × 10^-4) | AIDS | LINUX | PTC-MM |
|----------------|----------------|------|-------|--------|
| H2MN-H         | 1.63 0.36 1.18 |      |       |        |
| BASE           | 1.41 0.36 0.98 |      |       |        |
| BASE+T         | 3.21 0.93 1.24 |      |       |        |
| BASE+H         | 1.70 0.21 1.02 |      |       |        |
| BASE+T+PE_outr | 0.30 0.02 0.71 |      |       |        |
| BASE+T+PErw   | 0.56 0.04 0.79 |      |       |        |
| BASE+T+PEnp   | 0.84 0.14 0.91 |      |       |        |
| BASE+T+PEnp   | 1.12 0.43 1.02 |      |       |        |

The comparison of the number of parameters of our method with other baselines can be found in Table IX. For all datasets, the GCN learning layer is fixed to 3, and the transformer encoder head number is set to 8; The scope of the hyperparameter search is shown in the Table XI. The results of the hyperparameter search are shown in Table X.

F. Ablation Study

BASE6 denotes the proposed similarity computation paradigm whose graph representation model (GRM) is GCN. BASE+T’s GRM is GCwT without the permutation-invariant Positional Encoding. BASE+T+PE’s GRM is GCwT with our proposed permutation-invariant Positional Encoding. BASE+H’s GRM is hypergraph convolution used in [12]. We use the part of their code7 to obtain node embeddings and then pass these to our paradigm end-to-end.8 H2MN-H denotes that the H2MN model’s hyperparameter k is set to 1, indicating that hypergraph convolution degenerates to GCN. The results of the ablation study is illustrated in Table XII, and on which we have the four observations: 1) BASE outperforms previous methods except for H2MN, thus we compare BASE with H2MN-H in order to demonstrate the effectiveness of the computation paradigm. We find BASE outperforms H2MN-H, which means more interpretable forward computation can improve performance. 2) BASE+T has lower performance than BASE. It indicates that the performance of GCwT without the permutation-invariant Positional Encoding degrades. We attribute the reason to the loss of graph structural information since stacking Transformer layers only is equivalent to treating the graph as a fully connected graph. 3) BASE+T+PE_outr outperforms BASE, which confirms that GCwT with the permutation-invariant Positional Encoding not only increases the receptive field of the node but also learns the structural information through PE. More on Positional Encoding analysis can be found in the next subsection. 4) BASE+T+PE_outr outperforms BASE+H_outr and BASE+H_lp, implying that GCwT can be better adapted to the proposed computation paradigm and have decent global representation.

We also compared different Positional Encodings, such as Random Walk Positional Encoding PE_rw [44] and Laplacian Positional Encoding PE_lp [45]. In Table XII, we find BASE+T+PE_outr outperforms BASE+T+PE_rw and BASE+T+PE_lp, implying that our proposed Positional Encoding is more suitable for graph similarity calculation Task.

G. Intepratability Analysis, Case Study and Discussion

Positional Encoding: We reduce the Positional Encoding trained on AIDS(2-15) to two dimensions via PCA and present them in Fig. 8, where each red point denotes a Positional

6MCS-RSC is MCS-Related Similarity Computation. BASE = GCN + MCS-RSC; BASE+T = GCwT-w/-PE + MCS-RSC; BASE+T+PE = GCwT-w/-PE + MCS-RSC; BASE+H = HyperGCN + MCS-RSC; H2MN-H = GCN + H2MN-RSC.

7https://github.com/cszhangzhen/H2MN

8We use the random walk to construct the hypergraph and set hyperparameter k to 5.
Fig. 7. Sample of ranking results under the MCS metric on AIDS, LINUX, and PTC_MM datasets. In each section, each query graph (first column) is presented with a ranked list of graphs. The first row shows the graphs returned by our model in each demo, with the predicted similarity for each graph shown at the top. The bottom row describes the graphs returned by MCSPLIT.

Fig. 8. Positional Encoding analysis.

Fig. 9. Visualizations of inferring MCS.

Fig. 10. Interpretability analysis.

Encoding. Since the size of most samples is no more than 12, embeddings after position 12 are insufficiently trained. We present the top 12 Positional Encoding embeddings and obtain an exciting observation. The euclidean distance between Positional Encoding 0 and the following Positional Encoding gradually increases. The euclidean distance here corresponds to each node’s centrality in the graph, which means the Positional Encoding embeddings preserve the graph’s structural information well.

Infer MCS: During inference, we infer the MCS size \( m \) via multiplying the average size of the graph pair by the similarity score. Next, we extract a subgraph from \( G_1 \) that consists of nodes corresponding to the top \( m \) matching scores \( s \) in (10). Also, we use MCSPLIT to extract the true MCS. To evaluate the quality of our predicted MCS, we compute the similarity between the predicted and the actual MCS. The similarity results and some visualizations are presented in Figs. 9 and 10, where red subgraphs are MCS between the predicted and the actual MCS. The mean absolute error between the predicted and actual sizes is larger on the GED metric. We attribute the reason to the inconsistency in the model logic and the label construction process. However, we note that the similarity between the predicted MCS and the ground-truth MCS is higher than 0.8 on both metrics except for the AIDS (MCS) results, revealing the interpretability of our approach. We can predict the similarity score and infer MCS to understand why this similarity score is predicted.

Case Study: We demonstrate three example queries, one from each dataset in Fig. 7. The first row shows the graphs returned by our model in each demo, with the predicted similarity for each graph shown at the top. The bottom row describes the graphs returned by MCSPLIT. Notably, the top 5 results are precisely the isomorphic graphs to the query in the case of LINUX and PTC_MM.

Why is INFMCS effective? For the computation paradigm, our method ultimately only needs to consider \( n \) pairs of interaction information, which means it captures more critical information that affects the similarity compared to the previous methods. Since Positional Encoding well preserves the positional information of nodes, the interaction within the model compares the similarity of local structures between nodes and considers the relative position between nodes in two graphs. Intuitively, the more similar the two graphs are, the more similar their corresponding local regions should be.

The INFMCS method infers MCS (Maximum Common Subgraph) without explicit supervision through a series of steps in its
We analyze the time complexity of our proposed INFMCS. Here, $l_1$ denotes the number of GCN layers, $l_2$ denotes the number of Transformer Encoder layers, $d$ represents the hidden dimension and $h$ is number of heads in Transformer. First, the complexity of GCN is $O(l_1|\mathcal{G}||\mathcal{V}|d^2)$. Second, closeness centrality requires the time of $O(|\mathcal{V}|^2 \log|\mathcal{V}|)$. The complexity of Transformer is $O(l_2|\mathcal{V}|^2dh)$. Third, the complexity of cross-propagation is $O(|\mathcal{V}|_1||\mathcal{V}|_2)$.

V. CONCLUSION

This paper proposes a more interpretable end-to-end paradigm for graph similarity learning, whose interpretable computation process improves the performance of graph similarity learning. The model can implicitly infer Maximum Common Subgraph during inference. We stack some vanilla transformer encoder layers with graph convolution layers and propose a novel permutation-invariant node Positional Encoding to capture more global information. Comprehensive experiments and ablation studies demonstrate that INFMCS outperforms previous methods and is more interpretable.

REFERENCES

[1] X. Yan and J. Han, “gSpan: Graph-based substructure pattern mining,” in Proc. IEEE Int. Conf. Data Mining, 2002, pp. 721–724.
[2] S. Wang et al., “Heterogeneous graph matching networks,” 2019, arXiv: 1908.08074.
[3] J. Xi, D. Sun, C. Chang, S. Zhou, and Q. Huang, “An omics-to-omics joint knowledge association subtensor model for radiogenomics cross-modal modules from genomics and ultrasonic images of breast cancers,” Comput. Biol. Med., vol. 155, 2023, Art. no. 106672.
[4] H. Bunke, “What is the distance between graphs,” Bull. EATCS, vol. 20, pp. 35–39, 1983.
[5] H. Bunke and K. Shearer, “A graph distance metric based on the maximal common subgraph,” Pattern Recognit. Lett., vol. 19, no. 3/4, pp. 255–259, 1998.
[6] Z. Zeng, A. K. Tung, J. Wang, J. Feng, and L. Zhou, “Comparing stars: On approximating graph edit distance,” Proc. VLDB Endowment, vol. 2, no. 1, pp. 25–36, 2009.
[7] D. B. Blumenthal and J. Gamper, “On the exact computation of the graph edit distance,” Pattern Recognit. Lett., vol. 134, pp. 46–57, 2020.
[8] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, “Graph matching networks for learning the similarity of graph structured objects,” in Proc. Int. Conf. Mach. Learn., PMLR, 2019, pp. 3835–3845.
[9] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, and W. Wang, “SimGNN: A neural network approach to fast graph similarity computation,” in Proc. 12th ACM Int. Conf. Web Search Data Mining, 2019, pp. 384–392.
[10] Y. Bai, H. Ding, K. Gu, Y. Sun, and W. Wang, “Learning-based efficient graph similarity computation via multi-scale convolutional set matching,” in Proc. AAAI Conf. Artif. Intell., 2020, pp. 3219–3226.
[11] X. Ling et al., “Multilevel graph matching networks for deep graph similarity learning,” IEEE Trans. Neural Netw. Learn. Syst., vol. 34, no. 2, pp. 799–813, Feb. 2023.
[12] Z. Zhang et al., “H2MN: Graph similarity learning with hierarchical hypergraph matching networks,” in Proc. 27th ACM SIGKDD Conf. Knowl. Discov. Data Mining, 2021, pp. 2274–2284.
[13] H. Xu et al., “Graph partitioning and graph neural network based hierarchical graph matching for graph similarity computation,” Neurocomputing, vol. 439, pp. 348–362, 2021.
[14] V. C. Vashishth, S. Upadhyay, G. S. Tomar, and M. Faruqui, “Attention interpretability across NLP tasks,” 2019, arXiv: 1909.11218.
[15] M.-T. Luong, H. Pham, and C. D. Manning, “Effective approaches to attention-based neural machine translation,” 2015, arXiv: 1508.04025.
[16] Y. Liu and M. Lapata, “Learning structured text representations,” Trans. Assoc. Comput. Linguistics, vol. 6, pp. 63–75, 2018.
[17] Y. Ma, Z. Lan, L. Zong, and K. Huang, “Global-aware beam search for neural abstractive summarization,” in Proc. Adv. Neural Inf. Process. Syst., 2021, pp. 16545–16557.
[18] J. Xi et al., “Knowledge tensor embedding framework with association enhancement for breast ultrasound diagnosis of labeled samples,” Neurocomputing, vol. 468, pp. 60–70, 2022.
[19] R. Ghaeini, X. Z. Fern, and P. Tadepalli, “Interpreting recurrent and attention-based neural models: A case study on natural language inference,” 2018, arXiv: 1808.03894.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
