NODE NUMBER AWARENESS REPRESENTATION FOR GRAPH SIMILARITY LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

This work aims to address two important issues in the real-time graph similarity computation, one is **Node Number Awareness Issue (N^2AI)**, and the other is how to accelerate the inference speed of graph similarity computation in a real-time downstream task. We found that existing Graph Neural Network based graph similarity models have a large error in predicting the similarity scores of two graphs with similar number of nodes. Our analysis shows that this is because the global pooling function in graph neural networks maps graphs with similar numbers of nodes to similar embedding distributions, masking the differences between the two graphs and thus reducing the performance of the models, which we refer to as the **N^2AI**. Our motivation is to enhance the difference between the two embeddings, thus we leverage our proposed **Different Attention (DiffAtt)** to construct our **Node Number Awareness Graph Similarity Model (N^2AGim)**. In addition, we propose the **Graph Similarity Learning with Landmarks (GSL^2)** to accelerate similarity computation in real-time downstream tasks. GSL^2 uses the trained N^2AGim to generate the individual embedding for each graph without any additional learning, and this individual embedding can effectively help GSL^2 to improve its inference speed in real-time scenes. Experiments in real-world datasets demonstrate that our N^2AGim outperforms the second place on Mean Square Error by 24.3%(1.170 vs 1.546), 43.1%(0.066 vs 0.116), and 44.3%(0.308 vs 0.553) on the three datasets, and that our GSL^2 is at most 47.7 and 1.25 times faster than N^2AGim and the second fast model, respectively.

1 INTRODUCTION

Graph similarity computation is a fundamental problem for graph-based applications, e.g., graph data mining, graph retrieval, and graph clustering (Kriege et al., 2020; Ok & Korea, 2020). **Graph Edit Distance (GED)**, which is defined as the least number of graph edit operators to transform graph G_i to graph G_j, is one of the most popular graph similarity metrics (Gao et al., 2010; Neuhaus et al., 2006; Bougleux et al., 2015). The graph edit operators are insert or delete a node/edge, or relabel an edge. Unfortunately, the exact GED computation is NP-Hard in general (Zeng et al., 2009), which is too expensive to leverage in the real-time downstream tasks.

Thanks to the development of the Graph Neural Networks (GNNs), recently many GNN-based graph similarity computation algorithms have been proposed to compute the GED in a faster manner (Bai et al., 2019; 2020; Li et al., 2019; Ling et al., 2021; Bai & Zhao, 2021; Wang et al., 2021). The GNN-based algorithms transform the GED value to a similarity score and uses an end-to-end framework to learn how to map the given two graphs to their similarity score. As a general framework, the Siamese neural network can be used to aggregate information on each graph, while the feature fusion module can be used to capture the similarity between them, and the Multi-layer Perceptron (MLP) is then leveraged for the regression.

However, we found that the existing popular graph similarity models become very inaccurate in predicting the similarity of two graphs with similar number of nodes, which is shown in Fig 1. In order to better understand this issue, we presented a theoretical analysis of the widely used modules in the graph similarity models from a statistical view. As shown in Fig 2 (a)-(e), our conclusion is that all global pooling functions, also called graph readout functions, map graphs with similar numbers of nodes to similar embeddings. However, in a realistic scenario, two graphs with similar

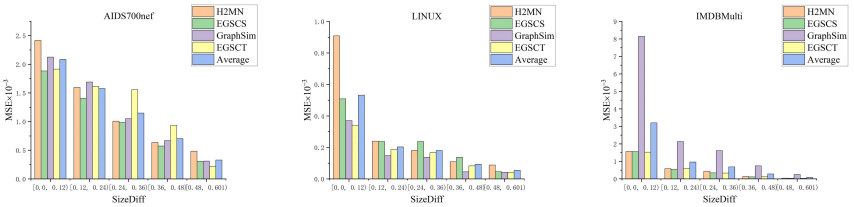


Figure 1: Histogram of the Mean Square Error (MSE) of the existing graph similarity models on three datasets at different level of SizeDiff. The SizeDiff is defined as $SizeDiff(G_1, G_2) = \frac{|N_1 - N_2|}{max(N_1, N_2)}$, where N_i is the number of nodes in G_i. It is clear that all models have a larger MSE when the SizeDiff is smaller, i.e. when the number of nodes in the graph pair is similar. See Fig 6 in Appendix A.4 for more details.

numbers of nodes may have completely different structures, e.g. with a very different number of edges, but these generated similar embeddings mask the differences between these graphs, resulting in the low accuracy of the GNN-based graph similarity models when predicting the similarity scores of graph pairs with similar numbers of nodes. We refer to this issue of indistinguishable embeddings of graphs with similar numbers of nodes as the **Node Number Awareness Issue (N^{2}AI)**.

Our motivation to address the N^{2}AI is to focus more on the differences between two similar embeddings in the learning process, and we propose the **Different Attention (DiffAtt)** to construct our **Node Number Awareness Graph Simialrity Model (N^{2}AGim)**. Our DiffAtt is simple in architecture, and can be added as a plug-and-play module to any global pooling methods. The evaluations on three real world datasets demonstrate that the models with different pooling methods achieve a significant improvement after using DiffAtt. Moreover, our N^{2}AGim achieves state-of-the-art performance compared to the popular GNN-based graph similarity models.

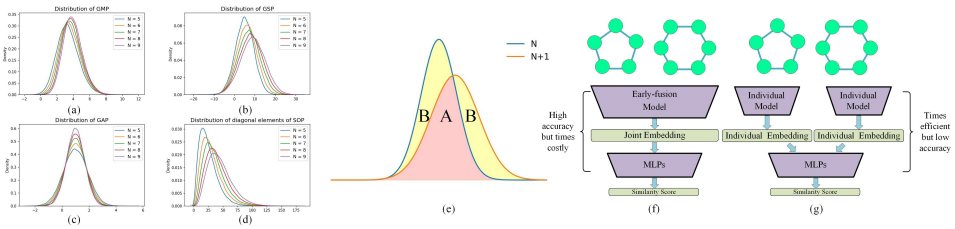


Figure 2: (a)-(d) Distribution of output from different global pooling functions with N nodes, which shows that all global pooling functions map graphs with similar numbers of nodes to similar distributions. See Section 3 for details. (e) Illustration of the N^{2}AI, i.e., the distribution of the embeddings of two graphs with similar numbers of nodes is indistinguishable. Region A represents where two distributions overlap, while B is the opposite. Our aim is to enhance the information in B to address the N^{2}AI. (e)-(f) Illustration of the Early Fusion Model and Individual Embedding Model.

Another issue of interest in the field of graph similarity learning is how to accelerate the inference speed of graph similarity models in real-time downstream tasks (Qin et al., 2021). The graph similarity models can be divided into two categories, one is the Early Fusion Model (EFM), shown in Fig 2(f), which performs feature fusion at an early stage to achieve high accuracy but slow inference, and one is the Individual Embedding Model (IEM), shown in Fig 2 (g), which generates an individual embedding for each graph and then performs fusion, which is fast but low accuracy. The existing method (Qin et al., 2021) uses a special designed Knowledge Distillation (KD) paradigm to leverage a EFM teacher to improve the individual embeddings generaged by the IEM student. However, motivated by Balcan et al. (2008), we propose a faster and more accurate IEM called **Graph Similarity Learning with Landmarks (GSL^2)** . In GSL^2, a subset of graphs, called **landmarks** $\mathcal{S}$, are selected, and then each graph G is represented as a vector as $\boldsymbol{u}_G = [GED(G, \hat{G}_1), \cdots, GED(G, \hat{G}_m)]^T$, where $\hat{G} \in \mathcal{S}$. Last, a MLP is learned to map the concatenation of the embeddings of two graphs to their GED target. Instead of learning the embeddings on the graph data, our GSL^2 uses an already trained graph similarity model to directly generate an individual embedding for each graph. This individual embedding not only can accelerate the training and inference of the model, but also has an upper limit on the ability to predict the exact GED value. To sum up, the contributions of this paper can be summarized as follows:

- We found that the existing graph similarity models have a relatively large error in predicting the similarity of two graphs with similar numbers of nodes, because the global pooling

function will map graphs with the similar numbers of nodes into two distributions that are similar, which we refer to as **N^{2}AI**, thus masking the difference information in the graphs and reducing the performance of the graph similarity learning.
- A novel GNN-based graph similarity model, named **N^{2}AGim**, is proposed. our N^{2}AGim achieves excellent results in the graph similarity learning task by leverage the proposed **DiffAtt** to effectively address the N^{2}AI.
- In order to address the real time downstream tasks, we propose the **GSL^2**. The GSL^2 directly represents each graph as a vector, where each component is the GED value between the graph and a landmark, and then learns the target GED values based on these graph representations.
- Experimental results on real-world benchmarks show that our N^{2}AGim achieves the state-of-the-art performance, while our GSL^2 achieves a good performance of accuracy and inference speed to efficiently handle real-time downstream tasks.

2 PRELIMINARIES

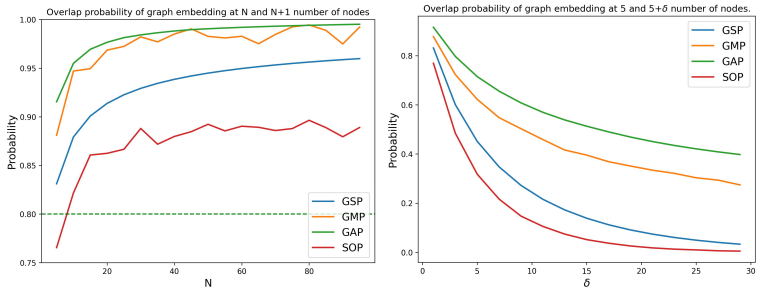
2.1 GRAPH NEURAL NETWORK

Graph data $\mathcal{G}$ can be viewed as a pair of adjacency matrix $A \in \{0,1\}^{N\times N}$ and a node feature matrix $X \in \mathbb{R}^{N\times C}$. N is the number of nodes in a graph, and C is the dimension of the initial node features. Node i and node j have an edge if and only if $A_{i,j} = 1$. Considering $X = [x_1, x_2, ..., x_N]^T$, a Message Passing Neural Network (MPNN) layer is defined as (Fey & Lenssen, 2019) : $x_i^{(k)} = \gamma^{(k)}(x_i^{(k-1)}, \square_{j\in\mathcal{N}(i)}\phi^{(k)}(x_i^{(k-1)}, x_j^{(k-1)}, e_{i,j}))$, where $x_i^{(k)} \in \mathbb{R}^{C_k}$ is an embedding of node i at the kth layer, and ϕ performs a differentiable transform on each node or edge, such as a MLP. $\square$ is an aggregate function to aggregate the transformed attributes of nodes and their neighbors, e.g., sum pooling, max pooling or average pooling. $\mathcal{N}(i)$ denotes the neighbors of node i and $e_{i,j}$ is the edge feature from node i to node j. γ is an activation function or other differentiable function to update the node embeddings.

Following the idea of MPNN, several GNNs and their variants have been proposed to deal with the graph mining tasks, e.g., Graph Convolutional Network (GCN) (Kipf & Welling, 2016), GraphSAGE (Hamilton et al., 2017) and Graph Attention Network (GAT) (Velickovic et al., 2017). One of the vital work is the Graph Isomorphism Network (GIN) (Xu et al., 2018), which is well known for the same powerful expression as the Weisfeiler-Lehman (WL) graph isomorphism test (Leman & Weisfeiler, 1968), and it is defined as : $x_i' = h_{\mathbf{\Theta}}\left((1+\epsilon)\cdot x_i + \sum_{j\in\mathcal{N}(i)} x_j\right)$, where $h_{\mathbf{\Theta}}$ is an MLP. We believe that this representation ability is helpful for the graph similarity learning, because those isomorphic graphs have the maximum similarity scores. Therefore, we leverage the GIN layer as the backbone to construct our N^{2}AGim.

2.2 DEEP GRAPH SIMILARITY LEARNING

The graph similarity problem is defined as: given two graphs G_i and G_j with their similarity metric, the graph similarity models learn a function that maps the two graphs to their similarity metric. Bai et al. (2019) first normalized the GED as $nGED(G_1, G_2) = \frac{GED(G_1,G_2)}{(N_1+N_2)/2}$ and transformed the $nGED$ to the similarity score as $s(G_1, G_2) = exp(-nGED(G_1, G_2))$, which is in the range of (0,1]. Graph Matching Network (GMN) (Li et al., 2019) was the first proposed deep graph similarity model which computes the similarity between two given graphs by a cross-graph attention mechanism. Bai et al. (2019) turned the graph similarity task into a regression task. They not only proposed the widely used graph similarity datasets, but they also leveraged the GCN layers and self-attention-based fusion to design SimGNN. Further, in their later work (Bai et al., 2020), the proposed GraphSim directly learns the similarity based on the node-level interaction of two given graphs. By leveraging a trained SimGNN to guide the search space of the A* algorithm, GENN-A* (Wang et al., 2021) achieves the best-in-class performance, but has a too long inference time on the test data, i.e., taking 290.1 hours to solve the GED computation on AIDS700nef dataset (Qin et al., 2021). Considering that GENN-A* is too costly for use in real-time downstream tasks, we do not compare our proposed methods with it in our evaluations. In order to achieve a faster speed,



(a) (b)

Figure 3: Overlapping probability of graph embedding with N and $N+\delta$ number of nodes.

Qin et al. (2021) proposed a special designed Knowledge Distillation (KD) paradigm to improve the individual graph embeddings generated by the student model (EGSCS).

However, we found that none of these existing graph similarity models are designed to address the N^{2}AI. In order to address the N^{2}AI, our N^{2}AGim leverages the GIN layers and the proposed DiffAtt to enhance the differences between the the embeddings of two graphs and therefore achieves the state-of-the-art performance on the real world datasets. Compared to the (Qin et al., 2021), our GSL^2 directly generate an individual graph representation for each graph, and the only learnable parameter of our GSL^2 is a simple MLPs. Besides, the evaluation on real world datasets also shows that our GSL^2 has a higher accuracy and faster inference time than EGSCS.

3 Node Number Awareness Issue (N^{2}AI)

In this section, we provide a formal theoretical analysis of N^{2}AI and reveal the reasons for its existence. GNNs usually obtain the embeddings of a graph by multi-layer GNN aggregation layers and global pooling methods. The GIN is known as having the same upper limit of ability as WL-Test, that is, to distinguish whether two graphs are isomorphic, which indicates that GIN is effective in distinguish the graphs with similar node numbers and address the N^{2}AI. Hence, we focus on the impact of the different widly used global pooling methods with no parameters on the N^{2}AI, including the one order statistical methods, i.e., Global Sum Pooling (GSP), Global Max Pooling (GMP), Global Average Pooling (GAP) and the second order statistical methods, i.e., Second Order Pooling (SOP), and our conclusions can be easily extended to the case with parameters.

Lets assume that the node feature matrix output by the graph neural network layers is $X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N]^T = [\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_C]$, where $\boldsymbol{x}_i \in \mathbb{R}^C$ is the ith node feature and $\boldsymbol{v}_i$ is the feature on the ith channel. We model all variables in X as i.i.d random variables that follow a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, where $\mu > 0$. The one order statistical methods $\mathcal{F}_i$ is used to convert $\boldsymbol{v}_i$ into a single value $g = \mathcal{F}(\boldsymbol{v}_i)$, which outputs a fix-sized vector, and the second order statistical methods convert $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ as a single value $g = \mathcal{F}(\boldsymbol{v}_i, \boldsymbol{v}_j)$, which outputs a fix-sized matrix. Here, we learn the N^{2}AI by studying whether a pooling method F_i can distinguish well between X with the number of nodes N and $N+\delta$, i.e., the differentiation between the two distributions $p(g|N, F_i)$ and $p(g|N+\delta, F_i)$, where δ denotes the difference between the number of nodes. We first assume that X obeys $\mathcal{N}(1, 4)$ and show the output distribution of different pooling functions for different numbers of nodes in Fig 2. Intuitively, the all four global pooling methods have a lot of overlap in the output distribution when the number of nodes is similar, and less overlap in the distribution when the number of nodes is very different.

We further define the probability of this overlap by the following equation:

$$O(\mathcal{F}_i, N, \delta) = \int \frac{min\left\{p(g|N, F_i), p(g|N+\delta, F_i)\right\}}{max\left\{p(g|N, F_i), p(g|N+\delta, F_i)\right\}} dg, \tag{1}$$

where $O(\mathcal{F}_i, N, \delta)$ denotes the proportion of the overlapping area that occupies the total area of $\mathcal{F}_i$ with N and $N+\delta$ nodes. Obviously, the outputs of the GSP and GAP obey the Gaussian distribution $\mathcal{N}(N\mu, N\sigma^2)$ and $\mathcal{N}(\mu, \frac{\sigma^2}{N})$, respectively, but it is difficult to obtain the distribution that the GMP and the SOP satisfy. Therefore, we perform a large number of randomized experiments and leverage the Kernel Density Estimation (KDE) to obtain an approximate distribution for the GMP and SOP. The overlapping probabilities of the four global pooling methods are shown in Fig 3.

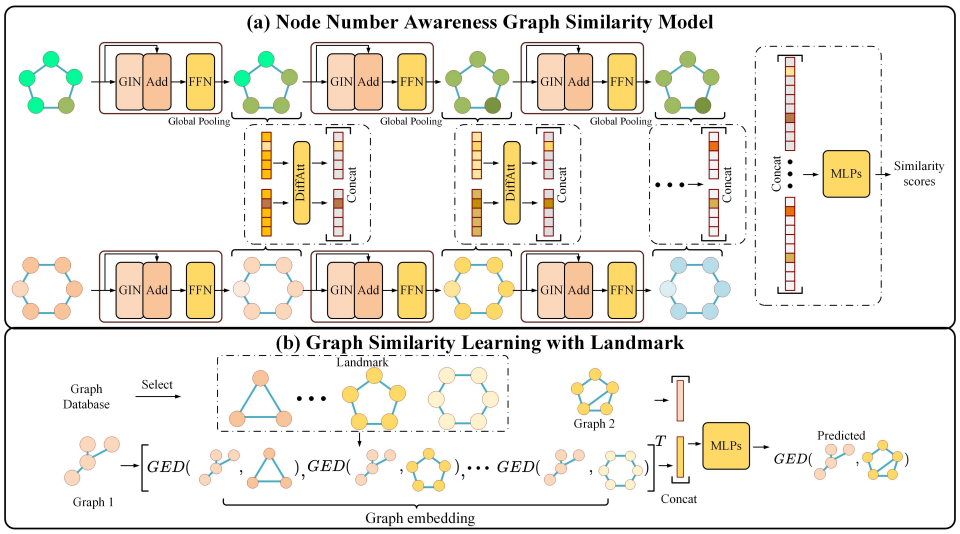


Figure 4: A block diagram of N^{2}AGim and GSL^2.

From Fig 3(a), it is clear that most of the global pooling methods overlap in greater than 80% of the area of the distribution with N and $N+1$ nodes, which means that existing graph similarity networks are very difficult to distinguish the graphs with similar number of nodes in the output distribution, thus leading to the N^{2}AI. The Fig 3(b) shows that as the difference in the number of nodes becomes larger, the probability of overlapping between the embedding distributions becomes smaller. This similar embedding masks the differences between the two graphs, resulting in reduced performance of the graph similarity models in predicting the similarity of two graphs with similar number of nodes. One way to address N^{2}AI is to make the graph similarity model more focused on where the two embeddings differ. Inspired by this, we propose the **DiffAtt** to enhance the difference between two embeddings generated by the above four global pooling methods. Surprisingly, the evaluation on real-world benchmarks show that our DiffAtt brings a huge performance improvement to the graph similarity models.

4 Proposed Methods

4.1 Node Number Awareness Graph Similarity Model (N^{2}AGim)

Our proposed N^{2}AGim is consists of three stages: Multi-Scale GIN layers, Different attention based feature fusion and the MLP regressor. Fig 4(a) shows a block diagram of our N^{2}AGim. Here are the details of our N^{2}AGim:

Multi-Scale GIN layers. Given a graph data $G = (A, X)$, the GIN layers, which can effectively address N^{2}AI because it has the upper limit of the ability to distinguish whether two graphs are isomorphic or not, are leveraged as our backbone to update the node embeddings. All the MLPs in GIN have one linear layer with the Layer Normalization (Ba et al., 2016) and ReLU activation function. Besides, we apply the residual connections (He et al., 2016) and an additional FeedForward Neural Network (FFN) to enhance the node embeddings. One GIN layer can only aggregate the information of one hop neighbors for each node, so we stack K GIN layers to aggregate multi-scale information of neighbors. After each GIN layer, a one order statistical pooling method is applied to generate the graph embeddings.

Different attention based feature fusion. Here we propose our **Different Attention (DiffAtt)** to enhance the different between the embeddings to address the N^{2}AI and obtain the joint embedding by fuse the features of the two graph-level embeddings at each layer. Given the graph embeddings $\boldsymbol{h}_i^k$ and $\boldsymbol{h}_j^k$ at the kth layer, the DiffAtt is defined as :

$$\begin{aligned} Att^k &= Softmax(MLPs^k(|\boldsymbol{h}_i^k - \boldsymbol{h}_j^k|)) \\ u_{G_i}^k &= flatten(Att^k \odot \boldsymbol{h}_i^k), u_{G_j}^k = flatten(Att^k \odot \boldsymbol{h}_j^k) \end{aligned} \tag{2}$$

where $\boldsymbol{u}_{G_i}^k \in \mathbb{R}^C$ is the enhancement embeddings of G_i and the $flatten(\cdot)$ denotes the flatten operation. Obviously, DiffAtt can product greater weight to the large differences in the two embeddings and dynamically capture the differences that really matter with learnable parameters, which can effectively address the N^{2}AI. Then we concatenate the two enhancement embed-

dings as their joint embeddings as $\boldsymbol{u}^k_{G_i,G_j} = concat([\boldsymbol{u}^k_{G_i}, \boldsymbol{u}^k_{G_j}])$. At the end, we concatenate all the joint embeddings $\boldsymbol{u}^k_{G_i,G_j}$ at different layers to obtain a multi-scale joint embedding as $\boldsymbol{u}_{G_i,G_j} = concat([\boldsymbol{u}^0_{G_i,G_j}, \cdots, \boldsymbol{u}^K_{G_i,G_j}])$.

MLP regressor. A two-layer MLPs is then applied to map $\boldsymbol{u}_{G_i,G_j}$ to the similarity scores. In the graph similarity task, the normalization GED is defined as $nGED(G_1, G_2) = \frac{GED(G_i,G_j)}{(|N_i|+|N_j|)/2}$, and the ground truth similarity score is defined as $exp(-nGED(G_i, G_j))$, which is in the range of (0,1]. We adopt the **Mean Square Error (MSE)** as the loss function to train our N^{2}AGim.

4.2 Graph Similarity Learning with Landmarks (GSL^2)

The graph similarity task inherently requires a deep fusion of the features of two graphs at the early stage and then learns from the joint embeddings to predict the similarity score, which is shown at Fig 2(f). This makes it difficult to extract the individual embedding of each graph, which leads to the higher computational costs in practice (Qin et al., 2021). Existing work (Qin et al., 2021) used a KD-paradigm to improve the individual embeddings generated by the student IEM, however here we provide a novel IEM framework, called **Graph Similarity Learning with Landmarks (GSL^2)**, which directly generates the individual embeddings of each graph without additional learning and can be used on any graph similarity model. The motivation of our GSL^2 is to use the relationship between GED values to predict new GED values, which avoids the need to perform complex computations directly on the graph data to learn a high-quality embedding.

Theorem 1 *Let landmarks $\mathcal{S}$ denotes an infinite set containing every graphs, and $\boldsymbol{u}_G$ is the embedding of graph G and defined as: $\boldsymbol{u}_G = [GED(G, \hat{G}_1), \cdots, GED(G, \hat{G}_{Infinity})]^T$, where $\hat{G}_i \in \mathcal{S}$. The GED value between G_1 and G_2 satisfies:*

$$GED(G_1, G_2) = \min_i \{\boldsymbol{u}_{G_{1_i}} + \boldsymbol{u}_{G_{2_i}}\}. \tag{3}$$

The proof of Theorem 1 is provided in the Appendix. Theorem 1 allows us to predict GED values not by learning on graph data, but by using the GED values between two graphs and landmarks. However, there are two problems here, the first being that it is not possible to obtain an infinite set S in reality, and the other being that the large number of computation of the exact GED values requires too much computational cost. The first problem can be solved by randomly selecting M graphs from the training graph set to form $\mathcal{S}$. For the second problem, a GNN-based graph similarity model can be used to very fast predict the GED values between graphs and landmarks. However, the $\tilde{\boldsymbol{u}}_G$ computed by the graph similarity models is with error compared to the exact $\boldsymbol{u}_G$, which is equivalent to be added a noise, i.e., $\tilde{\boldsymbol{u}}_G = \boldsymbol{u}_G + \boldsymbol{d}_G$, where $\boldsymbol{d}_G$ denotes random noise vector. Equation 3 is equal to: $GED(G_1, G_2) = \min_i\{\tilde{\boldsymbol{u}}_{G_{1_i}} + \tilde{\boldsymbol{u}}_{G_{2_i}} - \boldsymbol{d}_{G_{1_i}} - \boldsymbol{d}_{G_{2_i}}\}$. However, the noise vector leads to a relationship between the two generated embeddings $\tilde{\boldsymbol{u}}_G$ and their GED target that is difficult to confirm. Therefore, to minimize noise, we use our high-precision N^{2}AGim to generate $\tilde{\boldsymbol{u}}_G$ for each graph and leverage the MLPs to learn the relationship between the two embeddings $\tilde{\boldsymbol{u}}_G$ with noise and their GED targets.

A illustration of our GSL^2 is shown in Fig 4(b), and the details are as follows: **First**, a subset of the graphs, $\mathcal{S} = \{\hat{G}_1, \cdots, \hat{G}_M\}$, named **landmarks**, are randomly selected from the the training graph set. **Second**, any graph similarity model can be leveraged to fast obtain the individual embeddings for each graph by computing their GEDs to the landmarks. However, from the above analysis, we can see that reducing the random noise $\boldsymbol{d}_G$ can effectively improve the prediction accuracy. Therefore, we leverage our N^{2}AGim, which achieves the state-of-the-art performance, to calculate GED values for all graphs with landmarks to obtain their individual embeddings. However, we found that directly converting exponential similarity values to GED values caused significant errors, so we used the ATS^2 similarity metric to retrain N^{2}AGim, see the Appendix for details. **Third**, we concatenate two individual graph embeddings together as their joint embedding and learn a MLP to map the joint embedding to their GED target.

Obviously, our GSL^2 leverages the relationship between GEDs to predict the new GED values and therefore does not require any additional learning of the graph data. Compared to the existing complex KD process (Qin et al., 2021), our GSL^2 can be applied to all graph similarity calculators and even the MLPs can be replaced by other regression algorithms that are suitable in practice.

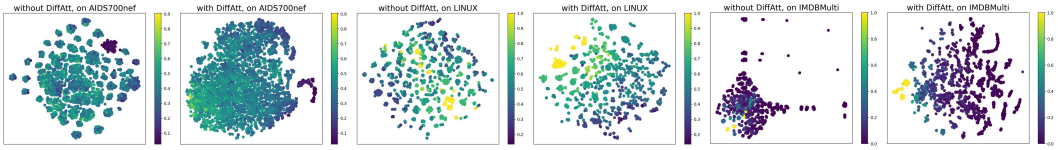


Figure 5: t-SNE visualisation of the joint embeddings generated by models using DiffAtt and those not using DiffAtt. The colours represent the similarity ground truth of the joint embeddings.

4.3 COMPARISON OF OUR N^2AGIM AND GSL^2

Accuracy. N^2AGim can effectively address N2AI by fusing the features of two graphs by DiffAtt at multiple scales, and thus can achieve better performance. However, GSL^2 uses N^2AGim to quickly generate an individual embedding with noise for each graph and learns from the noisy embeddings, so the performance will be lower than that of N^2AGim.

Inference speed. Suppose given q query graphs, the aim is to compute the similarity between all query graphs and the p graphs existing in the database. Assume the time to compute the similarity of a pair of graphs is T_N for N^2AGim and the time to compute the similarity of a pair of embeddings is T_{MLPs} for the MLPs in GSL^2. Since N^2AGim, as a EFM, requires fusion of the graph pairs to obtain the joint embeddings at each layer, it has a computational time of $p \times q \times T_N$. However, GSL^2, as a IEM, first generates the individual embedding of each graph using N^2AGim and then predicts the similarity on two embeddings using MLPs, requiring a computation time of $(p+q) \times M \times T_N + p \times q \times T_{MLPs}$. Since GSL^2 reduces the time complexity of computing joint embeddings and the inference speed of MLPs is generally lower than that of N^2AGim, query tasks can be address more quickly by GSL^2. Especially in industrial scenarios, graph data is usually preprocessed offline as the embedding, which can accelerate the inference speed of GSL^2. If all the graph embeddings are stored offline, the inference time of GSL^2 is just $p \times q \times T_{MLPs}$, which is suitable for use in real-time graph similarity calculation tasks.

5 EXPERIMENTS

In this section, we evaluate our proposals using the AIDS700nef, LINUX and IMDBMulti datasets provided by Bai et al. (2019) for the graph similarity learning and compare our method with other state-of-the-art methods. The statistics and details of the datasets and the data processing are provided in the Appendix A.2. Note that all the experiments are performed on a Linux server with Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz and 8 NVIDIA GeForce RTX 2080Ti. The evaluation metrics we adopted are Mean Square Error (MSE) (in the format of 10^{-3}), Spearman's Rank Correlation Coefficient (ρ) and Precision at 10 (p@10). All metrics with their meanings are listed in the Appendix. The **N^2AGim** is evaluated using the PyTorch Geometric (Fey & Lenssen, 2019). We use Adam optimizer, and a learning rate of 0.001, and the batch size is set to 2000, and the hidden channel is set to 64. We run 200 epochs on the three datasets, and after running 150 epochs, we perform validation at the end of every epoch. Ultimately, the parameter that results in the least validation loss is chosen to perform the evaluations on the test data. We implement the **GSL^2** using the PyTorch (Paszke et al., 2019), and the detailed details can be found in our code.

5.1 ABLATION STUDY

We perform an ablation study to show the influence of the DiffAtt in N^2AGim and GSL^2 par with the different GED computation algorithms. For N^2AGim, we compare the performance of using and without using DiffAtt across the four global pooling functions mentioned above. Besides, we compare our DiffAtt with the other popular attention based global pooling methods, i.e., Neural Tensor Network (NTN) (Bai et al., 2019), Embedding Fusion Network (EFN) (Qin et al., 2021), Global Soft Attention (GSA) (Li et al., 2015), Set2Set (Vinyals et al., 2015), Context Based Attention (CBA) Bai et al. (2019), and Cross Context Based Attention (C2BA), under the same architecture. It is worth noting that most of the existing graph similarity models leverage the CBA to generate graph embeddings, e.g., Li et al. (2019); Bai et al. (2019); Qin et al. (2021); Zhang et al. (2021), which is defined as $\mathcal{F}_{att}(X) = \sum_{n=1}^{N} sigmoid(\boldsymbol{x}_n^T \boldsymbol{c})\boldsymbol{x}_n$. The $\boldsymbol{c}$ denotes the context information of the graphs. C2BA is different from CBA only in that global context information $\boldsymbol{c}$ is from another graph in the graph pair. For our GSL^2, we experiment with different graph similarity models, and we also

Table 1: Results of the ablation study on using or without using our DiffAtt. Bold means the best and Δ denotes the improvement after using DiffAtt. The Average denotes the average value over the three datasets. The $\uparrow$ denotes that the larger this indicator is, the better it is, while the $\downarrow$ is the opposite.

$\mathcal{F}_i$	DiffAtt	AIDS700nef			LINUX			IMDBMulti			Average		
		MSE $\downarrow$	$\rho\uparrow$	P@10 $\uparrow$	MSE $\downarrow$	$\rho\uparrow$	P@10 $\uparrow$	MSE $\downarrow$	$\rho\uparrow$	P@10 $\uparrow$	MSE $\downarrow$	$\rho\uparrow$	P@10 $\uparrow$
GAP	✘	**3.004**	**0.819**	0.486	0.457	0.987	0.970	0.753	0.866	0.878	1.405	0.891	0.778
GAP	✔	3.042	0.810	**0.511**	**0.256**	**0.990**	**0.979**	**0.404**	**0.907**	**0.890**	**1.234**	**0.902**	**0.793**
GMP	✘	3.051	0.817	0.501	**0.396**	**0.988**	0.970	0.386	0.871	0.875	1.278	0.892	0.782
GMP	✔	**2.839**	**0.827**	**0.538**	0.440	0.987	**0.985**	**0.305**	**0.920**	**0.896**	**1.195**	**0.911**	**0.806**
SOP	✘	3.039	0.811	0.496	0.261	0.991	0.973	0.776	0.892	0.863	1.359	0.898	0.777
SOP	✔	**1.144**	**0.918**	**0.663**	**0.066**	**0.993**	**0.998**	**0.309**	**0.915**	**0.903**	**0.506**	**0.942**	**0.855**
GSP	✘	1.396	0.903	0.624	0.116	0.992	0.983	0.392	0.859	0.884	0.635	0.918	0.830
GSP	✔	**1.170**	**0.916**	**0.672**	**0.066**	**0.994**	**0.995**	**0.308**	**0.918**	**0.893**	**0.515**	**0.943**	**0.853**

Table 2: Results of the ablation study of comparing our DiffAtt with other attention methods. Bold means the best, and † means the next best.

	AIDS700nef			LINUX			IMDBMulti			Average		
	MSE $\downarrow$	$\rho\uparrow$	P@10 $\uparrow$	MSE $\downarrow$	$\rho\uparrow$	P@10 $\uparrow$	MSE $\downarrow$	$\rho\uparrow$	P@10 $\uparrow$	MSE $\downarrow$	$\rho\uparrow$	P@10 $\uparrow$
NTN	2.456	0.845	0.554	0.139	0.993†	0.987	0.463	0.891	0.874	1.019	0.91	0.805
GBA	3.224	0.814	0.499	0.754	0.982	0.961	0.368	0.869	0.881	1.449	0.888	0.78
Set2Set	3.308	0.815	0.511	0.543	0.985	0.974	0.355	0.876	0.878	1.402	0.892	0.788
CBA	1.487	0.902	0.611	0.165	0.99	0.987	0.406	0.855	0.884	0.686	0.916	0.827
C2BA	1.269	0.911	0.662	0.094	0.993	**0.995**	0.439	0.887	0.875	0.601	0.93	0.844†
EFN	1.249†	0.912†	0.644†	0.078†	0.993†	0.990†	0.315†	0.902†	0.891†	0.547†	0.936†	0.842
DiffAtt	**1.170**	**0.916**	**0.672**	**0.066**	**0.994**	**0.995**	**0.308**	**0.918**	**0.893**	**0.515**	**0.943**	**0.853**

provide in the Appendix more ablation experiments on GSL, including hyperparameter selection, using different random seeds and so on.

The results in Table 1 demonstrate that our DiffAtt effectively improves 44 metrics out of 48 metrics of the four global pooling methods, especially giving a huge boost to 12 metrics on the IMDBMulti dataset, such as 0.349 (0.404 vs 0.753), 0.081 (0.305 vs 0.386), 0.084 (0.308 vs 0.392) and 0.467 (0.309 vs 0.776) on the four MSE metrics. This shows that DiffAtt has powerful generalization to effectively solve the N^2AI of all pooling methods. In terms of the average results, SOP and GSP showed better results than GMP and GAP after the use of DiffAtt, which is a result of the smaller percentage of confusion area and greater distribution differences. Because of the higher performance and less computational effort of GSP, we finally chose it as the global pooling function in N^2AGim. Compared to other attention mechanisms, in Table 2, DiffAtt achieves the best results on all metrics under the same experimental setup and architecture, especially better than the EFN on average on three metrics 5.8%(0.515 vs 0.547), 0.7% (0.943 vs 0.936) and 1.3%(0.853 vs 0.842), respectively. As can be seen from the results in Table 3, the accuracy of GSL^2 increases as the generated GEDs get accurate to the true GED values, which validates our analysis in 4.2 and shows that the performance of GSL^2 can be improved by using our N^2AGim. To better understand the enhancement that DiffAtt brings to the model, we visualised the joint embeddings generated with and without DiffAtt in each of the three datasets using T-SNE, as shown in Fig 5. As can be seen from the Fig 5, the joint embeddings generated by the model with DiffAtt are more separable than the model without DiffAtt.

5.2 GRAPH SIMILARITY LEARNING

We compare our N^2AGim and GSL^2 with a number of state-of-the-art methods in graph similarity learning tasks: GMN (Li et al., 2019), SimGNN (Bai et al., 2019), H2MN (Zhang et al., 2021),

Table 3: Results of the ablation study of comparing our graph similarity model in GSL^2. All graph similarity models were trained using ATS^2 and we transform the results to the exponential similarity scores and report it. The brackets represent the GED algorithm used in GSL^2 and the GT represents the ground truth GED values. Bold means the best, and † means the next best.

	AIDS700nef			LINUX			IMDBMulti			Average		
	MSE $\downarrow$	$\rho\uparrow$	P@10 $\uparrow$	MSE $\downarrow$	$\rho\uparrow$	P@10 $\uparrow$	MSE $\downarrow$	$\rho\uparrow$	P@10 $\uparrow$	MSE $\downarrow$	$\rho\uparrow$	P@10 $\uparrow$
SimGNN	3.151	0.827	0.397	0.752	0.983	0.921	3.722	0.934	0.826	2.542	0.915	0.715
GraphSim	1.824	0.889	0.562	0.283	0.991	0.979	-	-	-	1.053	0.94	0.771
N^2AGim	1.184	0.917	0.675	0.071	0.994	0.989	0.341	0.973	0.9	0.532	0.961	0.855
GSL^2(SimGNN)	2.187	0.873	0.518	0.402	0.987	0.975†	0.668	0.949	0.852†	1.086	0.936	0.782
GSL^2(GraphSim)	1.824	0.883	0.505	0.210	0.990	0.971	-	-	-	1.017	0.936	0.738
GSL^2(N^2AGim)	1.470†	0.905†	0.604†	0.074†	0.994†	**0.995**	**0.510**	0.971†	**0.869**	0.685†	0.957†	0.822†
GSL^2(GT)	**1.258**	**0.915**	**0.633**	**0.068**	**0.995**	**0.995**	0.512†	**0.985**	0.850	**0.613**	**0.965**	**0.826**

Table 4: Results of the graph similarity learning task. Bold means the best.

	AIDS700nef			LINUX			IMDBMulti			Average		
	MSE ↓	ρ ↑	P@10 ↑	MSE ↓	ρ ↑	P@10 ↑	MSE ↓	ρ ↑	P@10 ↑	MSE ↓	ρ ↑	P@10 ↑
GMN	3.772	0.751	0.401	2.054	0.933	0.833	8.844	0.725	0.604	4.89	0.803	0.613
SimGNN	2.378	0.843	0.421	3.018	0.939	0.942	2.528	0.878	0.759	2.641	0.887	0.707
GraphSim	1.574	0.874	0.534	0.116	0.981	0.992	1.486	0.926	0.828	1.059	0.927	0.785
H2MN	1.826	0.881	0.521	0.210	0.990	0.975	1.178	0.913	0.889	1.071	0.928	0.795
EGSCT	1.601	0.901	0.658	0.163	0.988	0.994	0.553	0.938	0.872	0.772	0.942	0.841
EGSCS	1.546	0.898	0.649	0.293	0.984	0.978	0.581	0.935	0.857	0.807	0.939	0.828
N^2AGim	**1.170**	**0.916**	**0.672**	**0.066**	**0.994**	**0.995**	**0.308**	0.918	**0.893**	**0.515**	0.943	**0.853**
GSL^2	1.470	0.905	0.604	0.074	**0.994**	**0.995**	0.510	**0.971**	0.869	0.685	**0.957**	0.822

Table 5: The results of inference time for each model. The suffix of '-R' means that the input is the raw query graph, while the suffix of '-F' means that the embeddings of the query graph are stored offline. All times reported below are in seconds.

	SimGNN	N^2AGim	GraphSim	EGSCS-R	EGSCS-F	GSL^2-R	GSL^2-F
AIDS700nef	5.106	9.245	4.383	4.383	0.975	3.874	**0.718**
LINUX	8.582	13.163	9.120	9.12	1.414	5.007	**1.159**
IMDBMulti	122.939	87.032	114.676	87.032	2.256	15.792	**1.824**

GraphSim (Bai et al., 2020), and EGSC (Qin et al., 2021). We note discrepancies in the MSE reported by various papers. For examples, the MSE reported in Bai et al. (2019), Bai et al. (2020) and Zhang et al. (2021) is $\frac{1}{2|D|}\sum_{i=1}^{D}(s-\hat{s})^2$, but Fey & Lenssen (2019) reported MSE as $\frac{1}{|D|}\sum_{i=1}^{D}(s-\hat{s})^2$. To provide a consistent comparison, we use the MSE metric of the latter formula, and the results are shown in Table 4. Our N^2AGim achieves the best performance on all the three metrics On AIDS700nef, especially better by about 24.3%(1.170 vs 1.546 on MSE), 2.0%(0.916 vs 0.898 on ρ) and 3.5%(0.672 vs 0.649 on p@10) compared to EGSCS. On LINUX, our N^2AGim achieves the best performance in all three metrics, especially in MSE which is 43.1%(0.066 vs 0.116) better than the second best model, GraphSim. On IMDBMulti, N^2AGim achieves the best MSE and p@10 performance, but less well on ρ (0.918 vs 0.938) than EGSCT. Surprisingly, although GSL^2 does not learn embeddings directly in the graph data, it achieves state-of-the-art performance on three of the nine metrics on three datasets. Compared to the EGSCS, our GSL^2 achieved better performance in eight of nine metrics on three datasets, demonstrating the powerful expressive ability of the generated embeddings in GSL^2. Compared to GSL^2, which learns on embeddings with noise, N^2AGim achieves a better performance, espically better about 20.4%(1.170 vs 1.470), 1.2%(0.916 vs 0.905) and 11.3%(0.672 vs 0.604) on AIDS700nef.

5.3 Inference Time

In this subsection, we provide a comparison of inference times for GSL^2 and the rest of the graph similarity models on real test data. Our evaluation reflects real-world graph queries: we treat the training graph set as the graphs exist in the database and can be preprocessed, and the test graph set as the query graph. We calculate the similarity of a query graph to all graphs in the database at once, and all times are averaged over five tests. The results are shown in the Table 5. By obtaining individual embeddings of the graphs offline, GSL^2-F can be 12.9, 11.3 and 47.7 times faster than N^2AGim on the three datasets, respectively. Compared to EGSCS-F, GSL-F is 26.4, 18.0 and 19.1 times faster, respectively. This shows the potential of GSL^2 for real-time graph similarity downstream tasks.

6 Conclusion

This paper addresses two issue in graph similarity tasks, one is the **Node Number Awareness Issue (N2AI)** and the other is the issue of improving the speed of graph similarity model inference in real-time tasks. By analysing the performance of popular graph similarity models, we find that graph similarity models have difficulty in distinguishing the embeddings of two graphs with similar numbers of nodes, because the global pooling function maps graphs with similar numbers of nodes to similar embedding distributions, masking the difference information between the two graphs. Therefore, we propose **DiffAtt** to enhance the difference between two similar embeddings, thus proposing our N^2AGim that achieves the state-of-the-art performance. To handle real-time downstream tasks faster, we propose a lightweight model, called GSL^2. Instead of learning embeddings in graph data, GSL^2 generates individual embeddings directly by an already trained graph similarity model. Our analysis and experiments both demonstrate that such individual embeddings have a powerful expressive ability and can efficiently handle real-time downstream tasks.

REFERENCES

Moez Ali. *PyCaret: An open source, low-code machine learning library in Python*, April 2020. URL `https://www.pycaret.org`. PyCaret version 1.0.0.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Jiyang Bai and Peixiang Zhao. Tagsim: type-aware graph similarity learning and computation. *Proceedings of the VLDB Endowment*, 15(2):335–347, 2021.

Yunsheng Bai, Hao Ding, Song Bian, 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*, pp. 384–392, 2019.

Yunsheng Bai, Hao Ding, Ken Gu, Yizhou Sun, and Wei Wang. Learning-based efficient graph similarity computation via multi-scale convolutional set matching. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3219–3226, 2020.

Maria-Florina Balcan, Avrim Blum, and Nathan Srebro. A theory of learning with similarity functions. *Machine Learning*, 72(1):89–112, 2008.

Sébastien Bougleux, Luc Brun, Vincenzo Carletti, Pasquale Foggia, Benoit Gaüzere, and Mario Vento. A quadratic assignment formulation of the graph edit distance. *arXiv preprint arXiv:1512.07494*, 2015.

Stefan Fankhauser, 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*, pp. 102–111. Springer, 2011.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and applications*, 13(1):113–129, 2010.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

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

Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. A survey on graph kernels. *Applied Network Science*, 5(1), 2020. ISSN 23648228. doi: 10.1007/s41109-019-0195-3.

AA Leman and Boris Weisfeiler. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 2(9):12–16, 1968.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *International conference on machine learning*, pp. 3835–3845. PMLR, 2019.

Xiang Ling, Lingfei Wu, Saizhuo Wang, Tengfei Ma, Fangli Xu, Alex X Liu, Chunming Wu, and Shouling Ji. Multilevel graph matching networks for deep graph similarity learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

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

Seongmin Ok and South Korea. A Graph Similarity for Deep Learning. (NeurIPS), 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Can Qin, Handong Zhao, Lichen Wang, Huan Wang, Yulun Zhang, and Yun Fu. Slow learning and fast inference: Efficient graph similarity computation via knowledge distillation. *Advances in Neural Information Processing Systems*, 34, 2021.

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.

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

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *stat*, 1050:20, 2017.

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

Runzhong Wang, Tianqi Zhang, Tianshu Yu, Junchi Yan, and Xiaokang Yang. Combinatorial learning of graph edit distance via dynamic embedding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5241–5250, 2021.

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*, pp. 210–221. IEEE, 2012.

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

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1365–1374, 2015.

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.

Zhen Zhang, Jiajun Bu, Martin Ester, Zhao Li, Chengwei Yao, Zhi Yu, and Can Wang. H2mn: Graph similarity learning with hierarchical hypergraph matching networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 2274–2284, 2021.

A APPENDIX

A.1 PROOF OF THEOREM 1.

Here, we provide a proof of Theorem 1. The GED satisfies the triangle inequality, i.e., for any G_1 ,G_2 and G_3, there must exist:

$$GED(G_1, G_2) \leq GED(G_1, G_3) + GED(G_2, G_3). \tag{4}$$

In particular, when G_3 is isomorphic to a graph on the least-cost edit path between G1 and G2, there must exist

$$GED(G_1, G_2) = GED(G_1, G_3) + GED(G_2, G_3). \tag{5}$$

Thus, assume there exists an infinite set S containing every graphs, any graph can be encoded as an infinitely long vector as $\boldsymbol{u}_G = [GED(G, \hat{G}_1), GED(G, \hat{G}_2), \cdots, GED(G, \hat{G}_{Infinity})]^T$, where $\hat{G}_i \in \mathcal{S}$, so that for two any graphs G_1 and G_2, it exists $GED(G_1, G_2) = \min_i(\boldsymbol{u}_{G_{1_i}} + \boldsymbol{u}_{G_{2_i}})$.

A.2 DATASETS AND PRE-PROCESSING

We perform an evaluation of our methods on AIDS700nef, LINUX and IMDBMulti datasets provided by Bai et al. (2019). The brief introduction of benchmark datasets are as follows :

1. **AIDS700nef** dataset contains 700 graphs from AIDS dataset which represent antivirus screen chemical compound, and all of them have 10 or less than 10 nodes.
2. **LINUX** dataset contains 1000 graphs selected from Wang et al. (2012), which represent Program Dependence Graph (PDG) generated by Linux kernel.
3. **IMDBMulti** datasetYanardag & Vishwanathan (2015) contains ego-networks of actors/actresses, where nodes represent an actor/actress and edges indicate that these two actors/actress participated in the same movie.

For AIDS700nef and LINUX datasets, Bai et al. (2019) computes the GED of every graph pair using an algorithm named A*, and for IMDB datasets, the minimum of GED computed by three algorithms: Beam (Neuhaus et al., 2006), Hungarian (Riesen & Bunke, 2009) and VJ (Fankhauser et al., 2011), is considered as the ground truth. In order to enhance the node features of the graph, we concatenate the one-hot encoding of the node degree into its features on these three datasets. Note that, the GED metric is first normalized as $nGED = \frac{GED_{G_i,G_j}}{0.5\cdot(|G_i|+|G_j|)}$, where $|G_i|$ represents the number of nodes in G_i. and then adopted a function $\lambda(x) = e^{-x}$ to transform to range (0,1]. We randomly split datasets into $60\%, 20\%, 20\%$ as training graph set Tr, validation graph set V, and testing graph set Te, respectively. We take the Cartesian product of Tr labeled with their similarity scores as the the training set. The validation set (testing set) is defined as the Cartesian product of Tr and V (Te) labeled with the ground truth. The training set is defined as $\{(G_i, G_j, GED_{G_i,G_j})|G_i \in Tr, G_j \in Tr\}$, and the validation dataset and the testing dataset is $\{(G_i, G_j, GED_{G_i,G_j})|G_i \in Tr, G_j \in V\}$, $\{(G_i, G_j, GED_{G_i,G_j})|G_i \in Tr, G_j \in Te\}$, respectively. We compute the Cartesian product of the training set and the training set labeled with their nGED metric to build the training data as $\{(G_i, G_j, GED_{G_i,G_j})|G_i \in Tr, G_j \in Tr\}$. Similary, the validation dataset is defined as $\{(G_i, G_j, GED_{G_i,G_j})|G_i \in Tr, G_j \in V\}$, and the testing dataset is $\{(G_i, G_j, GED_{G_i,G_j})|G_i \in Tr, G_j \in Te\}$.

However, we found in GSL^2 that directly converting the exponential similarity scores predicted by the graph similarity models to GED values can cause significant errors. Therefore, we used a new similarity score formula to train the graph similarity models, called **Adaptive Transform Similarity Scores (ATS^2)**, and transform the results back to exponential similarity score for comparison with other models at test time. The ATS^2 is defined as:

$$ATS^2(G_1, G_2) = 1 - \lg(nGED(G_1, G_2) + 1)/\lg(\max_{i,j}\{nGED(G_i, G_j)\} + 1). \tag{6}$$

Table 6: Statistics of all the datasets used in our experiments.

Datasets	Graphs	Avg nodes	Avg edges	Pairs of testing graphs	Node attr
AIDS700nef	700	8.9	17.6	58,800	✔
LINUX	1000	7.58	13.87	120,000	✗
IMDBMulti	1500	13	65.94	270,000	✗

A.3 EVALUATION METRICS

The evaluation metrics that we adopted are the Mean Square Error (MSE), the Spearman's Rank Correlation Coefficient (ρ) (Spearman, 1961), and the Precision at 10 (p@10) (Bai et al., 2019). The MSE metric can accurately calculate the distance between the predition results from the model and the ground truth, and the ρ evaluates the matching between the global ranking result of the prediction results and the ground truth, while the p@10 is the intersection of the top 10 results of the prediction and the ground truth.

A.4 N^2AI

Here, we provide a more detailed description of the N^2AI for the now popular graph similarity model. We grouped the graphs in the test data and counted the MSEs, we normalized the MSEs with Min-Max Normalization, and visualized the results.

From the results, we can see that the large MSE are all concentrated in locations with similar number of nodes, which reflects the prevalence of N^2AI.

B ADDITIONAL EXPERIMENTS AND RESULTS

Here we provide more ablation experiments about GSL^2 in the same training framework as the Section 5.

We first experimented with selecting different numbers of landmarks on the performance of GSL^2, and the results are shown in the Tab 7, 8, and 9. From the experimental results, we can find that increasing the value of M can improve the accuracy of GSL^2, but it also increases the inference time. Considering the balance of inference speed and accuracy, we finally chose M as 60, 30, and 70 for three datasets, respectively.

Table 7: Experimental results for different numbers of landmarks were selected in the AIDS700nef dataset. M denotes the number of landmarks. Considering the balance of inference speed and accuracy, we finally chose M as 60.

M	GSL^2-R (s)	GSL^2-F (s)	MSE $\downarrow$	$\rho \uparrow$	$\tau \uparrow$	P@10 $\uparrow$	P@20 $\uparrow$
10	2.618	0.676	2.334	0.840	0.684	0.434	0.536
20	3.157	0.805	1.699	0.884	0.739	0.544	0.635
30	3.134	0.738	1.559	0.896	0.755	0.577	0.668
40	3.389	0.696	1.563	0.897	0.756	0.576	0.670
50	3.537	0.697	1.481	0.903	0.764	0.579	0.678
60	3.874	0.718	1.470	0.905	0.767	0.604	0.688
70	4.207	0.764	1.455	0.902	0.763	0.583	0.682
80	4.495	0.737	1.621	0.895	0.753	0.566	0.655
90	4.580	0.740	1.450	0.903	0.763	0.588	0.680
100	4.900	0.785	1.765	0.890	0.746	0.533	0.636
110	4.974	0.767	1.675	0.893	0.749	0.554	0.647
120	5.346	0.749	1.557	0.898	0.757	0.574	0.663

We next provide experimental results of GSL under different random seeds to test the sensitivity of GSL to the selected landmarks, which is shown in the Tab 10, 11, and 12, respectively.

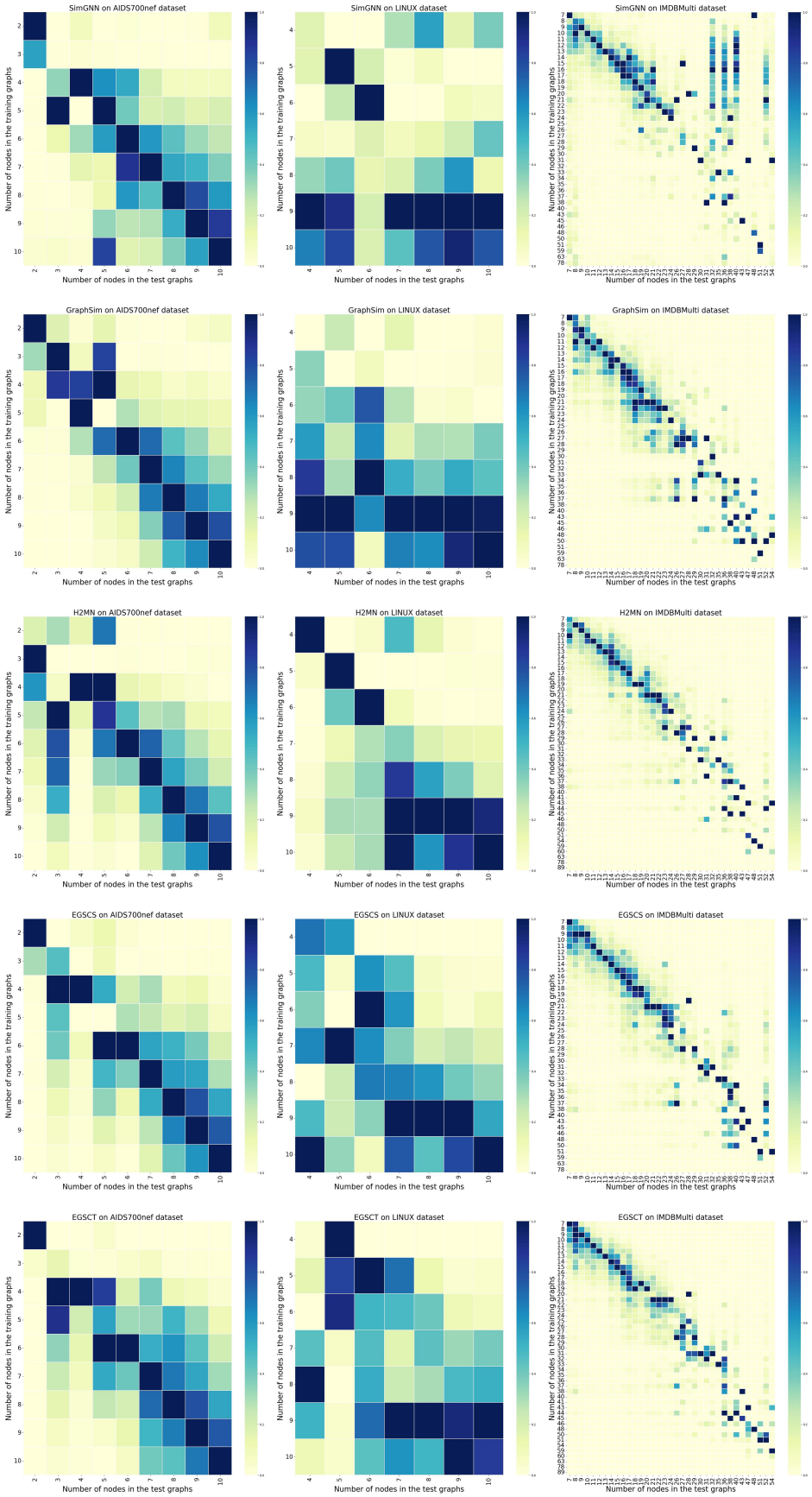


Figure 6: heatmap

Table 8: Experimental results for different numbers of landmarks were selected in the LINUX dataset. Considering the balance of inference speed and accuracy, we finally chose M as 30.

M	GSL^2-R (s)	GSL^2-F (s)	MSE $\downarrow$	$\rho \uparrow$	$\tau \uparrow$	P@10 $\uparrow$	P@20 $\uparrow$
10	4.254	1.015	0.696	0.981	0.920	0.944	0.939
20	4.283	1.035	0.382	0.987	0.944	0.975	0.951
30	5.007	1.159	0.074	0.994	0.964	0.995	0.991
40	5.235	1.132	0.080	0.994	0.967	0.990	0.988
50	5.467	1.052	0.084	0.994	0.967	0.990	0.993
60	5.753	1.017	0.081	0.994	0.967	0.991	0.993
70	6.018	1.040	0.088	0.993	0.960	0.987	0.989
80	6.634	1.039	0.209	0.991	0.957	0.973	0.960
90	6.763	0.970	0.077	0.994	0.964	0.989	0.989
100	7.703	1.115	0.130	0.993	0.962	0.982	0.976
110	7.752	1.111	0.212	0.990	0.947	0.974	0.960
120	8.268	1.111	0.087	0.994	0.966	0.993	0.994

Table 9: Experimental results for different numbers of landmarks were selected in the IMDBMulti dataset. Considering the balance of inference speed and accuracy, we finally chose M as 70.

M	GSL^2-R (s)	GSL^2-F (s)	MSE $\downarrow$	$\rho \uparrow$	$\tau \uparrow$	P@10 $\uparrow$	P@20 $\uparrow$
10	6.525	1.513	0.703	0.964	0.898	0.843	0.862
20	7.773	1.458	0.687	0.974	0.929	0.854	0.872
30	8.916	1.470	0.589	0.968	0.915	0.860	0.882
40	10.902	1.714	0.640	0.967	0.912	0.832	0.858
50	12.312	1.786	0.536	0.969	0.918	0.852	0.890
60	13.142	1.524	0.582	0.972	0.920	0.856	0.871
70	15.792	1.824	0.510	0.971	0.916	0.869	0.887
80	17.720	1.912	0.499	0.974	0.932	0.868	0.878
90	18.897	1.774	0.502	0.973	0.927	0.855	0.888
100	20.914	1.806	0.508	0.971	0.924	0.863	0.890
110	24.730	1.804	0.524	0.969	0.911	0.864	0.882
120	26.778	1.830	0.520	0.970	0.919	0.865	0.887

Table 10: Experimental results on GSL^2 using different random seeds on the AIDS700nef dataset.

Seed	MSE ↓	ρ ↑	τ ↑	P@10 ↑	P@20 ↑
2	1.470	0.905	0.767	0.604	0.688
3	1.499	0.904	0.765	0.599	0.681
4	1.509	0.904	0.765	0.591	0.682
5	1.477	0.902	0.763	0.574	0.667
2233	1.577	0.896	0.755	0.568	0.665
std	0.042	0.004	0.005	0.016	0.010

From the results, we can see that select different landmark affects the performance of GSL, but the effect is not significant, which shows the robustness of our GSL.

Table 11: Experimental results on GSL^2 using different random seeds on the LINUX dataset.

Seed	MSE ↓	ρ ↑	τ ↑	P@10 ↑	P@20 ↑
2	0.074	0.994	0.964	0.995	0.991
3	0.083	0.993	0.948	0.987	0.987
4	0.098	0.992	0.947	0.983	0.982
5	0.090	0.992	0.947	0.992	0.990
2233	0.084	0.993	0.948	0.984	0.991
std	0.009	0.001	0.008	0.005	0.004

Table 12: Experimental results on GSL^2 using different random seeds on the IMDBMulti dataset.

Seed	MSE ↓	ρ ↑	τ ↑	P@10 ↑	P@20 ↑
2	0.510	0.971	0.916	0.869	0.887
3	0.532	0.965	0.890	0.852	0.883
4	0.559	0.970	0.902	0.852	0.881
5	0.553	0.970	0.902	0.866	0.874
2233	0.562	0.968	0.898	0.863	0.881
std	0.022	0.002	0.009	0.008	0.005

We provide experiments using different regression algorithms in GSL^2 and the results are shown in the Tab 13 14, 15, respectively. The experimental setup is the same as in Section 5, and we use the default parameters in Pycaret (Ali, 2020) to train each model. In practice, the parameters of the learning algorithm can be adjusted to obtain better test results. It can be seen that the decision tree based regression algorithms achieve good performance in this noisy embeddings. This also illustrates the strong expressive ability of our generated embeddings.

Table 13: Experimental results on GSL^2 using different regression algorithms on the AIDS700nef dataset.

Model	MSE ↓	ρ ↑	τ ↑	P@10 ↑	P@20 ↑
Extra Trees	1.705	0.888	0.751	0.541	0.637
CatBoost	1.645	0.890	0.746	0.559	0.656
Random Forest	1.963	0.874	0.729	0.475	0.583
KNeighbors	1.909	0.872	0.732	0.561	0.657
XGBoost	1.870	0.874	0.724	0.524	0.618
LightGBM	2.317	0.849	0.694	0.456	0.567
Gradient Boosting	3.305	0.796	0.635	0.291	0.434
Decision Tree	4.854	0.743	0.609	0.382	0.479
Bayesian Ridge	6.108	0.504	0.383	0.066	0.139
Linear Regression	6.108	0.504	0.384	0.072	0.138
Ridge	6.108	0.504	0.384	0.071	0.138
Orthogonal Matching Pursuit	6.291	0.490	0.371	0.059	0.124
Huber	6.218	0.501	0.382	0.071	0.135
ElasticNet	6.470	0.483	0.372	0.077	0.143
Passive Aggressive Regressor	6.603	0.477	0.367	0.076	0.133
Lasso	7.325	0.432	0.333	0.076	0.143
AdaBoost	8.754	0.383	0.304	0.314	0.347
LassoLars	10.776	0.226	0.186	0.466	0.490
MLPs	1.470	0.905	0.767	0.604	0.688

Table 14: Experimental results on GSL^2 using different regression algorithms on the LINUX dataset.

Model	MSE ↓	ρ ↑	τ ↑	P@10 ↑	P@20 ↑
Extra Trees	0.078	0.999	0.996	0.987	0.991
Random Forest	0.103	0.998	0.994	0.981	0.981
XGBoost	0.175	0.992	0.964	0.978	0.967
KNeighbors	0.141	0.998	0.995	0.986	0.984
CatBoost	0.148	0.993	0.960	0.964	0.977
Decision Tree	0.222	0.997	0.994	0.987	0.988
LightGBM	0.980	0.979	0.921	0.951	0.949
Gradient Boosting	4.450	0.944	0.847	0.743	0.819
AdaBoost	24.366	0.591	0.472	0.298	0.300
ElasticNet	28.596	0.365	0.288	0.090	0.145
Orthogonal Matching Pursuit	30.056	0.319	0.265	0.045	0.045
Lasso	31.502	0.317	0.265	0.090	0.145
Huber	28.490	0.395	0.310	0.090	0.145
Bayesian Ridge	26.908	0.386	0.299	0.090	0.145
Ridge	26.908	0.388	0.305	0.090	0.145
Linear Regression	26.908	0.388	0.305	0.090	0.145
MLPs	0.074	0.994	0.964	0.995	0.991

Table 15: Experimental results on GSL^2 using different regression algorithms on the IMDBMulti dataset.

Model	MSE ↓	ρ ↑	τ ↑	P@10 ↑	P@20 ↑
Extra Trees	0.585	0.971	0.935	0.875	0.890
Random Forest	0.776	0.968	0.931	0.862	0.883
XGBoost	2.901	0.951	0.884	0.754	0.803
KNeighbors	0.837	0.961	0.924	0.876	0.899
LightGBM	13.573	0.925	0.837	0.728	0.758
Decision Tree	1.275	0.954	0.914	0.841	0.875
Gradient Boosting	48.646	0.825	0.700	0.206	0.320
Bayesian Ridge	75.886	0.333	0.250	0.015	0.034
Linear Regression	75.887	0.333	0.251	0.015	0.034
ElasticNet	75.690	0.330	0.252	0.013	0.033
Lasso	75.779	0.330	0.252	0.013	0.033
Ridge	75.959	0.336	0.254	0.015	0.036
Least Angle Regression	76.013	0.333	0.247	0.015	0.036
CatBoost	2.100	0.957	0.896	0.817	0.825
Huber	90.815	0.351	0.265	0.025	0.032
Orthogonal Matching Pursuit	174.995	0.236	0.178	0.022	0.045
MLPs	0.510	0.971	0.916	0.869	0.887