Learning-Based Efficient Graph Similarity Computation via Multi-Scale Convolutional Set Matching

Yunsheng Bai,[*1] **Hao Ding,**[*2†] **Ken Gu,**[1] **Yizhou Sun,**[1] **Wei Wang**[1]
[1]University of California, Los Angeles, [2]AWS AI Labs
yba@ucla.edu, haodin@amazon.com, ken.qgu@gmail.com, {yzsun, weiwang}@cs.ucla.edu

Abstract

Graph similarity computation is one of the core operations in many graph-based applications, such as graph similarity search, graph database analysis, graph clustering, etc. Since computing the exact distance/similarity between two graphs is typically NP-hard, a series of approximate methods have been proposed with a trade-off between accuracy and speed. Recently, several data-driven approaches based on neural networks have been proposed, most of which model the graph-graph similarity as the inner product of their graph-level representations, with different techniques proposed for generating one embedding per graph. However, using one fixed-dimensional embedding per graph may fail to fully capture graphs in varying sizes and link structures—a limitation that is especially problematic for the task of graph similarity computation, where the goal is to find the fine-grained difference between two graphs. In this paper, we address the problem of graph similarity computation from another perspective, by directly matching two sets of node embeddings without the need to use fixed-dimensional vectors to represent whole graphs for their similarity computation. The model, GRAPHSIM, achieves the state-of-the-art performance on four real-world graph datasets under six out of eight settings (here we count a specific dataset and metric combination as one setting), compared to existing popular methods for approximate Graph Edit Distance (GED) and Maximum Common Subgraph (MCS) computation.

1 Introduction

Recent years we have witnessed the growing importance of graph-based applications in the domains of chemistry, bioinformatics, recommender systems, social network study, static program analysis, etc. One of the fundamental problems related to graphs is the computation of distance/similarity between two graphs. It not only is a core operation in graph similarity search and graph database analysis (Zeng et al. 2009; Wang et al. 2012), but also plays a significant role in a wide range of applications. For example, in computer security, similarity between binary functions is useful for plagiarism and malware detection (Xu et al. 2017); in anomaly detection, similarity between communication graphs could help identify network intrusions from the graph-based connection records (Noble and Cook 2003); in social network analysis, similarity between different user message graphs may reveal interesting behavioral patterns (Koutra, Vogelstein, and Faloutsos 2013).

Among various definitions of graph similarity/distance, Graph Edit Distance (GED) (Bunke 1983) and Maximum Common Subgraph (MCS) (Bunke and Shearer 1998) are two popular and domain-agnostic metrics. However, the computation of exact GED and MCS is known to be NP-hard (Zeng et al. 2009; Bunke and Shearer 1998), incurring significant computational burden in practice (Blumenthal and Gamper 2018). For example, a recent study shows that even the state-of-the-art algorithms cannot reliably compute the exact GED between graphs of more than 16 nodes within a reasonable time (Blumenthal and Gamper 2018).

Given the great significance yet huge challenge of computing the exact graph distance/similarity, various approximate algorithms have been proposed to compute the graph distance/similarity in a fast but heuristic way, including traditional algorithmic approaches (Riesen and Bunke 2009; Fankhauser, Riesen, and Bunke 2011; Daller et al. 2018) as well as more recent data-driven neural network approaches (Riba et al. 2018; Ktena et al. 2017; Bai et al. 2019; Li et al. 2019).

Compared with traditional algorithmic approaches which typically involve knowledge and heuristics specific to a metric, the neural network approaches *learn* graph similarity from data: During training, the parameters are learned by minimizing the loss between the predicted similarity scores and the ground truth; during testing, unseen pairs of graphs can be fed into these models for fast approximation of their similarities.

However, a major limitation of most current neural network models is that they rely on graph-level embeddings to model the similarity of graphs: Each graph is first represented as a fixed-length vector, and then the similarity of two graphs can be modeled as a vector operation on the two embeddings, e.g. cosine similarity. However, real-world graphs typically come in very different sizes, which the fixed-length vector

[*]The two first authors made equal contributions.
[†]This work is done before Hao Ding joined AWS AI Labs.

representation may fail to fully capture. Even when the two graphs of interest are similar in sizes, the actual difference between them can lie in very small local substructures, which is hard to be captured by the single vector. This is especially problematic for the task of graph similarity computation, where the goal is to compare the difference between all nodes and edges of the two graphs. For simple or regular graphs, this approach may work well, but for more complicated scenarios in which graphs are of very different structures and/or the task is to find the fine-grained node-node correspondence (Zanfir and Sminchisescu 2018), this approach often produces less effective models.

In this paper, we propose to avoid the generation of graph-level embeddings, and instead directly perform neural operations on the two sets of node embeddings. Inspired by two classic families of algorithms for graph similarity/distance (Nikolentzos, Meladianos, and Vazirgiannis 2017; Riesen and Bunke 2009; Fankhauser, Riesen, and Bunke 2011), our model GRAPHSIM turns the two sets of node embeddings into a similarity matrix consisting of the pairwise node-node similarity scores, and is trained in an end-to-end fashion (Fig. 2). By carefully ordering the nodes in each graph, the similarity matrix encodes the similarity patterns specific to the graph pair, which allows the standard image processing techniques to be adapted to model the graph-graph similarity. The new challenges in the graph setting compared to the standard image processing using Convolutional Neural Networks (CNN) are that:

- ***Permutation invariance***. The same graph can be represented by different adjacency matrices by permuting the order of nodes, and the model should not be sensitive to such permutation.
- ***Spatial locality preservation***. CNN architectures assume the input data has spatial locality, i.e, close-by data points are more similar to each other. How to make our embedding-based similarity matrix preserve such spatial locality is important.
- ***Graph size invariance***. The CNN architecture requires fixed-length input. How to handle graphs with different sizes is another question to address.
- ***Multi-scale comparison.*** Finally, graphs naturally contain patterns of different scales that may be unknown in advance. The algorithm should be able to capture and leverage structural information and features of multiple granularities.

To tackle these challenges, we propose GRAPHSIM, which addresses the graph similarity computation task in a novel way by direct usage of node-level embeddings via pairwise node-node similarity scores. We show that GRAPHSIM can be combined with various node embedding approaches, improving performance on four graph similarity computation datasets under six out of eight settings. Finally, we show that GRAPHSIM can learn interpretable similarity patterns that exist in the input graph pairs.

2 Related Work

Graph Representation Learning Over the years, there have been a great number of works dealing with the representation of nodes (Hamilton, Ying, and Leskovec 2017), and graphs (Ying et al. 2018). Among the node embedding methods, neighbor aggregation based methods, e.g. GCN (Kipf and Welling 2016), GraphSAGE (Hamilton, Ying, and Leskovec 2017), GIN (Xu et al. 2019), etc., are permutation-invariant, and have gained a lot of attention.

Neural network based methods have been used in a broad range of graph applications, most of which are framed as node-level prediction tasks (Hamilton et al. 2018) or single graph classification (Ying et al. 2018). In this work, we consider the task of graph similarity computation, which is under the general problem of graph matching (Emmert-Streib, Dehmer, and Shi 2016).

Text and Graph Matching with Neural Networks Text matching has a long history with many successful applications (Mitra, Diaz, and Craswell 2017). Among various methods for text matching, promising results in matching sequences of word embeddings (He and Lin 2016) inspire us to explore the potential of using node embeddings for the task of graph matching directly without graph-level representations. In contrast, neural network based graph matching remains largely unexplored, and most existing works still rely on first generating one embedding per graph using graph neural networks, and then modeling the graph-graph similarity using the two graph-level representations.

We examine several existing works on similarity computation for graphs: (1) SIAMESE MPNN (SMPNN) (Riba et al. 2018) is an early work that models the similarity as a simple summation of certain node-node similarity scores. (2) GCNMEAN and GCNMAX (Ktena et al. 2017) apply the GCN architectures with graph coarsening (Defferrard, Bresson, and Vandergheynst 2016) to generate graph-level embeddings for the similarity. (3) SIMGNN (Bai et al. 2019) attempts to use node-node similarity scores by taking their histogram features, but still largely relies on the graph-level embeddings due to the histogram function being non-differentiable. (4) GMN (Li et al. 2019) is a recent work which manages to introduce node-node similarity information into graph-level embeddings via a cross-graph attention mechanism, but the cross-graph communication only updates the node embeddings, and still generates one embedding per graph from the updated node embeddings.

3 Problem Definition

Graphs are data structures with a node set $\mathcal{V}$ and a edge set $\mathcal{E}$, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. The number of nodes of $\mathcal{V}$ is denoted as $N = |\mathcal{V}|$. Each node and edge can be associated with labels, such as atom and chemical bond type in a molecular graph. In this study, we confine our graphs as undirected and unweighted graphs, but it is not hard to extend GRAPHSIM to other types of graphs, since GRAPHSIM is a general framework for graph similarity computation.

Given two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, different distance/similarity metrics can be defined.

Graph Edit Distance (GED) The edit distance between

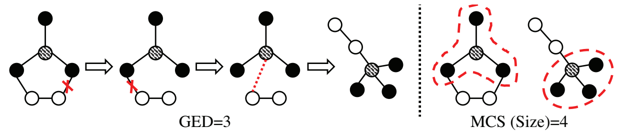

Figure 1: The GED is 3, as the transformation needs 3 edit operations: Two edge deletions, and an edge insertion. The MCS size is 4.

two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ is the number of edit operations in the optimal alignments that transform $\mathcal{G}_1$ into $\mathcal{G}_2$, where an edit operation on a graph $\mathcal{G}$ is an insertion or deletion of a node/edge or relabelling of a node [1]. We transform GED into a similarity metric ranging between 0 and 1 using a one-to-one mapping function.

Maximum Common Subgraph (MCS) A Maximum Common Subgraph of two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ is a subgraph common to both $\mathcal{G}_1$ and $\mathcal{G}_2$ such that there is no other subgraph of $\mathcal{G}_1$ and $\mathcal{G}_2$ with more nodes. The MCS definitions have two variants (Raymond and Willett 2002): In one definition, the MCS must be a connected graph; in the other, the MCS can be disconnected. In this paper, we adopt the former definition. For $\mathcal{G}_1$ and $\mathcal{G}_2$, their similarity score is defined as the number of nodes in their MCS, i.e. $|\mathrm{MCS}(\mathcal{G}_1, \mathcal{G}_2)|$.

Learning-based Graph Similarity Computation Given a graph similarity definition, our goal is to learn a neural network based function that takes two graphs as input and outputs the desired similarity score through training, which can be applied to any unseen graphs for similarity computation at the test stage.

In later sections, we will introduce how to design a neural network architecture to serve this purpose, and why such design is reasonable by providing connections to set matching.

4 The Proposed Approach: GraphSim

GraphSim consists of the following sequential stages: 1) *Multi-scale neighbor aggregation layers* generate vector representations for each node in the two graphs at different scales; 2) *Similarity matrix generation layers* compute the inner products between the embeddings of every pair of nodes in the two graphs, resulting in multiple similarity matrices capturing the node-node interaction scores at different scales; 3) *CNN layers* convert the similarity computation problem into a pattern recognition problem, which provides multi-scale features to a *fully connected network* to obtain a final predicted graph-graph similarity score. An overview of our model is illustrated in Fig. 2.

Multi-Scale Neighbor Aggregation

We build upon an active line of research on graph neural networks for generating node representations. Graph Convolutional Networks (GCN) (Defferrard, Bresson, and Vandergheynst 2016; Kipf and Welling 2016), for example, is a neighbor aggregation approach which generates node embeddings from the local substructure information of each node, which is an inductive method and can be applied to unseen nodes. In Fig. 2, different node types are represented by different colors and one-hot encoded as the initial node representation. For graphs with unlabeled nodes, we use the same constant vector as the initial representation.

The core operation, graph convolution, operates on the representation of a node, which is denoted as $\boldsymbol{u_i} \in \mathbb{R}^D$, and is defined as follows:

$$\mathrm{conv}(\boldsymbol{u_i}) = \mathrm{ReLU}(\sum_{j \in \mathcal{N}(i)} \frac{1}{\sqrt{d_i d_j}} \boldsymbol{u}_j \boldsymbol{W}^{(n)} + \boldsymbol{b}^{(n)}) \quad (1)$$

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

Intuitively, the graph convolution operation aggregates the features from the first-order neighbors of the node. Since applying the GCN layer once on a node can be regarded as aggregating the representations of its first-order neighbors and itself, sequentially stacking L layers would cause the final representation of a node to include its L-th order neighbors. In other words, the more GCN layers, the larger the scale of the learned embeddings.

Multi-Scale GCN The potential issue of using a deep GCN architecture is that the embeddings may lose subtle patterns in local neighborhood after aggregating neighbors multiple times. The issue is especially severe when the two graphs are very similar, and the differences mainly lie in small local substructures.

One natural way for humans to compare the difference between two graphs is to recursively break down the whole graph into its compositional subgraphs via a top-down approach. Each subgraph is further decomposed into additional subgraph levels, until the entire specification is reduced to node level, producing a hierarchy of subgraph (de)composition. We therefore propose a multi-scale framework to extract the output of each of the many GCN layers for the construction of similarity matrices, which is shown next.

GraphSim is a general framework for similarity computation, and can work with GCN and any of its successors such as GraphSAGE (Hamilton, Ying, and Leskovec 2017), GIN (Xu et al. 2019), etc.

Similarity Matrix Generation

The use of node-node similarity scores roots in some classic approaches to modeling graph similarity, such as the Optimal Assignment Kernels for graph classification and the Bipartite Graph Matching for GED approximation. The detailed connection between the proposed model and these methods is in the supplementary material. Here we present some key properties of these methods that are needed to motivate the usage of node-node similarity scores.

[1]Other variants of GED definitions exist (Riesen, Emmenegger, and Bunke 2013), and we adopt this basic version for each in this work.

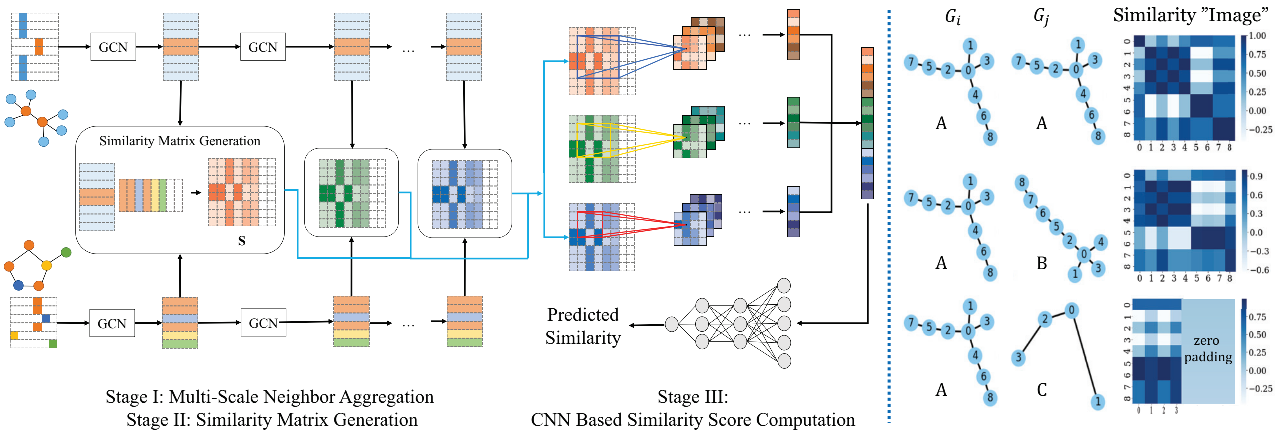
GCN
GCN
...
Similarity Matrix Generation
S
GCN
GCN
...
...
Predicted Similarity
Stage I: Multi-Scale Neighbor Aggregation
Stage II: Similarity Matrix Generation
Stage III:
CNN Based Similarity Score Computation
G_i
G_j
Similarity "Image"
A
A
A
B
A
C
zero padding

Figure 2: Left: An overview illustration of our proposed method GRAPHSIM. No graph-level representation is generated, and it directly uses the node-node similarity scores in the three similarity matrices corresponding to node embeddings at different scales. Right: Illustration of three similarity matrices from the LINUX (Wang et al. 2012) dataset. For two isomorphic graphs (A and A), a strong symmetric diagonal block pattern is observed; for two less similar graphs (A and B), the dark diagonal pattern is less evident; for two graphs that are not similar at all (A and C), the symmetric block pattern is almost gone. For graphs of different sizes, we devise a consistent max padding scheme. Intuitively, the block patterns can be thought of as graph-graph similarity patterns at different scales, which are suitable for CNNs to capture.

Optimal Assignment Kernels Given two graphs with node embeddings $\boldsymbol{X} \in \mathbb{R}^{N_1 \times D}$ and $\boldsymbol{Y} \in \mathbb{R}^{N_2 \times D}$, the Earth Mover's Distance graph kernel (Nikolentzos, Meladianos, and Vazirgiannis 2017) solves the following transportation problem (Rubner, Tomasi, and Guibas 2000):

$$\min \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} \boldsymbol{T}_{ij} ||\boldsymbol{x}_i - \boldsymbol{y}_j||_2 \tag{2}$$

subject to several constraints. Intuitively, each of the two graphs is represented as a set of node embeddings, and the kernel finds the optimal way to transform one set of node embeddings to the other, with the cost to transform one pair of nodes being their Euclidean distance.

Bipartite Graph Matching The HUNGARIAN (Riesen and Bunke 2009) and VJ (Fankhauser, Riesen, and Bunke 2011) algorithms for approximate GED computation solve the following assignment problem form:

$$\min \sum_{i=1}^{N'} \sum_{j=1}^{N'} \boldsymbol{T}_{ij} \boldsymbol{C}_{ij} \tag{3}$$

subject to several constraints with cost matrix $\boldsymbol{C}$ denoting the insertion, deletion, and substitution costs associated with the GED metric for every node pair in the two graphs.

Despite different goals, both the kernel method and GED approximate algorithms work directly on node-level information without the whole-graph representations. Specifically, both need the pairwise node-node distance scores between the two graphs, with different definitions of node-node distances.

Since GRAPHSIM is trained end-to-end for graph similarity computation, we can calculate the inner products between all pairs of node embeddings in the two graphs at multiple scales, resulting in multiple similarity matrices. Treat each matrix as an image, the task of graph similarity measurement is viewed as an image processing problem in which the goal is to discover the optimal node matching pattern encoded in the image by applying CNNs. Consider this process as a similarity operator that transforms two sets of node embeddings into a score. Then GRAPHSIM can be regarded as:

$$\min(h_{\boldsymbol{\Theta}}(\boldsymbol{X}, \boldsymbol{Y}) - s_{ij})^2 \tag{4}$$

where $h_{\Theta}(\boldsymbol{X}, \boldsymbol{Y})$ denotes the similarity matrix generation and its subsequent layers. The training is guided by the true similarity score s_{ij} for the update of weights $\boldsymbol{\Theta}$ associated with the neighbor aggregation layers that generate node embeddings $\boldsymbol{X}$ and $\boldsymbol{Y}$ as well as the subsequent CNNs. In contrast, for the optimization problems (2) and (3), in order to find the optimal value as the computed graph distance, the problem must be solved explicitly (Kuhn 1955; Jonker and Volgenant 1987).

BFS Ordering Different from pixels of images or words of sentences, nodes of a graph typically lack ordering. A different node ordering would lead to a different similarity matrix. To completely solve the graph node permutation problem, we need to find one canonical ordering for each graph in a collection of graphs, which is NP-hard as shown in an early work on CNN for graphs (Niepert, Ahmed, and Kutzkov 2016). Moreover, the CNNs require spatial locality preservation. To alleviate these two issues, we utilize the breadth-first-search (BFS) node-ordering scheme proposed in GraphRNN (You et al. 2018) to sort and reorder the node embeddings. Since BFS is performed on the graph, nearby nodes are ordered close to each other. It is worth noting that the BFS ordering scheme achieves a reasonable trade-off between efficiency and uniqueness of ordering, as it requires quadratic operations in the worst case (i.e. complete graphs) (You et al. 2018).

Besides the issues related to node permutation and spatial locality, one must address the challenge posed by graph size variance and the fact that CNN architecture requires fix-length input. To preserve the information of graph size variance while fix the size of similarity matrix, we propose the solutions as follows:

Max Padding One can fix the number of nodes in each graph by adding fake nodes to a pre-defined number, and therefore lead to a fix-size similarity matrix. However, such matrix will completely ignore the graph size information, which is pivotal as seen in the definitions of both GED and MCS. For example, the similarity matrix between two small but isomorphic graphs may be padded with a lot of zeros, potentially misleading the CNNs to predict they are dissimilar.

To reflect the difference in graph sizes in the similarity matrix, we devise max padding. Suppose $\mathcal{G}_1$ and $\mathcal{G}_2$ contain N_1 and N_2 nodes respectively, we pad $|N_1 - N_2|$ rows of zeros to the node embedding matrix of the smaller of the two graphs, so that both graphs contain $\max(N_1, N_2)$ nodes.

Matrix Resizing To apply CNNs to the similarity matrices, We resize the simlarity matrices through image resampling (Thévenaz, Blu, and Unser 2000). For implementation, we choose the bilinear interpolation, and leave the exploration of more advanced techniques for future work. The resulting similarity matrix $\boldsymbol{S}$ has fixed shape $M \times M$, where M is a hyperparameter controlling the degree of information loss caused by resampling.

The following equation summarizes the similarity matrix generation process:

$$\boldsymbol{S} = \mathrm{RES}_M(\widetilde{\boldsymbol{H}}_1 \widetilde{\boldsymbol{H}}_2^T) \tag{5}$$

where $\widetilde{\boldsymbol{H}}_i \in \mathbb{R}^{\max(N_1,N_2)\times D}, i \in \{1,2\}$ is the padded node embedding matrix $\boldsymbol{H}_i \in \mathbb{R}^{N_i \times D}, i \in \{1,2\}$ with zero or $|N_1 - N_2|$ nodes padded, and $\mathrm{RES}(\cdot) : \mathbb{R}^{\max(N_1,N_2)\times\max(N_1,N_2)} \mapsto \mathbb{R}^{M\times M}$ is the resizing function, where M is a hyperparameter controlling the degree of information loss caused by resampling.

CNN Based Similarity Score Computation

We feed these matrices through multiple CNNs in parallel. As shown in Fig. 2, three CNN "channels" are used, each with its own CNN filters. At the end, the results are concatenated and fed into multiple fully connected layers, so that a final similarity score $\hat{s}_{ij}$ is generated for the graph pair $\mathcal{G}_i$ and $\mathcal{G}_j$. The mean square error loss function is used to train our model: $\mathcal{L} = \frac{1}{|\mathcal{D}|}\sum_{(i,j)\in\mathcal{D}}(\hat{s_{ij}} - s_{ij})^2$ where $\mathcal{D}$ is the set of training graph pairs, and s_{ij} is the true similarity score coming from any graph similarity metric. For GED, we apply a one-to-one mapping function to transform the true distance score into the true similarity score; for MCS, the normalized MCS size is treated as the true similarity score.

5 Experiments

We evaluate our model, GraphSim, against a number of state-of-the-art approaches to GED and MCS computation, with the major goals of addressing the following questions:

Q1 How accurate (effective) and fast (efficient) is GraphSim compared to the state-of-the-art approaches for graph similarity computation, including both approximate similarity computation algorithms and neural network based models?

Q2 How do the proposed ordering, resizing, and multi-scale comparison techniques help with the CNN-based GraphSim model?

Q3 Does GraphSim yield meaningful and interpretable similarity matrices/images on the input graph pairs?

Datasets To probe the ability of GraphSim to compute graph-graph similarities from graphs in different domains, we evaluate on four real graph datasets, Aids, Linux, Imdb, and Ptc, whose detailed descriptions and statistics can be found in the supplementary material.

For each dataset, we split it into training, validation, and testing sets by 6:2:2, and report the averaged *Mean Squared Error (mse)*, *Spearman's Rank Correlation Coefficient (ρ)* (Spearman 1904), *Kendall's Rank Correlation Coefficient (τ)* (Kendall 1938), and *Precision at k ($p@k$)* to test the accuracy and ranking performance of each GED and MCS computation method. The supplementary material contains more details on the data preprocessing, parameter settings, result analysis, efficiency comparison, as well as parameter sensitivity study.

Baseline Methods We consider both the state-of-the-art GED/MCS computation methods and baselines using neural networks. To ensure consistency, all neural network models use GCN for node embeddings except for GMN, and to demonstrate the flexibility of our framework, we show the performance improvement of GraphSim by replacing GCN with the more powerful GMN's node embedding methods in the supplementary material.

Effectiveness

As shown in Table 1 and 2, our model, GraphSim, consistently achieves the best results on all metrics across all the datasets with both the GED and MCS metrics. Specifically, GraphSim achieves the smallest error and the best ranking performance on the task of graph similarity computation under six out of eight settings. We repeated the running of our model 10 times on Aids, and the standard deviation of mse is $4.56 * 10^{-5}$. The Aids dataset is relatively small in terms of the average number of nodes per graph, potentially causing the CNN model to overfit. In the supplementary material, we show that as a general framework, by replacing GCN with GMN's node embedding method, GraphSim achieves the best performance than all methods.

Efficiency

In Fig. 3, the results are averaged across queries and in wall time. EmbAvg is the fastest method among all, but its performance is poor, since it simply takes the dot product between two graph-level embeddings (average of node embeddings) as the predicted similarity score. Beam and Hungarian run fast on Linux, but due to their higher time complexity, they scale poorly on the largest dataset, Imdb. The exact

Table 1: Effectiveness results on the GED metric. On Aids and Linux, A* provides ground-truth results, labeled with superscript $*$. On Imdb and Ptc, A* fails to compute most GEDs within a 5-minute limit (thus denoted as $-$). Instead, the minimum GED returned by Beam, Hungarian, and VJ for each pair is used as the ground-truth GED. The mse is in 10^{-3}.

	Aids			Linux			Imdb			Ptc		
Method	**mse**	ρ	**p@10**	**mse**	ρ	**p@10**	**mse**	ρ	**p@10**	**mse**	ρ	**p@10**
A*	0.000*	1.000*	1.000*	0.000*	1.000*	1.000*	–	–	–	–	–	–
Beam	12.090	0.609	0.481	9.268	0.827	0.973	2.413*	0.905*	0.803*	0.552*	0.998*	0.982*
Hungarian	25.296	0.510	0.360	29.805	0.638	0.913	1.845*	0.932*	0.825*	112.326*	0.919*	0.159*
VJ	29.157	0.517	0.310	63.863	0.581	0.287	1.831*	0.934*	0.815*	154.791*	0.904*	0.102*
HED	28.925	0.621	0.386	19.553	0.897	0.982	19.400	0.751	0.801	978.318	0.919	0.169
Smpnn	5.184	0.294	0.032	11.737	0.036	0.009	32.596	0.107	0.023	116.473	0.148	0.082
EmbAvg	3.642	0.601	0.176	18.274	0.165	0.071	71.789	0.229	0.233	32.601	0.393	0.173
GCNMean	3.352	0.653	0.186	8.458	0.419	0.141	68.823	0.402	0.200	6.525	0.546	0.150
GCNMax	3.602	0.628	0.195	6.403	0.633	0.437	50.878	0.449	0.425	7.501	0.506	0.152
SimGNN	1.189	0.843	0.421	1.509	0.939	0.942	1.264	0.878	0.759	0.850	0.944	0.507
GMN	1.886	0.751	0.401	1.027	0.933	0.833	4.422	0.725	0.604	1.613	0.672	0.262
GraphSim	**0.787**	**0.874**	**0.534**	**0.058**	**0.981**	**0.992**	**0.743**	**0.926**	**0.828**	**0.749**	**0.956**	**0.529**

Table 2: Effectiveness results on the MCS metric. On all the datasets, Mcsplit provides ground-truth results, labeled with superscript $*$. The mse is in 10^{-3}.

	Aids			Linux			Imdb			Ptc		
Method	**mse**	ρ	**p@10**	**mse**	ρ	**p@10**	**mse**	ρ	**p@10**	**mse**	ρ	**p@10**
Mcsplit	0.000*	1.000*	1.000*	0.000*	1.000*	1.000*	0.000*	1.000*	1.000*	0.000*	1.000*	1.000*
Smpnn	4.592	0.755	0.348	3.558	0.126	0.236	11.018	0.330	0.003	11.001	0.502	0.146
EmbAvg	6.466	0.701	0.236	2.663	0.427	0.343	17.853	0.524	0.166	23.018	0.556	0.302
GCNMean	5.956	0.776	0.316	2.706	0.439	0.368	9.316	0.753	0.364	9.166	0.712	0.337
GCNMax	5.525	0.782	0.328	2.466	0.543	0.440	8.234	0.796	0.435	7.905	0.752	0.385
SimGNN	3.433	0.822	0.374	0.729	0.859	0.850	1.153	0.938	0.705	3.781	0.782	0.279
GMN	**1.750**	**0.909**	**0.591**	0.598	0.906	0.830	0.590	0.941	0.795	3.835	0.841	**0.530**
GraphSim	2.402	0.858	0.505	**0.164**	**0.962**	**0.951**	**0.307**	**0.976**	**0.817**	**2.268**	**0.854**	0.504

MCS solver Mcsplit is the state-of-the-art for MCS computation, and runs faster than the exact GED solver, A* on all datasets. However, in general, neural network based models are still much faster than these solvers, since they enjoy lower time complexity in general and additional benefits from parallelizability and acceleration provided by GPU.

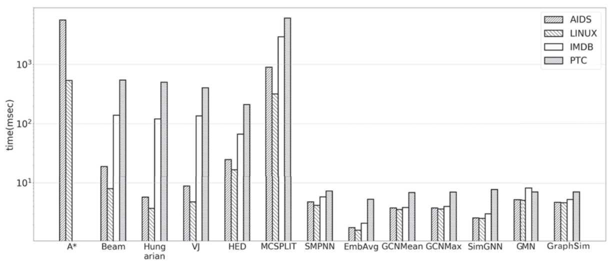

Figure 3: Running time comparison. The y-axis uses the log scale. The running time is averaged across queries. For neural network models, the running time for GED and MCS computation is very close to each other, so we take the average of the two.

Analysis of Various Proposed Techniques in GraphSim

To address **Q2**, we conduct several experiments by comparing GraphSim with three simpler variants, whose results are shown in Table 3. Among the three proposed techniques (i.e., max padding plus matrix resizing, ordering, and multi-scale comparison), max padding and matrix resizing affects the performance the most: Comparing GS-Pad with GS-Resize on Imdb and Ptc, our proposed padding and resizing technique greatly reduces the approximation error. Such significant improvements on Imdb and Ptc can be attributed to the large average graph size and graph size variance, as seen from the dataset statistics in the supplementary material. Besides, by comparing the GS-Resize and GraphSim, we can see a performance boost brought by the multi-scale framework. The advantage of BFS ordering can be observed by comparison of GS-NoOrd and GraphSim.

Analysis of Similarity "Images" in GraphSim

We demonstrate six similarity matrices (plotted as heatmap images) generated by GraphSim on Aids in Fig. 4. Node ids come from BFS ordering. Since both pairs are quite similar, as mentioned in Section 4, we expect to see block patterns, which are actually observed in the six similarity matrices. From pair (1) (the top row), we can see that larger scale (the blue matrix) helps distinguish between nodes 3 and 4

Table 3: GS-PAD and GS-RESIZE perform node ordering and use the node embeddings only by the last GCN layer to generate the similarity matrix. Since CNNs require fixed-length input, if the model does not use resizing, a simple way is to zero pad each similarity matrix to $N_{\max}$ by $N_{\max}$ (maximum graph size in the entire dataset), denoted as GS-PAD. GS-RESIZE uses the proposed max padding and resizing techniques. GS-NOORD uses all the proposed techniques including multi-scale comparison except for node ordering. The results are on the GED metric. The mse is in 10^{-3}.

	AIDS			LINUX			IMDB			PTC		
Method	**mse**	ρ	**p@10**	**mse**	ρ	**p@10**	**mse**	ρ	**p@10**	**mse**	ρ	**p@10**
GS-PAD	0.807	0.863	0.514	0.141	0.988	0.983	6.455	0.661	0.552	6.808	0.637	0.481
GS-RESIZE	0.811	0.866	0.499	0.103	0.991	0.986	0.896	0.914	0.810	0.791	0.948	0.513
GS-NOORD	0.803	0.867	0.478	0.071	0.983	0.976	1.105	0.902	0.744	0.838	0.945	0.515
GRAPHSIM	**0.787**	**0.874**	**0.534**	**0.058**	**0.993**	**0.981**	**0.743**	**0.926**	**0.828**	**0.749**	**0.956**	**0.529**

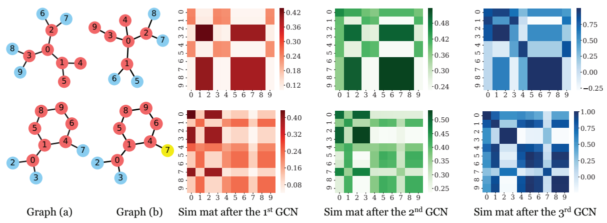

Figure 4: Similarity "images" on two pairs of graphs from AIDS trained with the GED metric. Different heatmap colors differentiate "images" from different scales of comparison. Top pair: GED=2—an edge deletion (edge between node 0 and 4 of graph (b)) and an edge addition (edge between node 3 and 4 of graph (b)). Bottom pair: GED=1—a node relabeling (node 7).

of graph (b), which contributes to the GED edit sequence as shown in the caption of Fig 4. From pair (2) (the bottom row), we can see that the smaller scale (the red matrix) helps differentiate between node 7 in graph (a) and node 7 in graph (b). Notice that comparing at larger scales does not help tell their difference in node types, because their structural equivalence causes their similarities to be higher as seen in the green and blue matrices. Thus, Fig. 4 shows the importance of using multi-scale similarity matrices rather than a single one.

Case Studies

We demonstrate four example queries, one from each dataset in Fig. 5. In each demo, the top row depicts the query along with the ground-truth ranking results, labeled with their normalized GEDs to the query. The bottom row shows the graphs returned by our model, each with its rank shown at the top. GRAPHSIM is able to retrieve graphs similar to the query. For example, in the case of LINUX, the top 6 results are exactly the isomorphic graphs to the query.

6 Conclusion

We introduced a CNN based method for better graph similarity computation. Using GRAPHSIM in conjunction with existing methods for generating node embeddings, we improve the performance in several datasets on two graph proximity metrics: Graph Edit Distance (GED) and Maximum Common Subgraph (MCS). Interesting future directions include using hierarchical graph representation learning techniques to reduce computational time complexity involved in the node-node similarity computation, and applying the CNN based graph matching method to other graph matching tasks, e.g. network alignment, as well as the explicit generation of edit sequence for GED and node correspondence for MCS.

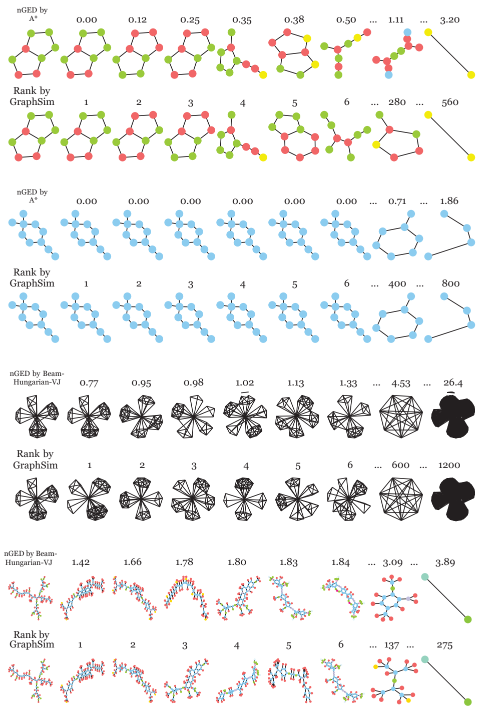

Figure 5: Visualization of ranking results under the GED metric. From top to bottom: AIDS, LINUX, IMDB, PTC.

Acknowledgments This work is partially supported by NSF III-1705169, NSF CAREER Award 1741634, NSF #1937599, Okawa Foundation Grant, and Amazon Research Award.

References

Bai, Y.; Ding, H.; Bian, S.; Chen, T.; Sun, Y.; and Wang, W. 2019. Simgnn: A neural network approach to fast graph similarity computation. *WSDM*.

Blumenthal, D. B., and Gamper, J. 2018. On the exact computation of the graph edit distance. *Pattern Recognition Letters*.

Bunke, H., and Shearer, K. 1998. A graph distance metric based on the maximal common subgraph. *Pattern recognition letters* 19(3-4):255–259.

Bunke, H. 1983. What is the distance between graphs. *Bulletin of the EATCS* 20:35–39.

Daller, É.; Bougleux, S.; Gaüzère, B.; and Brun, L. 2018. Approximate graph edit distance by several local searches in parallel. In *ICPRAM*.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 3844–3852.

Emmert-Streib, F.; Dehmer, M.; and Shi, Y. 2016. Fifty years of graph matching, network alignment and network comparison. *Information Sciences* 346:180–197.

Fankhauser, S.; Riesen, K.; and Bunke, H. 2011. Speeding up graph edit distance computation through fast bipartite matching. In *GbRPR*, 102–111. Springer.

Hamilton, W. L.; Bajaj, P.; Zitnik, M.; Jurafsky, D.; and Leskovec, J. 2018. Querying complex networks in vector space. *NIPS*.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *NIPS*, 1024–1034.

He, H., and Lin, J. 2016. Pairwise word interaction modeling with deep neural networks for semantic similarity measurement. In *NAACL HLT*, 937–948.

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

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

Kipf, T. N., and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *ICLR*.

Koutra, D.; Vogelstein, J. T.; and Faloutsos, C. 2013. Deltacon: A principled massive-graph similarity function. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, 162–170. SIAM.

Ktena, S. I.; Parisot, S.; Ferrante, E.; Rajchl, M.; Lee, M.; Glocker, B.; and Rueckert, D. 2017. Distance metric learning using graph convolutional networks: Application to functional brain networks. In *MICCAI*, 469–477. Springer.

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

Li, Y.; Gu, C.; Dullien, T.; Vinyals, O.; and Kohli, P. 2019. Graph matching networks for learning the similarity of graph structured objects. *ICML*.

Mitra, B.; Diaz, F.; and Craswell, N. 2017. Learning to match using local and distributed representations of text for web search. In *WWW*, 1291–1299. International World Wide Web Conferences Steering Committee.

Niepert, M.; Ahmed, M.; and Kutzkov, K. 2016. Learning convolutional neural networks for graphs. In *ICML*, 2014–2023.

Nikolentzos, G.; Meladianos, P.; and Vazirgiannis, M. 2017. Matching node embeddings for graph similarity. In *AAAI*, 2429–2435.

Noble, C. C., and Cook, D. J. 2003. Graph-based anomaly detection. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 631–636. ACM.

Raymond, J. W., and Willett, P. 2002. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of computer-aided molecular design* 16(7):521–533.

Riba, P.; Fischer, A.; Lladós, J.; and Fornés, A. 2018. Learning graph distances with message passing neural networks. In *ICPR*, 2239–2244.

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

Riesen, K.; Emmenegger, S.; and Bunke, H. 2013. A novel software toolkit for graph edit distance computation. In *GbRPR*, 142–151. Springer.

Rubner, Y.; Tomasi, C.; and Guibas, L. J. 2000. The earth mover's distance as a metric for image retrieval. *International journal of computer vision* 40(2):99–121.

Spearman, C. 1904. The proof and measurement of association between two things. *The American journal of psychology* 15(1):72–101.

Thévenaz, P.; Blu, T.; and Unser, M. 2000. Image interpolation and resampling. *Handbook of medical imaging, processing and analysis* 1(1):393–420.

Wang, X.; Ding, X.; Tung, A. K.; Ying, S.; and Jin, H. 2012. An efficient graph indexing method. In *ICDE*, 210–221. IEEE.

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

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019. How powerful are graph neural networks? *ICLR*.

Ying, R.; You, J.; Morris, C.; Ren, X.; Hamilton, W. L.; and Leskovec, J. 2018. Hierarchical graph representation learning with differentiable pooling. *NIPS*.

You, J.; Ying, R.; Ren, X.; Hamilton, W.; and Leskovec, J. 2018. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *ICML*, 5694–5703.

Zanfir, A., and Sminchisescu, C. 2018. Deep learning of graph matching. In *CVPR*, 2684–2693.

Zeng, Z.; Tung, A. K.; Wang, J.; Feng, J.; and Zhou, L. 2009. Comparing stars: On approximating graph edit distance. *PVLDB* 2(1):25–36.