

Graph Edit Distance Computation via Graph Neural Networks

Yunsheng Bai

University of California, Los Angeles
yba@ucla.edu

Hao Ding

Purdue University
ding209@purdue.edu

Song Bian

Zhejiang University
biansonghz@gmail.com

Ting Chen

University of California, Los Angeles
tingchen@cs.ucla.edu

Yizhou Sun

University of California, Los Angeles
yzsun@cs.ucla.edu

Wei Wang

University of California, Los Angeles
weiwang@cs.ucla.edu

ABSTRACT

Graph similarity search is among the most important graph-based applications, e.g. finding the chemical compounds that are most similar to a query compound. Graph similarity/distance computation, such as Graph Edit Distance (GED) and Maximum Common Subgraph (MCS), is the core operation of graph similarity search and many other applications, but very costly to compute in practice. Inspired by the recent success of neural network approaches to several graph applications, such as node or graph classification, we propose a novel neural network based approach to address this classic yet challenging graph problem, aiming to alleviate the computational burden while preserving a good performance.

The proposed approach, called SimGNN, combines two strategies. First, we design a learnable embedding function that maps every graph into an embedding vector, which provides a global summary of a graph. A novel attention mechanism is proposed to emphasize the important nodes with respect to a specific similarity metric. Second, we design a pairwise node comparison method to supplement the graph-level embeddings with fine-grained node-level information. Our model can be trained in an end-to-end fashion, achieves better generalization on unseen graphs, and in the worst case runs in quadratic time with respect to the number of nodes in two graphs. Taking GED computation as an example, experimental results on three real graph datasets demonstrate the effectiveness and efficiency of our approach. Specifically, our model achieves smaller error rate and great time reduction compared against a series of baselines, including several approximation algorithms on GED computation, and many existing graph neural network based models. Our study suggests SimGNN provides a new direction for future research on graph similarity computation and graph similarity search.

KEYWORDS

network embedding, neural networks, graph similarity computation, graph edit distance

1 INTRODUCTION

Graphs are ubiquitous nowadays and have a wide range of applications in bioinformatics, chemistry, recommender systems, social network study, program static analysis, etc. Among these, one of the fundamental problems is to retrieve a set of similar graphs from a database given a user query. Different graph similarity/distance metrics are defined, such as Graph Edit Distance (GED) [4], Maximum Common Subgraph (MCS) [6], etc. However, the core operation, namely computing the GED or MCS between two graphs, is known to be NP-complete [6, 55]. For GED, even the state-of-the-art algorithms cannot reliably compute the exact GED within reasonable time between graphs with more than 16 nodes [1].

Given the huge importance yet great difficulty of computing the exact graph distances, there have been two broad categories of methods to address the problem of graph similarity search. The first category of remedies is the pruning-verification framework [31, 55, 56], under which the total amount of exact graph similarity computations for a query can be reduced to a tractable degree, through a series of database indexing techniques and pruning strategies. However, the fundamental problem of the exponential time complexity of exact graph similarity computation [33] remains. The second category tries to reduce the cost of graph similarity computation directly. Instead of calculating the exact similarity metric, these methods find approximate values in a fast and heuristic way [2, 8, 11, 33, 41]. However, these methods usually require rather complicated design and implementation based on discrete optimization or combinatorial search. The time complexity is usually still polynomial or even sub-exponential in the number of nodes in the graphs, such as A*-Beamsearch (Beam) [33], Hungarian [41], VJ [11], etc.

In this paper, we propose a novel approach to speed-up the graph similarity computation, with the same purpose as the second category of methods mentioned previously. However, instead of directly computing the approximate similarities using combinatorial search, our solution turns it into a *learning* problem. More specifically, we design a neural network-based function that maps a pair of graphs into a similarity score. At the training stage, the parameters involved in this function will be learned by minimizing the difference between the predicted similarity scores and the ground truth, where each training data point is a pair of graphs together with their true similarity score. At the test stage, by feeding the learned function with any pair of graphs, we can obtain a predicted similarity score. We name such approach as **SimGNN**, i.e., Similarity Computation via Graph Neural Networks.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference'17, Washington, DC, USA

© 2016 ACM. 978-x-xxxx-xxxx-x/YY/MM...\$15.00

DOI: 10.1145/nnnnnnn.nnnnnnn

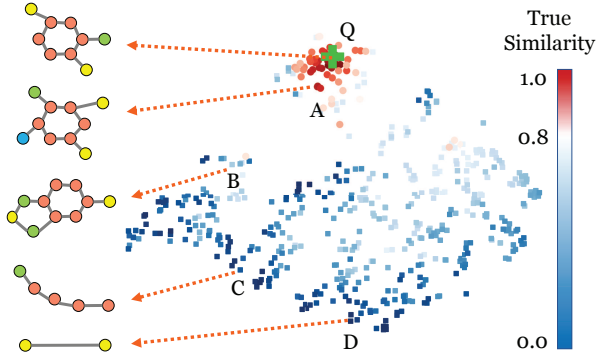


Figure 1: Illustration of similarity-preserving graph embedding. Each graph is mapped into an embedding vector (denoted as a dot in the plot), which preserves their similarity between each other in terms of a specific graph similarity metric. The green “+” sign denotes the embedding of an example query graph. Colors of dots indicate how similar a graph is to the query based on the ground truth (from red to blue, meaning from the most similar to the least similar).

SimGNN enjoys the key advantage of efficiency due to the nature of neural network computation. As for effectiveness, however, we need to carefully design the neural network architecture to satisfy the following three properties:

- (1) **Representation-invariant.** The same graph can be represented by different adjacency matrices by permuting the order of nodes. The computed similarity score should be invariant to such changes.
- (2) **Inductive.** The similarity computation should generalize to unseen graphs, i.e. compute the similarity score for graphs outside the training graph pairs.
- (3) **Learnable.** The model should be adaptive to any similarity metric, by adjusting its parameters through training.

To achieve these goals, we propose the following two strategies. First, we design a learnable embedding function that maps every graph into an embedding vector, which provides a global summary of a graph through aggregating node-level embeddings. We propose a novel attention mechanism to select the important nodes out of an entire graph with respect to a specific similarity metric. This graph-level embedding can already largely preserve the similarity between graphs. For example, as illustrated in Fig. 1, Graph A is very similar to Graph Q according to the ground truth similarity, which is reflected by the embedding as its embedding is close to Q in the embedding space. Also, such embedding-based similarity computation is very fast. Second, we design a pairwise node comparison method to supplement the graph-level embeddings with fine-grained node-level information. As one fixed-length embedding per graph may be too coarse, we further compute the pairwise similarity scores between nodes from the two graphs, from which the histogram features are extracted and combined with the graph-level information to boost the performance of our model. This results in the quadratic amount of operations in terms of graph size, which, however, is still among the most efficient methods for graph similarity computation.

We conduct our experiments on GED computation, which is one of the most popular graph similarity/distance metrics. To demonstrate the effectiveness and efficiency of our approach, we conduct experiments on three real graph datasets. Compared with the baselines, which include several approximate GED computation algorithms, and many graph neural network based methods, our model achieves smaller error and great time reduction. It is worth mentioning that, our Strategy 1 already demonstrates superb performances compared with existing solutions. When running time is a major concern, we can drop the more time-consuming Strategy 2 for trade-off.

Our contributions can be summarized as follows:

- We address the challenging while classic problem of graph similarity computation by considering it as a learning problem, and propose a neural network based approach, called SimGNN, as the solution.
- Two novel strategies are proposed. First, we propose an efficient and effective attention mechanism to select the most relevant parts of a graph to generate a graph-level embedding, which preserves the similarity between graphs. Second, we propose a pairwise node comparison method to supplement the graph-level embeddings for more effective modeling of the similarity between two graphs.
- We conduct extensive experiments on a very popular graph similarity/distance metric, GED, based on three real network datasets to demonstrate the effectiveness and efficiency of the proposed approach.

The rest of this paper is organized as follows. We introduce the preliminaries of our work in Section 2, describe our model in Section 3, present experimental results in Section 4, discuss related work in Section 5, and point out future directions in Section 6. A conclusion is provided in Section 7.

2 PRELIMINARIES

2.1 Graph Edit Distance (GED)

In order to demonstrate the effectiveness and efficiency of SimGNN, we choose one of the most popular graph similarity/distance metric, Graph Edit Distance (GED), as a case study. GED has been widely used in many applications, such as graph similarity search [31, 51, 55, 56, 58], graph classification [40, 41], handwriting recognition [12], image indexing [52], etc.

Formally, the edit distance between \mathcal{G}_1 and \mathcal{G}_2 , denoted by $\text{GED}(\mathcal{G}_1, \mathcal{G}_2)$, 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 vertex/edge or relabelling of a vertex¹. Intuitively, if two graphs are identical (isomorphic), their GED is 0. Fig. 2 shows an example of GED between two simple graphs.

Once the distance between two graphs is calculated, we transform it to a similarity score ranging between 0 and 1. More details about the transformation function can be found in Section 4.2.

2.2 Graph Convolutional Networks (GCN)

Both strategies in SimGNN require node embedding computation. In Strategy 1, to compute graph-level embedding, it aggregates

¹Although other variants of GED exist [42], we adopt this basic version.

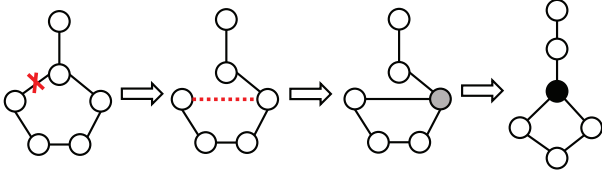


Figure 2: The GED between the graph to the left and the graph to the right is 3, as the transformation needs 3 edit operations: (1) an edge deletion, (2) an edge insertion, and (3) a node relabeling.

node-level embeddings using attention; and in Strategy 2, pairwise node comparison for two graphs is computed based on node-level embeddings as well.

Among many existing node embedding algorithms, we choose to use Graph Convolutional Networks (GCN) [26], as it is graph *representation-invariant*, as long as the initialization is carefully designed. It is also *inductive*, since for any unseen graph, we can always compute the node embedding following the GCN operation. GCN now is among the most popular models for node embeddings, and belong to the family of neighbor aggregation based methods. Its core operation, graph convolution, operates on the representation of a node, which is denoted as $\mathbf{u}_n \in \mathbb{R}^D$, and is defined as follows:

$$\text{conv}(\mathbf{u}_n) = f_1\left(\sum_{m \in N(n)} \frac{1}{\sqrt{d_n d_m}} \mathbf{u}_m \mathbf{W}_1^{(l)} + \mathbf{b}_1^{(l)}\right) \quad (1)$$

$N(n)$ is the set of the first-order neighbors of node n plus n itself, d_n is the degree of node n plus 1, $\mathbf{W}_1^{(l)} \in \mathbb{R}^{D^l \times D^{l+1}}$ is the weight matrix associated with the l -th GCN layer, $\mathbf{b}_1^{(l)} \in \mathbb{R}^{D^{l+1}}$ is the bias, and $f_1(\cdot)$ is an activation function such as $\text{ReLU}(x) = \max(0, x)$. Intuitively, the graph convolution operation aggregates the features from the first-order neighbors of the node.

3 THE PROPOSED APPROACH: SIMGNN

Now we introduce our proposed approach SimGNN in detail, which is an end-to-end neural network based approach that attempts to learn a function to map a pair of graphs into a similarity score. An overview of SimGNN is illustrated in Fig. 3. First, our model transforms the node of each graph into a vector, encoding the features and structural properties around each node. Then, two strategies are proposed to model the similarity between the two graphs, one based on the interaction between two graph-level embeddings, the other based on comparing two sets of node-level embeddings. Finally, two strategies are combined together to feed into a fully connected neural network to get the final similarity score. The rest of the section details these two strategies.

3.1 Strategy One: Graph-Level Embedding Interaction

This strategy is based on the assumption that a good graph-level embedding can encode the structural and feature information of a graph, and by interacting the two graph-level embeddings, the similarity between two graphs can be predicted. It involves the following stages. (1) The node embedding stage, which transforms

each node of a graph into a vector, encoding its features and structural properties. (2) The graph embedding stage, which produces one embedding for each graph, by aggregating node embeddings generated in the previous stage. We propose an effective attention mechanism to allow the model to focus on the most relevant parts of the graph. (3) The graph-graph interaction stage, which receives two graph-level embeddings, and returns the interaction scores representing the graph-graph similarity. (4) The final stage, graph similarity score computation, which further reduces the interaction scores into one final similarity score. It will be compared against the ground-truth similarity score to update parameters involved in the 4 stages.

3.1.1 Stage I: Node Embedding. Among the existing state-of-the-art approaches, we adopt GCN, a neighbor aggregation based method, because it learns an aggregation function (Eq. 1) that are representation-invariant and can be applied to unseen nodes. In Fig. 3, different colors represent different node types, and the original node representations are one-hot encoded. Notice that the one-hot encoding is based on node types (e.g., all the nodes with carbon type share the same one-hot encoding vector), so even if the node ids are permuted, the aggregation results would be the same. For graphs with unlabeled nodes, we treat every node to have the same label, resulting in the same constant number as the initialize representation. After multiple layers of GCNs (e.g., 3 layers in our experiment), the node embeddings are ready to be fed into the Attention module (Att), which is described as follows.

3.1.2 Stage II: Graph Embedding: Global Context-Aware Attention. To generate one embedding per graph using a set of node embeddings, one could perform an unweighted average of node embeddings, or a weighted sum where a weight associated with a node is determined by its degree. However, which nodes are more important and should receive more weights is dependent on the specific similarity metric. Thus, we propose the following attention mechanism to let the model learn weights guided by the specific similarity metric.

Denote the input node embeddings as $\mathbf{U} \in \mathbb{R}^{N \times D}$, where the n -th row, $\mathbf{u}_n \in \mathbb{R}^D$ is the embedding of node n . First, a global graph context $\mathbf{c} \in \mathbb{R}^D$ is computed, which is a simple average of node embeddings followed by a nonlinear transformation: $\mathbf{c} = \tanh\left(\left(\frac{1}{N} \sum_{n=1}^N \mathbf{u}_n\right) \mathbf{W}_2\right)$, where $\mathbf{W}_2 \in \mathbb{R}^{D \times D}$ is a learnable weight matrix. The context \mathbf{c} provides the global structural and feature information of the graph that is adaptive to the given similarity metric, via learning the weight matrix. Based on \mathbf{c} , we can compute one attention weight for each node.

For node n , to make its attention a_n aware of the global context, we take the inner product between \mathbf{c} and its node embedding. The intuition is that, nodes similar to the global context should receive higher attention weights. A sigmoid function $\sigma(x) = \frac{1}{1+\exp(-x)}$ is applied to the result to ensure the attention weights is in the range $(0, 1)$. We do not normalize the weights into length 1, since it is desirable to let the embedding norm reflect the graph size, which is essential for the task of graph similarity computation. Finally, the graph embedding $\mathbf{h} \in \mathbb{R}^D$ is the weighted sum of node embeddings, $\mathbf{h} = \sum_{n=1}^N a_n \mathbf{u}_n$. The following equation summarizes the proposed

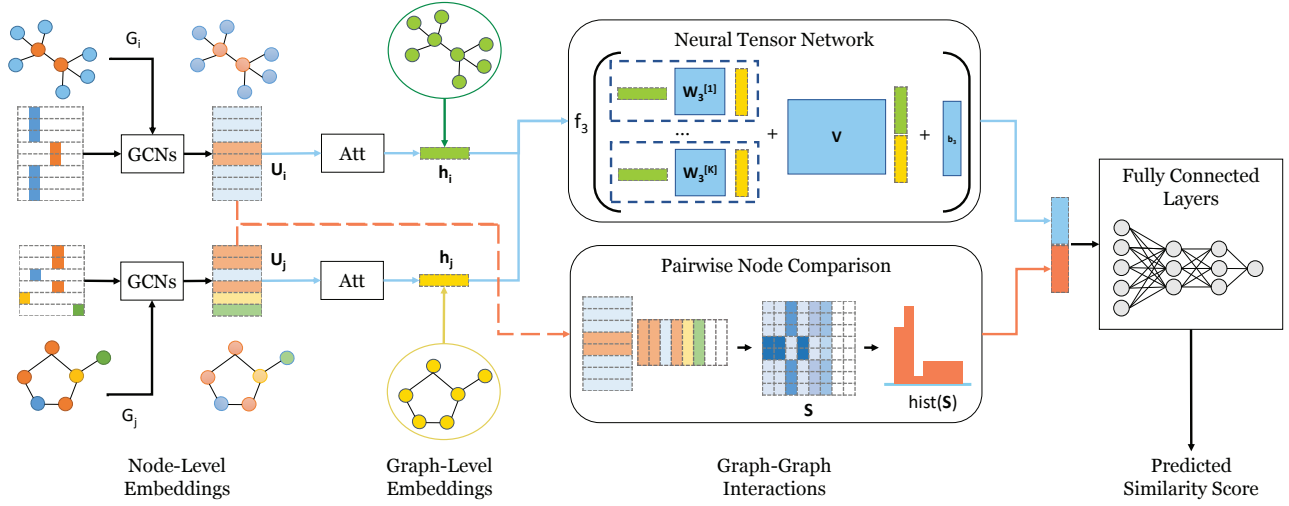


Figure 3: An overview illustration of SimGNN. The blue arrows denote the data flow for Strategy 1, which is based on graph-level embeddings. The red arrows denote the data flow for Strategy 2, which is based on pairwise node comparison.

node attentive mechanism:

$$\mathbf{h} = \sum_{n=1}^N f_2(\mathbf{u}_n^T \mathbf{c}) \mathbf{u}_n = \sum_{n=1}^N f_2(\mathbf{u}_n^T \tanh((\frac{1}{N} \sum_{m=1}^N \mathbf{u}_m) \mathbf{W}_2)) \mathbf{u}_n \quad (2)$$

where $f_2(\cdot)$ is the sigmoid function $\sigma(\cdot)$.

3.1.3 Stage III: Graph-Graph Interaction: Neural Tensor Network.

Given the graph-level embeddings of two graphs produced by the previous stage, a simple way to model their relation is to take the inner product of the two, $\mathbf{h}_i \in \mathbb{R}^D$, $\mathbf{h}_j \in \mathbb{R}^D$. However, as discussed in [45], such simple usage of data representations often lead to insufficient or weak interaction between the two. Following [45], we use Neural Tensor Networks (NTN) to model the relation between two graph-level embeddings:

$$g(\mathbf{h}_i, \mathbf{h}_j) = f_3(\mathbf{h}_i^T \mathbf{W}_3^{[1:K]} \mathbf{h}_j + \mathbf{V} \begin{bmatrix} \mathbf{h}_i \\ \mathbf{h}_j \end{bmatrix} + \mathbf{b}_3) \quad (3)$$

where $\mathbf{W}_3^{[1:K]} \in \mathbb{R}^{D \times D \times K}$ is a weight tensor, $[]$ denotes the concatenation operation, $\mathbf{V} \in \mathbb{R}^{K \times 2D}$ is a weight vector, $\mathbf{b}_3 \in \mathbb{R}^K$ is a bias vector, and $f_3(\cdot)$ is an activation function. K is a hyperparameter controlling the number of interaction (similarity) scores produced by the model for each graph embedding pair.

3.1.4 Stage IV: Graph Similarity Score Computation.

After obtaining a list of similarity scores, we apply a standard multi-layer fully connected neural network to gradually reduce the dimension of the similarity score vector. In the end, one score, $\hat{s}_{ij} \in \mathbb{R}$, is predicted, and it is compared against the ground-truth similarity score using the following mean squared error loss function:

$$\mathcal{L} = \frac{1}{|\mathcal{D}|} \sum_{(i,j) \in \mathcal{D}} (\hat{s}_{ij} - s(\mathcal{G}_i, \mathcal{G}_j))^2 \quad (4)$$

where \mathcal{D} is the set of training graph pairs, and $s(\mathcal{G}_i, \mathcal{G}_j)$ is the ground-truth similarity between \mathcal{G}_i and \mathcal{G}_j .

3.1.5 Limitations of Strategy One. As mentioned in Section 1, the node-level information such as the node feature distribution and graph size may be lost by the graph-level embedding. In many

cases, the differences between two graphs lie in small substructures and are hard to be reflected by the graph level embedding. An analogy is that, in Natural Language Processing, the performance of sentence matching based on one embedding per sentence can be further enhanced through using fine-grained word-level information [19, 21]. This leads to our second strategy.

3.2 Strategy Two: Pairwise Node Comparison

To overcome the limitations mentioned previously, we consider bypassing the NTN module, and using the node-level embeddings directly. As illustrated in the bottom data flow of Fig. 3, if \mathcal{G}_i has N_i nodes and \mathcal{G}_j has N_j nodes, there would be $N_i N_j$ pairwise interaction scores, obtained by $\mathbf{S} = \sigma(\mathbf{U}_i \mathbf{U}_j^T)$, where $\mathbf{U}_i \in \mathbb{R}^{N_i \times D}$ and $\mathbf{U}_j \in \mathbb{R}^{N_j \times D}$ are the node embeddings of \mathcal{G}_i and \mathcal{G}_j , respectively. Since the node-level embeddings are not normalized, the sigmoid function is applied to ensure the similarities scores are in the range of (0, 1). For two graphs of different sizes, to emphasize their size difference, we pad fake nodes to the smaller graph. As shown in Fig. 3, two fake nodes with zero embedding are padded to the bottom graph, resulting in two extra columns with zeros in \mathbf{S} .

Denote $N = \max(N_1, N_2)$. The pairwise node similarity matrix $\mathbf{S} \in \mathbb{R}^{N \times N}$ is a useful source of information, since it encodes fine-grained pairwise node similarity scores. One simple way to utilize \mathbf{S} is to vectorize it: $\text{vec}(\mathbf{S}) \in \mathbb{R}^{N^2}$, and feed it into the fully connected layers. However, there is usually no natural ordering between graph nodes. Different initial node ordering of the same graph would lead to different \mathbf{S} and $\text{vec}(\mathbf{S})$.

To ensure the model is invariant to the graph representations as mentioned in Section 1, one could preprocess the graph by applying some node ordering scheme [34], but we consider a much more efficient and natural way to utilize \mathbf{S} . We extract its histogram features: $\text{hist}(\mathbf{S}) \in \mathbb{R}^B$, where B is a hyperparameter that controls the number of bins in the histogram. In the case of Fig. 3, seven bins

are used for the histogram. The histogram feature vector is normalized and concatenated with the graph-level interaction scores $g(\mathbf{h}_i, \mathbf{h}_j)$, and fed to the fully connected layers to obtain a final similarity score for the graph pair.

It is important to note that the histogram features alone are not enough to train the model, since the histogram is not a continuous differential function and does not support backpropagation. In fact, we rely on Strategy 1 as the primary strategy to update the model weights, and use Strategy 2 to supplement the graph-level features, which brings extra performance gain to our model.

To sum up, we combine the coarse global comparison information captured by Strategy 1, and the fine-grained node-level comparison information captured by Strategy 2, to provide a thorough view of the graph comparison to the model.

3.3 Time Complexity Analysis

Once SimGNN has been trained, it can be used to compute a similarity score for any pair of graphs. The time complexity involves two parts: (1) the node-level and global-level embedding computation stages, which needs to be computed once for each graph; and (2) the similarity score computation stage, which needs to be computed for every pair of graphs.

The node-level and global-level embedding computation stages.

The time complexity associated with the generation of node-level and graph-level embeddings is $O(E)$ [26], where E is the number of edges of the graph. Notice that the graph-level embeddings can be pre-computed and stored, and in the setting of graph similarity search, the unseen query graph only needs to be processed once to obtain its graph-level embedding.

The similarity score computation stage. The time complexity for Strategy 1 is $O(D^2K)$, where D is the dimension of the graph level embedding, and K is the feature map dimension of the NTN. The time complexity for our Strategy 2 is $O(D^2N^2)$, where N is the number of nodes in the larger graph. This can potentially be reduced by node sampling to construct the similarity matrix S . Moreover, the matrix multiplication $S = \sigma(U_1 U_2^T)$ can be greatly accelerated with GPUs. Our experimental results in Section 4.6.2 verify that there is no significant runtime increase when the second strategy is used.

In conclusion, among the two strategies we have proposed: Strategy 1 is the primary strategy, which is efficient but solely based on coarse graph level embeddings; and Strategy 2 is auxiliary, which includes fine-grained node-level information but is more time-consuming. In the worst case, the model runs in quadratic time with respect to the number of nodes, which is among the state-of-the-art algorithms for approximate graph distance computation.

4 EXPERIMENTS

4.1 Datasets

Three real-world graph datasets are used for the experiments. A concise summary and detailed visualizations can be found in Table 1 and Fig. 4, respectively.

AIDS. AIDS is a collection of antivirus screen chemical compounds from the Developmental Therapeutics Program at NCI/NIH

Table 1: Statistics of datasets.

Dataset	Graph Meaning	#Graphs	#Pairs
AIDS	Chemical Compounds	700	490K
LINUX	Program Dependency Graphs	1000	1M
IMDB	Actor/Actress Ego-Networks	1500	2.25M

7², and has been used in several existing works on graph similarity search [31, 51, 55, 56, 58]. It contains 42,687 chemical compound structures with Hydrogen atoms omitted. We select 700 graphs, each of which has 10 or less than 10 nodes. Each node is labeled with one of 29 types, as illustrated in Fig. 4a.

LINUX. The LINUX dataset was originally introduced in [51]. It consists of 48,747 Program Dependence Graphs (PDG) generated from the Linux kernel. Each graph represents a function, where a node represents one statement and an edge represents the dependency between the two statements. We randomly select 1000 graphs of equal or less than 10 nodes each. The nodes are unlabeled.

IMDB. The IMDB dataset [53] (named “IMDB-MULTI”) consists of 1500 ego-networks of movie actors/actresses, where there is an edge if the two people appear in the same movie. To test the scalability and efficiency of our proposed approach, we use the full dataset without any selection. The nodes are unlabeled.

4.2 Data Preprocessing

For each dataset, we randomly split 60%, 20%, and 20% of all the graphs as training set, validation set, and testing set, respectively. The evaluation reflects the real-world scenario of graph query: For each graph in the testing set, we treat it as a query graph, and let the model compute the similarity between the query graph and every graph in the database. The database graphs are ranked according to the computed similarities to the query.

Since graphs from AIDS and LINUX are relatively small, an exponential-time exact GED computation algorithm named A^* [42] is used to compute the GEDs between all the graph pairs. For the IMDB dataset, however, A^* can no longer be used, as a recent survey of exact GED computation [1] concludes, “no currently available algorithm manages to reliably compute GED within reasonable time between graphs with more than 16 nodes.”

To properly handle the IMDB dataset, we take the smallest distance computed by three well-known approximate algorithms, Beam [33], Hungarian [28, 41], and VJ [11, 22]. The minimum is taken instead of the average, because their returned GEDs are guaranteed to be greater than or equal to the true GEDs. Details on these algorithms can be found in Section 4.3. Incidentally, the ICPR 2016 Graph Distance Contest³ also adopts this approach to obtaining ground-truth GEDs for large graphs.

To transform ground-truth GEDs into ground-truth similarity scores to train our model, we first normalize the GEDs according to [39]: $nGED(\mathcal{G}_1, \mathcal{G}_2) = \frac{GED(\mathcal{G}_1, \mathcal{G}_2)}{(|\mathcal{G}_1| + |\mathcal{G}_2|)/2}$, where $|\mathcal{G}_i|$ denotes the number of nodes of \mathcal{G}_i . We then adopt the exponential function $\lambda(x) = e^{-x}$ to transform the normalized GED into a similarity score in the range of (0, 1]. Notice that there is a one-to-one mapping between the GED and the similarity score.

²<https://wiki.nci.nih.gov/display/NCIDTPdata/AIDS+Antiviral+Screen+Data>

³<https://gdc2016.greyc.fr/>

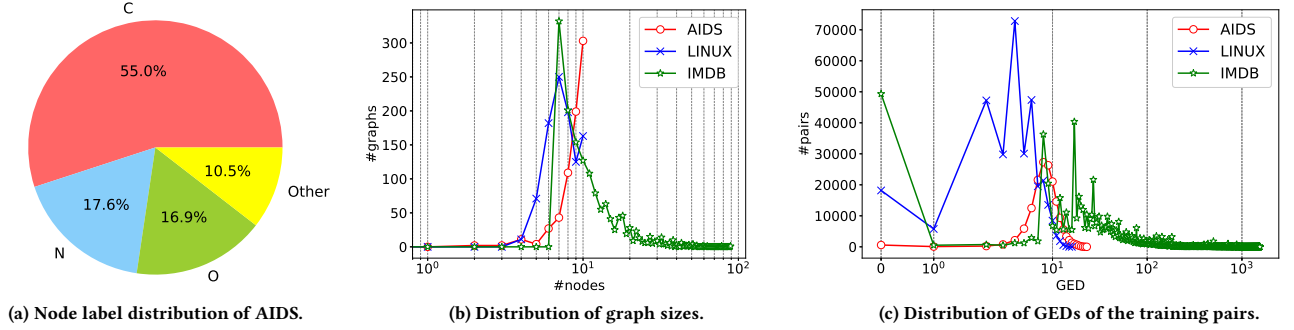


Figure 4: Some statistics of the datasets.

4.3 Baseline Methods

Our baselines include two types of approaches, fast approximate GED computation algorithms and neural network based models.

The first category of baselines includes three classic algorithms for GED computation. (1) *A*-Beamsearch (Beam)* [33]. It is a variant of the A* algorithm in sub-exponential time. (2) *Hungarian* [28, 41] and (3) *VJ* [11, 22] are two cubic-time algorithms based on the Hungarian Algorithm for bipartite graph matching, and the algorithm of Volgenant and Jonker, respectively.

The second category of baselines includes seven models of the following neural network architectures. (1) *SimpleMean* simply takes the unweighted average of all the node embeddings of a graph to generate its graph-level embedding. (2) *HierarchicalMean* and (3) *HierarchicalMax* [9] are the original GCN architectures based on graph coarsening, which use the global mean or max pooling to generate a graph hierarchy. We use the implementation from the Github repository of the first author of GCN⁴. The next four models apply the attention mechanism on nodes. (4) *AttDegree* uses the natural log of the degree of a node as its attention weight, as described in Section 3.1.2. (5) *AttGlobalContext* and (6) *AttLearnableGlobalContext (AttLearnableGC)* both utilize the global graph context to compute the attention weights, but the former does not apply the nonlinear transformation with learnable weights on the context, while the latter does. (7) *SimGNN* is our full model that combines the best of Strategy 1 (AttLearnableGC) and Strategy 2 as described in Section 3.2.

4.4 Parameter Settings

For the model architecture, we set the number of GCN layers to 3, and use ReLU as the activation function. For the initial node representations, we adopt the one-hot encoding scheme for AIDS reflecting the node type, and the constant encoding scheme for LINUX and IMDB, since their nodes are unlabeled, as mentioned in Section 3.1.1. The output dimensions for the 1st, 2nd, and 3rd layer of GCN are 64, 32, and 16, respectively. For the NTN layer, we set K to 16. For the pairwise node comparison strategy, we set the number of histogram bins to 16. We use 4 fully connected layers to reduce the dimension of the concatenated results from the NTN module, from 32 to 16, 16 to 8, 8 to 4, and 4 to 1.

⁴https://github.com/mdeff/cnn_graph

We conduct all the experiments on a single machine with an Intel i7-6800K CPU and one Nvidia Titan GPU. As for training, we set the batch size to 128, use the Adam algorithm for optimization [25], and fix the initial learning rate to 0.001. We set the number of iterations to 10000, and select the best model based on the lowest validation loss.

4.5 Evaluation Metrics

The following metrics are used to evaluate all the models: *Time*. The wall time needed for each model to compute the similarity score for a pair of graphs is collected. *Mean Squared Error (mse)*. The mean squared error measures the average squared difference between the computed similarities and the ground-truth similarities.

We also adopt the following metrics to evaluate the ranking results. *Spearman's Rank Correlation Coefficient (ρ)* [46] and *Kendall's Rank Correlation Coefficient (τ)* [24] measure how well the predicted ranking results match the true ranking results. *Precision at k ($p@k$)*. $p@k$ is computed by taking the intersection of the predicted top k results and the ground-truth top k results divided by k . Compared with $p@k$, ρ and τ evaluates the global ranking results instead of focusing on the top k results.

4.6 Results

4.6.1 Effectiveness. The effectiveness results on the three datasets can be found in Table 2, 3, and 4. Our model, SimGNN, consistently achieves the best or the second best performance on all metrics across the three datasets. Within the neural network based methods, SimGNN consistently achieves the best results on all metrics. This suggests that our model can learn a good embedding function that generalizes to unseen test graphs.

Beam achieves the best precisions at 10 on AIDS and LINUX. We conjecture that it can be attributed to the imbalanced ground-truth GED distributions. As seen in Fig. 4c, for AIDS, the training pairs have GEDs mostly around 10, causing our model to train the very similar pairs less frequently than the dissimilar ones. For LINUX, the situation for SimGNN is better, since most GEDs concentrate in the range of $[0, 10]$, the gap between the precisions at 10 of Beam and SimGNN become smaller.

It is noteworthy that among the neural network based models, AttDegree achieves relatively good results on IMDB, but not on

Table 2: Results on AIDS.

Method	mse(10^{-3})	ρ	τ	p@10	p@20
Beam	12.090	0.609	0.463	0.481	0.493
Hungarian	25.296	0.510	0.378	0.360	0.392
VJ	29.157	0.517	0.383	0.310	0.345
SimpleMean	3.115	0.633	0.480	0.269	0.279
HierarchicalMean	3.046	0.681	0.629	0.246	0.340
HierarchicalMax	3.396	0.655	0.505	0.222	0.295
AttDegree	3.338	0.628	0.478	0.209	0.279
AttGlobalContext	1.472	0.813	0.653	0.376	0.473
AttLearnableGC	1.340	0.825	0.667	0.400	0.488
SimGNN	1.189	0.843	0.690	0.421	0.514

Table 3: Results on LINUX.

Method	mse(10^{-3})	ρ	τ	p@10	p@20
Beam	9.268	0.827	0.714	0.973	0.924
Hungarian	29.805	0.638	0.517	0.913	0.836
VJ	63.863	0.581	0.450	0.287	0.251
SimpleMean	16.950	0.020	0.016	0.432	0.465
HierarchicalMean	6.431	0.430	0.525	0.750	0.618
HierarchicalMax	6.575	0.879	0.740	0.551	0.575
AttDegree	8.064	0.742	0.609	0.427	0.460
AttGlobalContext	3.125	0.904	0.781	0.874	0.864
AttLearnableGC	2.055	0.916	0.804	0.903	0.887
SimGNN	1.509	0.939	0.830	0.942	0.933

Table 4: Results on IMDB. Beam, Hungarian, and VJ together are used to determine the ground-truth results.

Method	mse(10^{-3})	ρ	τ	p@10	p@20
SimpleMean	3.749	0.774	0.644	0.547	0.588
HierarchicalMean	5.019	0.456	0.378	0.567	0.553
HierarchicalMax	6.993	0.455	0.354	0.572	0.570
AttDegree	2.144	0.828	0.695	0.700	0.695
AttGlobalContext	3.555	0.684	0.553	0.657	0.656
AttLearnableGC	1.455	0.835	0.700	0.732	0.742
SimGNN	1.264	0.878	0.770	0.759	0.777

AIDS or LINUX. It could be due to the unique ego-network structures commonly present in IMDB. As seen in Fig. 10, the high-degree central node denotes the particular actor/actress himself/herself, focusing on which could be a reasonable heuristic. In contrast, AttLearnableGC adapts to the GED metric via a learnable global context, and consistently performs better than AttDegree. Combined with Strategy 2, SimGNN achieves even better performances.

Visualizations of the node attentions can be seen in Fig. 5. We observe that the following kinds of nodes receive relatively higher attention weights: hub nodes with large degrees, e.g. the “S” in (a) and (b), nodes with labels that rarely occur in the dataset, e.g. the “Ni” in (f), the “Pd” in (g), the “Br” in (h), nodes forming special substructures, e.g. the two middle “C”s in (e), etc. These patterns make intuitive sense, further confirming the effectiveness of the proposed approach.

4.6.2 Efficiency. The efficiency comparison on the three datasets is shown in Fig. 6. The neural network based models consistently achieve the best results across all the three datasets. Specifically, compared with the exact algorithm, A*, SimGNN is 2174 times faster on AIDS, and 212 times faster on LINUX. The A* algorithm cannot even be applied on large graphs, and in the case of IMDB, its variant, Beam, is still 46 times slower than SimGNN. Moreover,

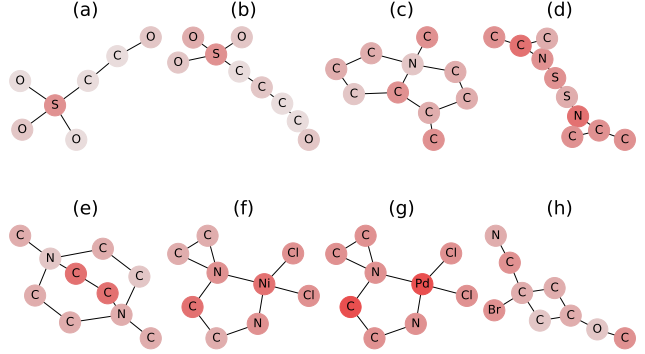


Figure 5: Visualizations of node attentions. The darker the color, the larger the attention weight.

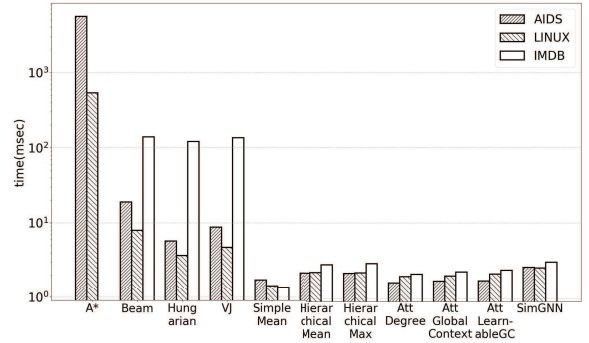


Figure 6: Runtime comparison.

the time measured for SimGNN includes the time for graph embedding. As mentioned in Section 3.3, if graph embeddings are pre-computed and stored, SimGNN would spend even less time. All of these suggest that in practice, it is reasonable to use SimGNN as a fast approach to graph similarity computation, which is especially true for large graphs, as in IMDB, our computation time does not increase much compared with AIDS and LINUX.

4.7 Parameter Sensitivity

We evaluate how the dimension of the graph-level embeddings and the number of the histogram bins can affect the results. We report the mean squared error on AIDS. As can be seen in Fig. 7a, the performance becomes better if larger dimensions are used. This makes intuitive sense, since larger embedding dimensions give the model more capacity to represent graphs. In our Strategy 2, as shown in Fig. 7b, the performance is relatively insensitive to the number of histogram bins. This suggests that in practice, as long as the histogram bins are not too few, relatively good performance can be achieved.

4.8 Case Studies

We demonstrate three example queries, one from each dataset, in Fig. 8, 9, and 10. In each demo, the top row depicts the query along

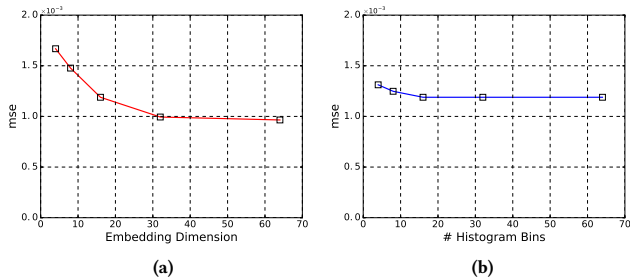


Figure 7: Mean squared error w.r.t. the number of dimensions of graph-level embeddings, and the number of histogram bins.

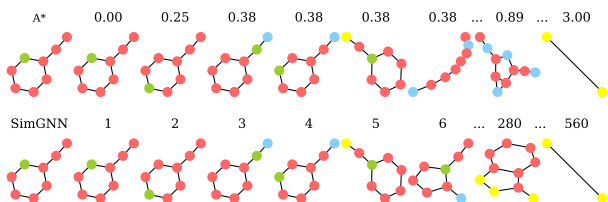


Figure 8: A query case study on AIDS. Meanings of the colors can be found in Fig. 4a.

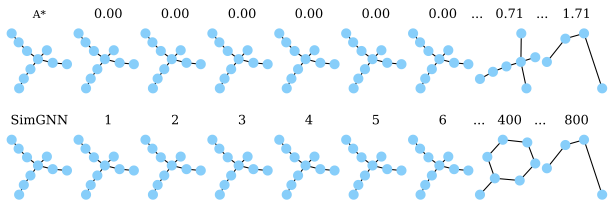


Figure 9: A query case study on LINUX.

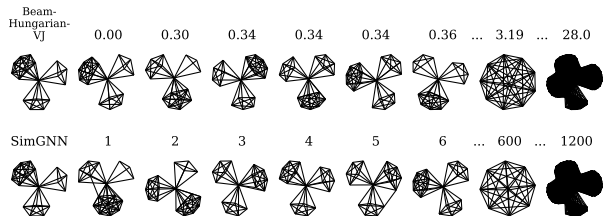


Figure 10: A query case study on IMDB.

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. SimGNN is able to retrieve graphs similar to the query, e.g. in the case of LINUX (Fig. 9), the top 6 results are exactly the isomorphic graphs to the query.

5 RELATED WORK

To precisely position our task, outline the scope of our work, and compare various methods related to our model, we briefly survey the following topics.

5.1 Network/Graph Embedding

Node-level embedding. Over the years, there are several categories of methods that have been proposed for learning node representations, including matrix factorization based methods (NetMF [38]), skip-gram based methods (DeepWalk [37], Node2Vec [15], LINE [47]), autoencoder based methods (SDNE [50]), neighbor aggregation based methods (GCN [9, 26, 27], GraphSAGE [16]), etc.

Graph-level embedding. The most intuitive way to generate one embedding per graph is to aggregate the node-level embeddings, either by a simple average or some weighted average [7, 10, 57], named the “sum-based” approaches [18]. A more sophisticated way to represent graphs can be achieved by viewing a graph as a hierarchical data structure and applying graph coarsening [3, 9, 44, 54]. Besides, [23] aggregate sets of nodes via histograms, and [34] applies node ordering on a graph to make it CNN suitable.

Graph neural network applications. A great amount of graph-based applications have been tackled by neural network based methods, most of which are framed as node-level prediction tasks. However, once moving to the graph-level tasks, most existing work deal with the classification of a single graph [9, 14, 34, 44, 44, 54, 57]. In this work, we consider the task of graph similarity computation for the first time.

5.2 Graph Similarity Computation

Graph distance/similarity metrics. The Graph Edit Distance (GED) [4] can be considered as an extension of the String Edit Distance metric [30], which is defined as the minimum cost taken to transform one graph to the other via a sequence graph edit operations. Another well-known concept is the Maximum Common Subgraph (MCS), which has been shown to be equivalent to GED under a certain cost function [5]. Graph kernels can be considered as a family of different graph similarity metrics, used primarily for graph classification. Numerous graph kernels [13, 20, 36] and extensions [35, 53] have been proposed across the years.

Pairwise GED computation algorithms. A flurry of approximate algorithms has been proposed to reduce the time complexity with the sacrifice in accuracy [2, 8, 11, 33, 41]. We are aware of some very recent work claiming their time complexity is $O(n^2)$ [2], but their code is unstable at this stage for comparison.

Graph Similarity search. Computing GED is a primitive operator in graph database analysis, and has been adopted in a series of works on graph similarity search [31, 51, 55, 56, 58]. It must be noted, however, that these studies focus on database-level techniques to speed up the overall querying process involving exact GED computations, while our model, at the current stage, is more comparable in its flavor to the approximate pairwise GED computation algorithms.

6 DISCUSSIONS AND FUTURE DIRECTIONS

There are several directions to go for the future work: (1) our model can handle graphs with node types but cannot process edge features. In chemistry, bonds of a chemical compound are usually labeled, so it is useful to incorporate edge labels into our model; (2) it is promising to explore different techniques to further boost the precisions at the top k results, which is not preserved well mainly due to the skewed similarity distribution in the training dataset;

and (3) given the constraint that the exact GEDs for large graphs cannot be computed, it would be interesting to see how the learned model generalize to large graphs, which is trained only on the exact GEDs between small graphs.

7 CONCLUSION

We are at the intersection of graph deep learning and graph search problem, and taking the first step towards bridging the gap, by tackling the core operation of graph similarity computation, via a novel neural network based approach. The central idea is to learn a neural network-based function that is representation-invariant, inductive, and adaptive to the specific similarity metric, which takes any two graphs as input and outputs their similarity score. Our model runs very fast compared to existing classic algorithms on approximate Graph Edit Distance computation, and achieves very competitive accuracy.

REFERENCES

- [1] David B Blumenthal and Johann Gamper. 2018. On the exact computation of the graph edit distance. *Pattern Recognition Letters* (2018).
- [2] Sébastien Bogleux, Luc Brun, Vincenzo Carletti, Pasquale Foggia, Benoit Gaüzère, and Mario Vento. 2017. Graph edit distance as a quadratic assignment problem. *Pattern Recognition Letters* 87 (2017), 38–46.
- [3] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral networks and locally connected networks on graphs. *ICLR* (2014).
- [4] H Bunke. 1983. What is the distance between graphs. *Bulletin of the EATCS* 20 (1983), 35–39.
- [5] Horst Bunke. 1997. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters* 18, 8 (1997), 689–694.
- [6] Horst Bunke and Kim Shearer. 1998. A graph distance metric based on the maximal common subgraph. *Pattern recognition letters* 19, 3-4 (1998), 255–259.
- [7] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative embeddings of latent variable models for structured data. In *ICML*. 2702–2711.
- [8] Évariste Daller, Sébastien Bogleux, Benoit Gaüzère, and Luc Brun. 2018. Approximate graph edit distance by several local searches in parallel. In *ICPRAM*.
- [9] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*. 3844–3852.
- [10] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*. 2224–2232.
- [11] Stefan Fankhauser, Kaspar Riesen, and Horst Bunke. 2011. Speeding up graph edit distance computation through fast bipartite matching. In *International Workshop on Graph-Based Representations in Pattern Recognition*. Springer, 102–111.
- [12] Andreas Fischer, Ching Y Suen, Volkmar Frinken, Kaspar Riesen, and Horst Bunke. 2013. A fast matching algorithm for graph-based handwriting recognition. In *International Workshop on Graph-Based Representations in Pattern Recognition*. Springer, 194–203.
- [13] Thomas Gärtner, Peter Flach, and Stefan Wrobel. 2003. On graph kernels: Hardness results and efficient alternatives. In *COLT*. Springer, 129–143.
- [14] Vladimir Gligorijević, Meet Barot, and Richard Bonneau. 2017. deepNF: Deep network fusion for protein function prediction. *bioRxiv* (2017), 223339.
- [15] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*. ACM, 855–864.
- [16] Will Hamilton, Zitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*. 1024–1034.
- [17] William L Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. 2018. Querying Complex Networks in Vector Space. *NIPS* (2018).
- [18] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *Data Engineering Bulletin* (2017).
- [19] Hua He and Jimmy Lin. 2016. Pairwise word interaction modeling with deep neural networks for semantic similarity measurement. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 937–948.
- [20] Tamás Horváth, Thomas Gärtner, and Stefan Wrobel. 2004. Cyclic pattern kernels for predictive graph mining. In *SIGKDD*. ACM, 158–167.
- [21] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014. Convolutional neural network architectures for matching natural language sentences. In *NIPS*. 2042–2050.
- [22] Roy Jonker and Anton Volgenant. 1987. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing* 38, 4 (1987), 325–340.
- [23] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. 2016. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design* 30, 8 (2016), 595–608.
- [24] Maurice G Kendall. 1938. A new measure of rank correlation. *Biometrika* 30, 1/2 (1938), 81–93.
- [25] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *ICLR* (2015).
- [26] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *ICLR* (2016).
- [27] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning* (2016).
- [28] Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly* 2, 1-2 (1955), 83–97.
- [29] John Boaz Lee, Ryan Rossi, and Xiangnan Kong. 2018. Graph Classification using Structural Attention. In *SIGKDD*. ACM, 1666–1674.
- [30] Vladimir I Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, Vol. 10. 707–710.
- [31] Yongjiang Liang and Peixiang Zhao. 2017. Similarity search in graph databases: A multi-layered indexing approach. In *ICDE*. IEEE, 783–794.
- [32] Tengfei Ma, Cao Xiao, Jiayu Zhou, and Fei Wang. 2018. Drug Similarity Integration Through Attentive Multi-view Graph Auto-Encoders. *IJCAI* (2018).
- [33] Michel Neuhaus, Kaspar Riesen, and Horst Bunke. 2006. 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)*. Springer, 163–172.
- [34] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutikov. 2016. Learning convolutional neural networks for graphs. In *ICML*. 2014–2023.
- [35] Giannis Nikolentzos, Polykarpos Meladianos, Stratis Limnios, and Michalis Vazirgiannis. 2018. A Degeneracy Framework for Graph Similarity. In *IJCAI*. 2595–2601.
- [36] Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. 2017. Matching Node Embeddings for Graph Similarity. In *AAAI*. 2429–2435.
- [37] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *SIGKDD*. ACM, 701–710.
- [38] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *WSDM*. ACM, 459–467.
- [39] Rashid Jalal Qureshi, Jean-Yves Ramel, and Hubert Cardot. 2007. Graph based shapes representation and recognition. In *International Workshop on Graph-Based Representations in Pattern Recognition*. Springer, 49–60.
- [40] Kaspar Riesen and Horst Bunke. 2008. IAM graph database repository for graph based pattern recognition and machine learning. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer, 287–297.
- [41] Kaspar Riesen and Horst Bunke. 2009. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision computing* 27, 7 (2009), 950–959.
- [42] Kaspar Riesen, Sandro Emmenegger, and Horst Bunke. 2013. A novel software toolkit for graph edit distance computation. In *International Workshop on Graph-Based Representations in Pattern Recognition*. Springer, 142–151.
- [43] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *ESWC*. Springer, 593–607.
- [44] Martin Simonovsky and Nikos Komodakis. 2017. Dynamic edgeconditioned filters in convolutional neural networks on graphs. In *Proc. CVPR*.
- [45] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. 2013. Reasoning with neural tensor networks for knowledge base completion. In *NIPS*. 926–934.
- [46] Charles Spearman. 1904. The proof and measurement of association between two things. *The American journal of psychology* 15, 1 (1904), 72–101.
- [47] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. International World Wide Web Conferences Steering Committee, 1067–1077.
- [48] Kiran K Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. 2018. Attention-based Graph Neural Network for Semi-supervised Learning. *ICLR* (2018).
- [49] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. *ICLR* (2018).
- [50] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *SIGKDD*. ACM, 1225–1234.
- [51] Xiaoli Wang, Xiaofeng Ding, Anthony KH Tung, Shanshan Ying, and Hai Jin. 2012. An efficient graph indexing method. In *ICDE*. IEEE, 210–221.
- [52] Bing Xiao, Xinbo Gao, Dacheng Tao, and Xuelong Li. 2008. HMM-based graph edit distance for image indexing. *International Journal of Imaging Systems and Technology* 18, 2-3 (2008), 209–218.

- [53] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *SIGKDD*. ACM, 1365–1374.
- [54] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. *arXiv preprint arXiv:1806.08804* (2018).
- [55] Zhiping Zeng, Anthony KH Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. 2009. Comparing stars: On approximating graph edit distance. *PVLDB* 2, 1 (2009), 25–36.
- [56] Xiang Zhao, Chuan Xiao, Xuemin Lin, Qing Liu, and Wenjie Zhang. 2013. A partition-based approach to structure similarity search. *PVLDB* 7, 3 (2013), 169–180.
- [57] Xiaohan Zhao, Bo Zong, Ziyu Guan, Kai Zhang, and Wei Zhao. 2018. Substructure Assembling Network for Graph Classification. *AAAI* (2018).
- [58] Weiguo Zheng, Lei Zou, Xiang Lian, Dong Wang, and Dongyan Zhao. 2013. Graph similarity search with edit distance constraint in large graph databases. In *CIKM*. ACM, 1595–1600.
- [59] Marinka Zitnik, Monica Agrawal, and Jure Leskovec. 2018. Modeling polypharmacy side effects with graph convolutional networks. *ISMB* (2018).