Interpretable Graph Similarity Computation via Differentiable Optimal Alignment of Node Embeddings

Khoa D. Doan[1*], Saurav Manchanda[2*], Suchismit Mahapatra[3], Chandan K. Reddy[1]
[1]Department of Computer Science, Virginia Tech, Arlington, VA, USA
[2]Department of Computer Science, University of Minnesota, Minneapolis, MN, USA
[3]Department of Computer Science, University at Buffalo, NY, USA
khoadoan@vt.edu,manch043@umn.edu,suchismi@buffalo.edu,reddy@cs.vt.edu

ABSTRACT

Computing graph similarity is an important task in many graph-related applications such as retrieval in graph databases or graph clustering. While numerous measures have been proposed to capture the similarity between a pair of graphs, Graph Edit Distance (GED) and Maximum Common Subgraphs (MCS) are the two widely used measures in practice. GED and MCS are domain-agnostic measures of structural similarity between the graphs and define the similarity as a function of pairwise alignment of different entities (such as nodes, edges, and subgraphs) in the two graphs. The explicit explainability offered by the pairwise alignment provides transparency and justification of the similarity score, thus, GED and MCS have important practical applications. However, their exact computations are known to be NP-hard. While recently proposed neural-network based approximations have been shown to accurately compute these similarity scores, they have limited ability in providing comprehensive explanations compared to classical combinatorial algorithms, e.g., Beam search. This paper aims at efficiently approximating these domain-agnostic similarity measures through a neural network, and simultaneously learning the alignments (i.e., explanations) similar to those of classical intractable methods. Specifically, we formulate the similarity between a pair of graphs as the minimal "transformation" cost from one graph to another in the learnable node-embedding space. We show that, if node embedding is able to capture its neighborhood context closely, our proposed similarity function closely approximates both the alignment and the similarity score of classical methods. Furthermore, we also propose an efficient differentiable computation of our proposed objective for model training. Empirically, we demonstrate that the proposed method achieves up to 50%-100% reduction in the Mean Squared Error for the graph similarity approximation task and up to 20% improvement in the retrieval evaluation metrics for the graph retrieval task. The source code is available at https://github.com/khoadoan/GraphOTSim.

SIGIR '21, July 11–15, 2021, Virtual Event, Canada

ACM ISBN 978-1-4503-8037-9/21/07…$15.00
https://doi.org/10.1145/3404835.3462960

CCS CONCEPTS

• **Mathematics of computing → Graph algorithms**; • **Computing methodologies → Neural networks**; • **Information systems → Similarity measures**; *Top-k retrieval in databases.*

KEYWORDS

Graph similarity, GCN, similarity search, model interpretability

ACM Reference Format:
Khoa D. Doan[1] [1], Saurav Manchanda[2] [1], Suchismit Mahapatra[3], Chandan K. Reddy[1]. 2021. Interpretable Graph Similarity Computation via Differentiable Optimal Alignment of Node Embeddings. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '21), July 11–15, 2021, Virtual Event, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3404835.3462960

1 INTRODUCTION

Graphs are non-linear data structures used to model a set of interconnected objects (nodes) and their relationships (edges). Recently, the ubiquitous expressive power of graphs has received immense attention in data modeling across various research areas including social networks, natural sciences (drugs and protein-protein interaction networks), and knowledge graphs. Consequently, a wide range of graph analytical techniques are being developed to learn and extract useful patterns from graph data. One of the challenging problems when dealing with graph databases is to compute the similarity between a pair of graphs, which is critical to several graph applications such as retrieval from graph databases and graph clustering. To tackle this problem, different graph similarity measures have been proposed in the literature. Among them, Graph Edit Distance (GED) and Maximum Common Subgraphs (MCS) are two examples of the most prevalent measures. GED and MCS, which are domain-agnostic measures of structural similarity between the graphs, define the similarity as a function of pairwise alignment of different entities (such as nodes, edges, and subgraphs) in the two graphs. As such, GED and MCS have important applications in many domains such as bioinformatics and cheminformatics [12, 13], because the explicit explainability of the similarity score provides transparency and justification, as well as additional informative inductive knowledge towards decision making in downstream analysis in these domains. For example, in medicinal chemistry in particular, the availability of 'rules of thumb' underscores the willingness, in certain situations, to sacrifice accuracy in favor of models that better fit the human intuition [15]. In addition to being domain-agnostic similarity measures, GED

*Both authors are equal contributors.

Table 1: Characteristics of graph similarity algorithms along with the representative methods. × indicates the method lacks the specific characteristic. ✓ indicates the method achieves the objective. 'N/A' stands for Not Applicable. A desirable approach will have *high accuracy*, *high interpretability*, and *low computation*. Our proposed GOTSim method satisfies all these characteristics by employing an *end-to-end* learning-based approach.

	Bipartite [8, 25]	SimGNN [1]	GraphSim [2]	Graph Matching [6, 19, 21]	OAK [23]	GOTSim (Ours)
Learning-based	×	✓	✓	✓	✓	✓
End-to-End	×	✓	✓	✓	×	✓
Accuracy	Low	High	High	High	Low	High
Interpretability	High	N/A	N/A	Low	Low	High
Computation	High	Low	Low	Low	Low	Low

and MCS can be deployed in a real application without any prior domain knowledge.

Formally, GED finds the number of edit operations which transforms one graph to another, and MCS finds the most common subgraphs between the two graphs. GED, under a particular cost function setup, is equivalent to MCS [4]. Exact computation of GED or MCS, using classical, combinatorial search approaches, however, is NP-Hard. Computing GED and MCS is intractable for larger graphs that have more than a few tens of nodes. Recently, by learning a similarity function to approximate the similarity metric of interest, the works in [1, 2] show that one can efficiently find similar graphs to a query graph, compared to using classical combinatorial methods such as Beam [22], or Bipartite [8, 25]. These methods model the similarity function as a neural network either on the two graph-level embeddings representing the two graphs or on the two sets of node embeddings representing the two graphs. While their performance in both similarity approximation and retrieval tasks approaches that of the classical computations of GED and MCS, they lack several important characteristics that are typically seen in the classical methods, specifically the explainable alignment between nodes or edges between the two graphs.

A similar research setting is also seen in several works solving the graph matching problem [6, 9, 19, 21]. However, their primary goal is fundamentally different. Specifically, graph matching focuses on approximating the similarity between graph entities such as nodes or edges, regardless of the overall structural similarity between the entire graphs. The learned matching is only required to be a discrete distribution over potential correspondences of a node in a graph to each node in the other graph, but not necessarily an injective mapping between nodes. Furthermore, some methods require fine-grained annotations such as node-node or edge-edge similarities in order to learn a sparse correspondence distribution [9]. Our work is different from these methods and primarily focuses on approximating not only the injective assignment, but also capturing the explanation for node deletion and insertions. Hence, this will be similar to the explicit explanability found in classical combinatorial methods by using only annotated graph-level similarities.

It should be noted that a low similarity computation cost and an explainable alignment as part of the computation are two of the desired properties of graph similarity search. To the best of our knowledge, *none of the existing learning-based methods can achieve the explicit node alignment results while approximating the graph-graph similarity score somewhat identical to classical methods.* In this paper, we denote such explicit node-alignment as the "interpretability" of the graph similarity algorithm. In this work, we first propose to learn the graph embedding through a context-aware Graph Neural Network (GNN) model, such as Graph Convolutional Network (GCN), so that we can capture the local and global structures around the nodes in the embedding space. Then, we propose to directly solve for the optimal alignment between nodes of the two graphs from their node-node similarity matrix. However, *our proposed optimal alignment formulation is guaranteed to have a stable and differentiable solution, which is crucial for both gradient-based stochastic optimization of the neural models and finding sparse exact alignment solutions similar to those of classical GED computations.*

We name our method GOTSim. Given two graphs, GOTSim first learns node embeddings using any suitable graph embedding framework, such as GCN [18]. Then, GOTSim directly solves the optimal assignment problem between the two sets of embeddings using a novel differentiable algorithm. Our contributions are as follows:

- Propose a novel optimal assignment objective based on the graphs' node embeddings to approximate graph similarity. In this framework, the similarity between the two graphs is related to the optimal assignment cost, which is the minimum cost of transforming one graph to another in the (node) embedding space. This learning-based framework works on any notion of graph similarity, such as GED or MCS.
- Develop an efficient differentiable algorithm to solve the optimal assignment objective with a polynomial computational complexity, which is also the complexity of existing neural learning-based graph-similarity methods. The algorithm is suitable for any gradient-based learning (such as SGD).
- Demonstrate the effectiveness of our approach on various widely used graph benchmark datasets. GOTSim achieves the lowest error in similarity estimation and superior ranking performance (on several ranking metrics) compared to existing state-of-the-art graph similarity methods.

For domain experts, GOTSim can provide valuable insights into the downstream tasks that are being studied; for example, one can visualize the learned node alignment between two predictably similar protein graphs for further biological investigation. Thus, GOTSim fulfills the desired characteristics of an ideal graph similarity algorithm (as shown in Table 1).

The rest of the paper is organized as follows. We discuss the related work in Section 2. In Section 3, we describe the problem

definition and the details of the proposed GOTSim method. Finally, we present quantitative and qualitative experimental results in Section 4 and conclude our discussion in Section 5.

2 RELATED WORK

We first discuss representation learning in graphs and then discuss other works related to graph similarity and graph matching.

2.1 Graph Representation Learning

GNNs have gained huge popularity in the past few years. These models apply deep neural networks on graph data and can address the limitations of earlier representation learning methods. GCNs are a special class of GNNs, that apply message passing on graphs and compute node representations with input node/edge features and graph structures. These models can be trained with any downstream task to learn node representations specifically for a given task and achieve better performance. Specifically, GCNs use a convolutional layer to perform neighborhood aggregation, thus generating a node's representation by aggregating its own features and its neighbors' features, where the neighbors generally correspond to directly connected nodes. By stacking multiple such convolutional layers, GCNs enable us to estimate context-aware representations of the nodes in graphs.

The initial GCN network, proposed by [18], uses all nodes that are directly connected to v (the target node for which we want to estimate the representation), in addition to v itself (i.e., self-edge), as the neighbors of v. The message is constructed as a projection of the features of the neighboring nodes, and the aggregation is modeled by taking the sum of the incoming messages and normalizing the sum. GAT [31] extends GCN by introducing the attention mechanism as a substitute for the statically normalized convolution operation. The attention weights are used to perform a weighted combination of the messages from the neighbors, as opposed to the vanilla sum used in GCN. We can apply message passing on a graph multiple times to have a node gather information from nodes multiple hops away, instead of just its direct neighbors. When considering tasks on graphs, one critical component to design is the readout function , i.e., the mapping from the set of node representations to a fixed-size vector representation of the graph. A readout function can be a pooling operation (such as mean, max etc., similar to CNNs) or can also have learnable parameters.

The message passing formulation of GCNs has also been extended to heterogeneous graphs, i.e., graphs with multiple node-types and edge-types. Popular examples include Relational Graph Convolution Network [27], Heterogeneous Graph Attention Network [33], Metapath Aggregated Graph Neural Network [11] and Deep Heterogenous Graph Convolutional Networks [20].

2.2 Graph Similarity Search

Searching for graphs involves selection of the appropriate pairwise, graph-graph similarity measure and an efficient similarity computation. In this section, we provide a detailed discussion in this direction.

2.2.1 Characteristics of graph similarity algorithms. Computing similarity between the structured graph objects is a challenging problem with many important real-world applications, in particular, similarity-based retrieval in graph databases. For example, in the field of computer security, we need to search for similar binary functions, where given a binary which may or may not contain code with known vulnerabilities, we wish to check whether any control-flow graph in this binary is sufficiently similar to a database of known vulnerable functions [19].

Graph similarity is typically defined based on (sub-)graph isomorphism [3, 28] or some structural similarity measure such as GED or MCS [24, 34]. Exact similarity computation is known to be computationally expensive in practice. For example, exact GED calculation is NP-Hard and does not scale well to graphs with more than a few tens of nodes. Thus, approximation algorithms have been designed to calculate these similarity measures. These approximate methods can be broadly divided into two categories: (i) search-based similarity computation, and (ii) function-estimation similarity computation.

2.2.2 Search-based similarity computation. Examples of the search-based methods (specifically GED) include Beam search [22], Bipartite approximation [8, 25], and Hausdorff approximation [10]. Beam heuristically explores the search-graph by expanding the most promising node in a limited set, thus approximating the A^* algorithm to find the GED. Bipartite provides an upper bound, while Hausdorff provides a lower bound to the GED. These algorithms essentially estimate the set of edit operations with a minimal cost of transforming one graph to another graph and this minimal cost represents the edit distance between the pair of graphs. Consequently, such optimal edit operations can be used as an explanation of the associated edit similarity score. Such explainable characteristic of these classical algorithms is extremely useful for downstream tasks in domains such as bioinformatics and cheminformatics.

2.2.3 Function-estimation similarity computation. Recently, several function-estimation similarity computation methods have been proposed, including SimGNN [1], GraphSim [2] and Graph Matching Network (GMN) [19]. Function-estimation algorithms have a significant computational advantage over search-based approaches. These approaches learn their functional parameters from empirical data using the principle of empirical risk minimization, and hence they are also suitable for any similarity definition, including GED and MCS. By employing end-to-end deep GNN models, recent works [1, 2] have demonstrated a superior performance over classical methods such as Bipartite or Hausdorff. Although these existing methods effectively estimate the graph similarity, their lack of explanation in node alignments is a crucial limitation.

Our proposed method also belongs to the category of learning-based similarity computation. Unlike existing learning-based approaches, our method is capable of both accurately approximating the graph similarity and providing similar explanation which is found in classical search-based approaches when approximating similarity measures such as the GED.

2.3 Graph Matching

Identifying correspondences between nodes of graphs, such as in the case of calculating GED, is a prevalent problem that arises in different domains, and thus has been studied under various terminologies. A set of fundamentally different techniques commonly

referred to as graph matching or graph alignment have been developed in bioinformatics and computer vision. The techniques developed in these areas, however, are non-exact because large networks without any specific structural properties are commonly studied. In graph matching, given two graphs G_s and G_t with node sets $\mathcal{V}_s$ and $\mathcal{V}_t$, respectively, and adjacency matrices A_s and A_t, respectively, the goal is to find correspondences that can avoid mapping of adjacent nodes in the source graph to different regions in the target graph [26], as expressed in the following maximization objective function:

$$\sum_{(i,i')\in\mathcal{V}_s} \sum_{(j,j')\in\mathcal{V}_t} A_{i,i'} A_{j,j'} S_{i,j} S_{i',j'} \tag{1}$$

Recently, various deep graph-matching methods which employ GNN embeddings have been developed. In soft graph matching methods, such as Graph Matching Network (GMN) [19], Graph Optimal Transport [21], and Optimal Assignment Kernel (OAK) [23], the rectangular matching matrix $S \in \mathbb{R}^{\mathcal{V}_s \times \mathcal{V}_t}$ is a doubly stochastic matrix. In sparse graph matching methods, such as in [9], S is constrained to be a rectangular permutation matrix. To learn the model parameters, either node-level (each pair of nodes of two graphs have a similarity label) or graph-level (each pair of graphs has a similarity label) matching annotation is used.

It can be shown that optimizing the objective in Equation (1) is equivalent to solving Equation (2) in Section 3.2 under certain constraints. When GED is used as a graph-level annotation, graph matching approximates the GED. However, the learned assignments between graph matching methods and classical GED computation can be very different. For example, soft graph matching learns a doubly stochastic matrix, while GED computation learns a permutation matrix. While some of the graph matching works [9] attempt to approximate a sparse assignment matrix (equivalently, this can be seen as an approximation to the permutation matrix), these methods require domain-specific node-level annotation. Thus, they are not capable of estimating the graph-level similarity scores. Furthermore, as discussed previously, the requirement of this type of annotation significantly limits the applications of these approaches in several domains. Contrary to this, our work can approximate the graph similarity while simultaneously learning a sparse permutation assignment only using the graph-level annotation. Furthermore, our method can additionally provide explanation for operations such as deletion and insertion similar to those of classical methods.

3 THE PROPOSED GOTSIM MODEL

We can formally define graph as $\mathcal{G} = \{\mathcal{V}, \mathbf{A}\}$, where $\mathcal{V}$ represents the node set consisting of nodes $\{v_i\}_{i=1,\ldots,N}$ and $\mathbf{A} \in \mathbb{R}^{N\times N}$ is the corresponding adjacency matrix where $\mathbf{A}_{ij}$ represents the edge weight between nodes v_i and v_j. We only consider undirected and unweighted graphs in this work and thus $\mathbf{A}$ is symmetric and generally sparse. However, our method is a general graph-similarity computation framework, and can be extended to other graph types. The notations used in this paper are given in Table 2.

Problem Statement: Given two graphs $G_1 = \{\mathcal{V}_1, \mathbf{A}_1\}$ and $G_2 = \{\mathcal{V}_2, \mathbf{A}_2\}$, the goal of the graph similarity computation task is to approximate the similarity score between the graphs, denoted by $\hat{\mathbf{s}}(G_1, G_2)$.

Table 2: Notations used in this paper.

Notation	Description
$\mathbf{h}_{G,k,i}$	Embedding of i^{th} node of graph G at layer k in the GCN.
$\mathbf{H}_{G,k}$	Set of node embeddings of a graph G at layer k in the GCN.
$\mathbf{C}_{G_1,G_2,k}$	Pairwise similarity matrix between two graphs G_1 and G_2 at layer k in the GCN.
$c_{k,i,j}$	Distance (in embedding space) between the nodes v_i and v_j of G_1 and G_2, respectively, i.e., the substitution cost.
$\mathbf{d}_{k,i}$	Deletion cost of node v_i in G_1.
$\mathbf{a}_{k,j}$	Insertion cost of node v_j in G_2.
$\mathbf{s}(G_1, G_2)$	True similarity score of graphs G_1 and G_2.
$\hat{\mathbf{s}}(G_1, G_2)$	Predicted similarity score of graphs G_1 and G_2.
θ	Parameters of the network.
$\mathbf{M}^*$	Permumation matrix that is the optimal solution to the LP problem.
η	Learning rate.
K_e	Number of training epochs.

The proposed GOTSim model consists of two sequential modules: 1) multiple-scaled convolutional aggregations and 2) differentiable graph matching. For a pair of graphs, GOTSim first computes the vector embeddings of the nodes in the graphs at multiple GCN layers; then, at each layer, GOTSim computes the pairwise similarity matrix between the two sets of node embeddings representing the graphs; finally, GOTSim computes the optimal graph transformation cost, at each layer, from the similarity matrix and approximates the similarity score by aggregating the optimal costs from all GCN layers. The parameters of all computational operations are learned in an end-to-end manner.

3.1 Multiple-scaled Node Embeddings

In our proposed approach we use context-aware node embeddings. To compute them, we employ GCN [7, 18] which uses a convolutional layer to perform neighborhood aggregation, thus generating a node's representation by aggregating its own features and its neighbors' features, where the neighbors generally correspond to directly connected nodes. By stacking multiple convolutional layers, GCNs enable us to learn context-aware representations of the nodes in graphs, when a K-depth GCN is able to estimate the representation of a node using the context within K-hops.

More concretely, the Graph Convolution operator at the k-th GCN layer transforms the representation of node v_i as follows:

$$\mathbf{h}_{G,k,i} = \sigma(\hat{\mathbf{h}}_{G,k-1,i}\mathbf{W}^k + \mathbf{b}^k), \;\; \hat{\mathbf{h}}_{G,k-1,i} = \sum_{j\in\mathcal{N}(i)} \frac{1}{\sqrt{\mathbf{d}_i \mathbf{d}_j}} \mathbf{h}_{G,k-1,j}$$

where $\sigma(\cdot)$ is nonlinear activation, $\mathbf{d}_i$ is the degree of node v_i plus 1 (assuming self-loop), $\mathcal{N}(i)$ represents the set of first-order neighbors of v_i and $\{\mathbf{W}^k, \mathbf{b}^k\}$ represent the filter parameters of the k-th GCN layer, which are shared by all nodes.

In multi-layer GCN, each layer has the effect of capturing a particular higher-order structural information around node v in its representation. While comparing two graphs using their node embeddings, using a particular higher-order representation is usually not sufficient. An intuitive and effective approach is to utilize the embeddings at multiple GCN layers. Specifically, let $\mathbf{H}_{G,k} = \{\mathbf{h}_{G,k,i} \mid v_i \in \mathcal{V}_G, \; i = 1, ..., N\}$ be the set of node embeddings of a

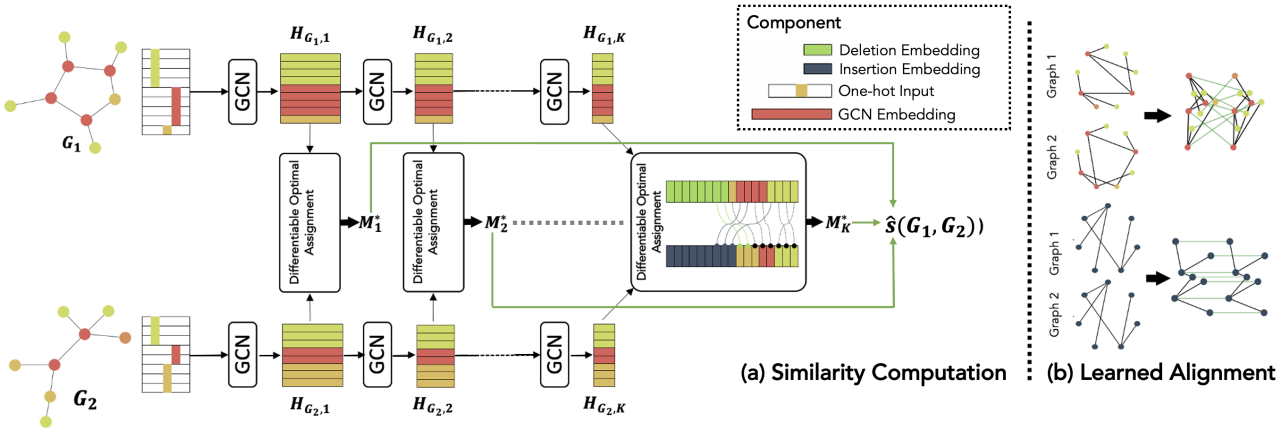


Figure 1: GOTSim model (a) and an illustration of the learned graph matching (b). (a) the one-hot encoded node input are given as input to the GCN, which estimates the node embeddings; the 'Differentiable Optimal Assignment' block computes the optimal assignment between two sets of node embeddings; the final similarity score is the average of the optimal assignment costs for each GCN layer. Solving the optimal assignment problem at each GCN layer results in an interpretable alignment (b) between the two graphs using the information at that layer; for two isormorphic graphs, GOTSim gives the perfect alignment.

graph G at layer k in the GCN, G is represented by K sets of vectors as $\mathbf{H}_G = \{\mathbf{H}_{G,k} \mid k = 1, ..., K\}$.

At each layer k, we can compute the pairwise similarity matrix $\mathbf{C}_{G_1,G_2,k}$ between two graphs G_1 and G_2. Since node orderings are not graph invariant, it is very strict to assume an order in the similarity matrix. For this reason, we take a completely different modeling approach than existing learning-based graph similarity methods and compare graphs using their bag-of-vectors representations.

3.2 Optimal-Assignment Similarity

As discussed earlier, a graph can be represented as a "bag" of its building blocks, e.g., context-aware nodes where the context encodes the structural information around the node. Given two bag-of-vectors $\mathbf{H}_{G_1,k}$ and $\mathbf{H}_{G_2,k}$ from graphs G_1 and G_2, respectively, at layer k, and $\mathbf{c}$ is a pairwise distance function between two node embeddings, we define the similarity matrix between G_1 and G_2 as:

$$\mathbf{C}_{G_1,G_2,k} = \mathbf{H}_{G_1,k}^T \mathbf{H}_{G_2,k} = \begin{bmatrix} \mathbf{c}_{k,1,1} & \cdots & \mathbf{c}_{k,1,N} \\ \vdots & \ddots & \vdots \\ \mathbf{c}_{k,N,1} & \cdots & \mathbf{c}_{k,N,N} \end{bmatrix}$$

where $\mathbf{c}_{k,i,j} = \mathbf{c}(\mathbf{h}_{G_1,k,i}, \mathbf{h}_{G_2,k,j})$ is the distance (in embedding space) between nodes v_i and v_j from G_1 and G_2, respectively. First, let us assume that G_1 and G_2 have the same cardinality N. The graph similarity at layer k can be defined as the optimal transformation cost from G_1 to G_2 as follows:

$$\mathbf{L}_{G_1,G_2,k} = \min_{\mathbf{M}} \mathbf{M} \circ \mathbf{C}_{G_1,G_2,k}, \tag{2}$$

where $\mathbf{M}$ is subject to the following constraints:

$$\sum_i \mathbf{M}_{ij} = 1 \ \ \forall j, \quad \sum_j \mathbf{M}_{ij} = 1 \ \ \forall i, \quad \mathbf{M}_{ij} \geq 0 \ \ \forall i, j \tag{3}$$

Intuitively, this objective is the optimal transformation cost from G_1 to G_2 in the embedding space. When a node v_i in G_1 is assigned to a node v_j in G_2, they should have similar local and global structures, which are captured in the embedding space. In other words, the embedding vectors of $\mathbf{h}_{G_1,*,i}$ (and $\mathbf{h}_{G_2,*,j}$) of v_i (and v_j) are closer in the embedding space. Therefore, the minimal assignment cost directly approximates the similarity between two graphs. Note that this formulation is order invariant. In fact, any permutation of the similarity matrix still results in the same optimal solution.

However, the above formulation has two major problems. First, its solution and the gradients are highly unstable when M is a doubly stochastic matrix [14]. Second, it is only applicable when G_1 and G_2 have the same cardinality N. By changing to a new square matrix M in the formulation of the optimal assignment problem (proposed in this Section) and enforcing a permutation matrix solution (proposed in Section 3.3), we simultaneously achieve both stable solution/gradients and a highly interpretable model.

Without loss of generality, let us now assume that $N_1 \neq N_2$, where $N_1 = |G_1|$ and $N_2 = |G_2|$. In this scenario, besides the node-assignment cost, we need to account for the costs of the node-deletion and the node-insertion operations. Specifically, from the perspective of G_1, there are at most N_2 node-insertion and at most N_1 node-deletion operations. This motivates us to extend the similarity matrix $C_{G_1,G_2,k}$ as follows:

$$\mathbf{C}_{G_1,G_2,k} = \left[\begin{array}{ccc|ccc} \mathbf{c}_{k,1,1} & \cdots & \mathbf{c}_{k,1,N_2} & \mathbf{d}_{k,1} & \cdots & \infty \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{c}_{k,N_1,1} & \cdots & \mathbf{c}_{k,N_1,N_2} & \infty & \cdots & \mathbf{d}_{k,N_1} \\ \hline \mathbf{a}_{k,1} & \cdots & \infty & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \infty & \cdots & \mathbf{a}_{k,N_2} & 0 & \cdots & 0 \end{array}\right] \tag{4}$$

The original similarity matrix is extended to include the cost blocks for deletion and insertion operations. $\mathbf{d}_{k,i}$ denotes the deletion cost of node v_i in G_1 while $\mathbf{a}_{k,j}$ denotes the insertion cost of node v_j in G_2. Note that, a deletion of a node in a graph is equivalent to an

insertion of another node in the other graph, thus, this configuration covers both node insertions and deletions in both the graphs.

There are several ways to define $\mathbf{d}_{k,i}$ and $\mathbf{a}_{k,j}$ from node-embedding vectors $\mathbf{h}_{G_1,k,i}$ and $\mathbf{h}_{G_2,k,j}$ of the two graphs. One can define $\mathbf{d}_{k,i}$ and $\mathbf{a}_{k,i}$ as constant costs, i.e., $\forall i, j$ and $\mathfrak{d}_k, \mathfrak{a}_k \in \{x : x \in \mathbb{R}, 0 \leq x < \infty\}$, $\mathbf{d}_{k,i} = \mathfrak{d}_k$ and $\mathbf{a}_{k,j} = \mathfrak{a}_k$, which are independent of the nodes. However, such definition results in hyperparameter choices, which in turn, require additional training time for model selection. Therefore, we propose to approximate such cost by first defining two global embedding vectors $\mathbf{h}_{k,d}$ and $\mathbf{h}_{k,a}$ and then computing deletion and insertion costs as follows:

$$\mathbf{d}_{k,i} = c(\mathbf{h}_{G_1,k,i}, \mathbf{h}_{k,d}), \quad \mathbf{a}_{k,i} = c(\mathbf{h}_{G_2,k,i}, \mathbf{h}_{k,a}) \tag{5}$$

The global embedding vectors $\mathbf{h}_{k,d}$ and $\mathbf{h}_{k,a}$ are learned in model training, along with other GCN vectors $\mathbf{h}_{G_1,k,i}$ and $\mathbf{h}_{G_2,k,j}$ of the two graphs. We can show that this is related to the Hungarian formulation [25] of approximating the GED. However, our formulation is more powerful because it generalizes to graph embedding and is suitable for similarity function estimation (because of its differentiable solution, which will be discussed in the next section).

Given the optimal assignment cost at each layer-k of the GCN, the final similarity score can be computed from the weighted average of the normalized assignment costs at all GCN levels as:

$$\hat{\mathbf{s}}(G_1, G_2) = 1 - \frac{1}{K}\sum_{k}^{K} \frac{\mathbf{L}_{G_1,G_2,k}}{0.5 \times (N_1 + N_2)} \tag{6}$$

We learn the parameters of GOTSim by minimizing the empirical mean squared error, as follows:

$$\mathcal{L} = \frac{1}{|\mathcal{D}|} \sum_{(G_1,G_2)\in\mathcal{D}} ||\hat{\mathbf{s}}(G_1, G_2) - \mathbf{s}(G_1, G_2)||_2^2, \tag{7}$$

where $\mathbf{s}(G_1, G_2)$ is the target similarity score of graphs G_1 and G_2 and $\mathcal{D}$ is the set of training graph pairs.

3.3 Solving the Optimal Assignment Problem

The objective in Equation (2) admits the solution $\mathbf{M}^*$ that is doubly stochastic. This means that each node v_i in G_1 can be partially assigned to a node v_j in G_2. However, it is known that this Linear Programming (LP) formulation has an unstable and not always unique solution [14] i.e., $\mathbf{L}_k$ is not continuously differentiable.

Fortunately, since our objective is to learn an explainable solution similar to that of the classical GED computations, we have a more stringent constraint: a node in G_1 can be mapped to one and only one node in G_2. This is also true for the insertion and deletion operations. We can easily show that enforcing this sparsity constraint turns our optimal assignment problem into a linear sum assignment program [5]. Consequently, instead of the doubly stochastic matching matrix $\mathbf{M}^*$, the solution is a permutation matrix, i.e., $\mathbf{M}_{ij} \in \{0, 1\}$. The permutation-matrix is also a more stable solution and we can easily compute the derivative of the objective $\mathbf{L}_{G_1,G_2,k}$ with respect to the model's parameters. Specifically, Theorem 1 shows that the model parameters can be efficiently computed by solving for the optimal solution.

Theorem 1. *The derivative of $L_{G_1,G_2,k}$ can be calculated as follows:*

$$\frac{\partial\, \mathbf{L}_{G_1,G_2,k}}{\partial\theta} = \mathbf{M}_k^* \frac{\partial\, \mathbf{C}_{G_1,G_2,k}}{\partial\theta} \tag{8}$$

Algorithm 1: GOTSim Model Training

```
Require: Training graph pairs (G_i, G_j), number of training epochs K_e, learning rate η.
Output: parameters of the network {θ}.
for number of training epochs K_e do
    Sample a minibatch {(G_1, G_2)_l | l = 1, ..., m} of m graph pairs.
    for each graph pair (G_1, G_2) do
        for GCN layer k do
            Compute h_{G_1,i} and h_{G_2,j} ∀ v_i ∈ G_1, v_j ∈ G_2.
            Compute similarity matrix C_{G_1,G_2,k} using Eq. (4).
            Solve for the solution M*_k of RHS of Eq. (2) and compute L_{G_1,G_2,k} using Eq. (2).
            Compute the gradient of L_{G_1,G_2,k} with respect to the network parameters using the Eq. (8).
        end
        Compute ŝ(G_1, G_2) in Eq. (6) using L_{G_1,G_2,k} computed at each GCN layer k.
    end
    Compute the mini-batch loss L using Eq. (7).
    Update θ ← θ − η ∂L/∂θ.
end
```

where $\mathbf{M}_k^*$ is the permutation-matrix solution at layer k.

Proof. Using the chain rule, the derivative of $\mathbf{L}_{G_1,G_2,k}$ is:

$$\frac{\partial\, \mathbf{L}_{G_1,G_2,k}}{\partial\theta} = \frac{\partial\, \mathbf{M}_k^*}{\partial\theta}\mathbf{C}_{G_1,G_2,k} + \mathbf{M}_k^* \frac{\partial\, \mathbf{C}_{G_1,G_2,k}}{\partial\theta} \tag{9}$$

Using sensitivity analysis, we can see that the first derivative $\frac{\partial \mathbf{M}_k^*}{\partial\theta} = 0$ for non-degenerate solutions M_k^*. Intuitively, a infinitesimal change in θ results in the exact same optimal solution $\mathbf{M}_k^*$. Therefore, we arrive at the formulation of the derivative in the Theorem. □

Given Theorem 1, any continuously differentiable distance function $\mathbf{c}$ is suitable. Some examples are the euclidean distance or the cosine distance. In our paper, we employ the cosine distance, which is also used in other learning-based graph similarity methods.

The algorithm of training GOTSim is described in Algorithm 1. GOTSim has a computational complexity of $O(max(N_1, N_2)^{2.5})$ (dominated by the optimal assignment complexity [16]). Also, similar to high computation costs in existing learning-based graph-similarity methods [2, 19], this is a non-trivial cost. The price to pay for the generalization ability of (neural) learning-based graph-similarity methods is that they cannot directly be used for large graphs. In practice, however, GOTSim is very efficient when being trained on all real-world datasets which are evaluated in this paper. Also note that, GOTSim, and other learning-based methods, can easily approximate the similarity between graphs with an order of magnitude more number of nodes than the current limit of existing classical, non-learning based approaches.

Table 3: Results of graph similarity approximation when training with GED targets. On AIDS and LINUX, the ground-truth targets are provided by A^* algorithm (ExactGT). On PTC and IMDB, the ground-truth target of each pair is the minimum GED returned by Bipartite and Beam. The values are in 10^{-3} unit.

	AIDS		LINUX		PTC		IMDB	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ExactGT	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	–	–	–	–
Bipartite	130.68 ± 9.91	337.28 ± 16.11	266.11 ± 5.60	485.26 ± 6.66	60.68 ± 1.10	177.72 ± 3.36	13.22 ± 0.60	69.23 ± 3.93
Hausdorff	77.48 ± 5.64	255.39 ± 11.58	58.07 ± 1.46	221.84 ± 3.41	171.57 ± 7.94	388.30 ± 9.62	24.98 ± 1.71	90.91 ± 4.39
EmbMean	8.34 ± 0.50	69.60 ± 2.15	15.29 ± 1.92	151.12 ± 4.23	10.98 ± 1.22	79.92 ± 7.07	63.72 ± 2.37	102.63 ± 4.24
EmbMax	9.37 ± 0.24	74.44 ± 0.90	13.11 ± 1.88	140.73 ± 4.27	10.60 ± 1.52	80.18 ± 8.29	54.74 ± 1.94	98.12 ± 3.38
EmbGated	5.92 ± 0.32	65.46 ± 2.35	14.54 ± 2.01	119.01 ± 5.17	5.71 ± 2.21	51.05 ± 16.10	6.28 ± 3.23	99.23 ± 4.87
GMN	5.01 ± 0.25	61.23 ± 6.56	7.23 ± 0.94	55.78 ± 3.38	5.82 ± 1.89	56.7 ± 8.20	74.12 ± 4.22	168.02 ± 9.73
SimGNN	2.70 ± 0.30	38.34 ± 0.30	4.43 ± 0.62	50.41 ± 3.52	1.98 ± 0.43	27.85 ± 3.02	9.05 ± 4.12	78.01 ± 3.01
GraphSim	8.60 ± 0.33	37.06 ± 0.21	4.75 ± 0.78	45.72 ± 5.12	5.85 ± 0.83	55.17 ± 4.79	6.87 ± 4.02	111.39 ± 7.96
GOTSim	**2.36 ± 0.12**	**35.19 ± 0.15**	**4.25 ± 0.60**	**44.27 ± 3.23**	**1.90 ± 0.43**	**26.79 ± 0.43**	**5.92 ± 3.19**	**75.31 ± 3.33**

Table 4: Graph similarity retrieval results when training with GED targets. On AIDS and LINUX, the ground-truth targets are provided by A^* (ExactGT). On PTC and IMDB, the ground-truth target is the minimum GED returned by Bipartite and Beam.

	AIDS			LINUX			PTC			IMDB		
	τ	ρ	P@10	τ	ρ	P@10	τ	ρ	P@10	τ	ρ	P@10
ExactGT	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	–	–	–	–	–	–
Bipartite	0.26 ± 0.02	0.37 ± 0.02	0.32 ± 0.04	0.32 ± 0.02	0.43 ± 0.03	0.45 ± 0.03	0.65 ± 0.02	0.82 ± 0.02	0.79 ± 0.03	0.85 ± 0.01	0.92 ± 0.01	0.97 ± 0.01
Hausdorff	0.48 ± 0.02	0.63 ± 0.03	0.54 ± 0.05	0.78 ± 0.01	0.88 ± 0.01	0.78 ± 0.05	0.78 ± 0.01	0.91 ± 0.01	0.90 ± 0.02	0.64 ± 0.01	0.76 ± 0.01	0.88 ± 0.04
EmbMean	0.41 ± 0.04	0.53 ± 0.04	0.60 ± 0.10	0.55 ± 0.02	0.60 ± 0.02	0.51 ± 0.01	0.16 ± 0.04	0.23 ± 0.06	0.41 ± 0.08	0.56 ± 0.03	0.09 ± 0.02	0.09 ± 0.02
EmbMax	0.42 ± 0.02	0.57 ± 0.01	0.61 ± 0.06	0.57 ± 0.03	0.65 ± 0.01	0.71 ± 0.01	0.21 ± 0.03	0.29 ± 0.03	0.49 ± 0.03	0.58 ± 0.02	0.10 ± 0.01	0.23 ± 0.02
EmbGated	0.43 ± 0.02	0.66 ± 0.02	0.70 ± 0.04	0.58 ± 0.03	0.88 ± 0.01	0.81 ± 0.01	0.66 ± 0.10	0.81 ± 0.13	0.84 ± 0.08	0.60 ± 0.03	0.82 ± 0.01	0.41 ± 0.01
GMN	0.47 ± 0.05	0.69 ± 0.02	0.72 ± 0.02	0.78 ± 0.03	0.88 ± 0.01	0.80 ± 0.01	0.40 ± 0.02	0.71 ± 0.03	0.70 ± 0.04	0.46 ± 0.03	0.62 ± 0.02	0.35 ± 0.03
SimGNN	0.67 ± 0.03	0.82 ± 0.02	0.84 ± 0.04	0.80 ± 0.01	0.92 ± 0.01	0.82 ± 0.05	0.80 ± 0.02	0.93 ± 0.01	0.91 ± 0.01	0.71 ± 0.03	0.66 ± 0.02	0.63 ± 0.02
GraphSim	0.68 ± 0.03	0.57 ± 0.03	0.76 ± 0.03	0.83 ± 0.02	0.92 ± 0.02	0.84 ± 0.03	0.79 ± 0.01	0.91 ± 0.01	0.91± 0.02	0.74 ± 0.02	0.71± 0.02	0.66 ± 0.02
GOTSim	**0.72 ± 0.02**	**0.86± 0.02**	**0.87± 0.03**	**0.89 ± 0.02**	**0.92± 0.01**	**0.86± 0.02**	**0.82 ± 0.01**	**0.95± 0.01**	**0.94± 0.01**	**0.80 ± 0.03**	**0.85± 0.01**	**0.73± 0.02**

4 EXPERIMENTAL RESULTS

We evaluate GOTSim against several existing state-of-the-art baselines for GED and MCS computation, with a primary goal of addressing the following aspects of the graph similarity task:

- **(RQ1) Effectiveness**: How accurate is GOTSim, compared to the state-of-the-art approaches, in terms of both similarity score approximation and graph similarity retrieval tasks? The results are presented in Section 4.2.
- **(RQ2) Interpretability**: Can we explain how GOTSim makes its prediction, in a similar way to classical methods like the exact GED algorithm A^* and the approximate GED algorithm (Bipartite)? We discuss this in Section 4.3.

4.1 Experimental Setup

4.1.1 Datasets: We employ four widely-used real-world datasets AIDS, LINUX, IMDB and PTC. For each dataset, we employ five-fold cross validation and split the dataset into 5 subsets: one for validation, one for testing and the rest for training. **AIDS [32]** is a collection of antivirus screen chemical compounds from the Developmental Therapeutics Program at NCI/NIH[1], and has been used in several existing works on graph similarity search [32, 36]. **LINUX [32]** dataset 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 any two statements. **PTC [30]** dataset consists of 344 chemical compound graphs that report the carcinogenicity for male and female rats. Each node in the PTC dataset has one out of 17 possible labels. **IMDB [35]** dataset consists of 1,500 ego-networks of movie actors, where there is an edge if two people appear in the same movie. We follow the same pre-processing steps that are described in [2]. These are the standard datasets that have been used in existing works [1, 19]. While the number of graphs may appear to be small, the similarity is computed between each pair of the graphs, making the total number of unique data-instances (~100K for PTC, ~0.5M for AIDS, ~1M for LINUX, ~2.25M for IMDB).

4.1.2 Evaluation metrics: For evaluation on the similarity approximation task, we report the average Mean Square Error (MSE) and Mean Absolute Error (MAE). For evaluation on the retrieval task, we report Spearman's Rank Correlation Coefficient (ρ) [29], Kendall's Rank Correlation Coefficient (τ) [17], and Precision at k (P@k).

4.1.3 Comparison methods: We evaluate GOTSim against a variety of competing approaches, including various similarity-learning neural approaches and classic approximation approaches. Since our method is related to graph matching approaches and most of these graph matching methods are not specifically designed for the graph similarity task, we select a representative graph matching method (GMN) which has been successfully used to predict graph similarity in our experiments for a fair evaluation. Specifically, we compare our method against the following baselines:

- Bipartite matching [8, 25]: It is a GED approximation deduced from a linear sum assignment of the nodes of the two graphs, which can be efficiently computed in polynomial time.

[1] https://wiki.nci.nih.gov/display/NCIDTPdata/AIDS+Antiviral+Screen+Data

Table 5: Results of graph similarity approximation when training with MCS targets. The ground-truth targets are provided by Mcsplit algorithm (ExactGT). Note that Bipartite is not included because it is not an MCS-estimation algorithm. The values are in 10^{-3} unit.

	AIDS		LINUX		PTC		IMDB	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ExactGT	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
EmbMean	7.83 ± 1.19	71.65 ± 6.44	1.11 ± 0.41	30.46 ± 1.27	8.22 ± 0.68	72.93 ± 3.65	8.63 ± 0.59	109.12 ± 3.75
EmbMax	5.84 ± 0.23	61.76 ± 1.62	1.07 ± 0.37	30.45 ± 1.16	7.92 ± 0.70	72.53 ± 4.49	11.84 ± 0.55	117.32 ± 3.99
EmbGated	5.91 ± 0.46	62.19 ± 3.50	1.03 ± 0.32	56.49 ± 4.03	5.32 ± 0.65	69.89 ± 2.37	4.59 ± 0.50	98.75 ± 3.45
GMN	3.73 ± 0.32	48.55 ± 2.32	0.90 ± 0.06	28.39 ± 2.57	5.01 ± 0.65	49.00 ± 0.94	3.75 ± 0.23	50.23 ± 3.47
SimGNN	3.65 ± 0.65	47.25 ± 4.67	0.71 ± 0.06	19.90 ± 1.07	3.09 ± 0.70	42.14 ± 5.07	2.66 ± 0.31	21.78 ± 3.23
GraphSim	7.28 ± 0.79	68.93 ± 3.97	0.69 ± 0.23	22.18 ± 1.35	8.61 ± 0.72	39.32 ± 1.48	2.60 ± 0.35	21.65 ± 2.89
GOTSim	**2.26 ± 0.53**	**32.94 ± 3.90**	**0.65 ± 0.12**	**21.78 ± 1.01**	**2.97 ± 0.65**	**38.17 ± 1.57**	**2.38 ± 0.40**	**21.12 ± 1.37**

Table 6: Graph similarity retrieval results when training with MCS targets. The ground-truth targets are provided by Mcsplit algorithm (ExactGT). Note that Bipartite is not included because it is not an MCS-estimation algorithm.

	AIDS			LINUX			PTC			IMDB		
	τ	ρ	P@10	τ	ρ	P@10	τ	ρ	P@10	τ	ρ	P@10
ExactGT	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
EmbMean	0.36 ± 0.09	0.46 ± 0.12	0.20 ± 0.04	0.58 ± 0.03	0.40 ± 0.02	0.23 ± 0.01	0.32 ± 0.06	0.42 ± 0.70	0.30 ± 0.01	0.50 ± 0.02	0.67 ± 0.03	0.27 ± 0.04
EmbMax	0.50 ± 0.03	0.64 ± 0.03	0.31 ± 0.01	0.57 ± 0.02	0.41 ± 0.02	0.25 ± 0.01	0.45 ± 0.02	0.57 ± 0.03	0.33 ± 0.04	0.45 ± 0.02	0.38 ± 0.02	0.10 ± 0.03
EmbGated	0.51 ± 0.05	0.65 ± 0.05	0.32 ± 0.04	0.56 ± 0.04	0.43 ± 0.02	0.29 ± 0.02	0.47 ± 0.03	0.72 ± 0.05	0.51 ± 0.08	0.53 ± 0.03	0.62 ± 0.02	0.38 ± 0.02
GMN	0.57 ± 0.02	0.71 ± 0.03	0.45 ± 0.04	0.35 ± 0.03	0.37 ± 0.02	0.25 ± 0.03	0.27 ± 0.03	0.39 ± 0.03	0.31 ± 0.04	0.45 ± 0.02	0.39 ± 0.02	0.28 ± 0.02
SimGNN	0.56 ± 0.04	0.72 ± 0.05	0.47 ± 0.06	0.47 ± 0.02	0.58 ± 0.03	0.59 ± 0.09	0.61 ± 0.04	0.61 ± 0.04	0.61 ± 0.06	0.78 ± 0.03	0.63 ± 0.01	0.49 ± 0.02
GraphSim	0.57 ± 0.04	0.61 ± 0.05	0.27 ± 0.02	0.56 ± 0.02	0.62 ± 0.04	0.65 ± 0.05	0.63 ± 0.02	0.54 ± 0.04	0.42 ± 0.04	0.79 ± 0.01	0.65 ± 0.02	0.50 ± 0.02
GOTSim	**0.58 ± 0.03**	**0.82± 0.03**	**0.67± 0.02**	**0.61 ± 0.01**	**0.68± 0.03**	**0.79± 0.03**	**0.69 ± 0.02**	**0.73± 0.03**	**0.65± 0.02**	**0.81 ± 0.02**	**0.73± 0.03**	**0.52± 0.02**

- Hausdorff matching [10]: Hausdorff-matching is a quadratic-time approximation of graph edit distance based on Hausdorff matching, which underestimates the true distance.
- EmbMean, EmbMax: employ mean-readout and max-readout functions, respectively, on the node-level embeddings to get the graph-level embedding.
- EmbGated [19]: EmbGated employs a projection on the node-level embeddings through a gated network, followed by mean-readout to get the graph-level embedding.
- GMN [19]: a representative graph-matching method which learns a soft matching between nodes.
- SimGNN [1]: SimGNN employs a differentiable attention-based readout function and a non-differentiable histogram function on the node-level embeddings to get the graph-level embedding.
- GraphSim [2]: GraphSim estimates the similarity function by using CNN on the similarity matrices of the node-embeddings.

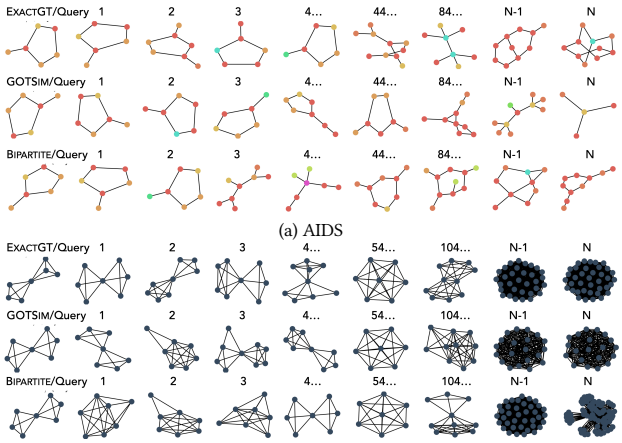
4.1.4 Implementation details. For a fair comparison, all learning-based neural methods use GCN to learn the node embeddings. We the same network architecture for the proposed method and the learning-based graph similarity baselines, for all the datasets. For the GCN-based baselines, the number of GCN layers is 3 and the activation function is ReLU. For GraphSim and SimGNN, we used the same network architectures as reported in their papers.

4.2 Effectiveness of Graph Similarity Approximation (RQ1)

In this section, we present the results of graph similarity estimation and graph retrieval ranking. Tables 3 and 4 show the similarity score estimation and ranking results for the GED problem, respectively. Similarly, Tables 5 and 6 show the similarity score estimation and ranking results for the MCS problem, respectively. GOTSim achieves the lowest estimation error, compared to other approaches (see similarity score estimation results in Tables 3 and 5). In graph retrieval, GOTSim also shows the highest ranking performance across all the datasets. Note that, in the PTC and IMDB experiments reported in Tables 3 and 4, Bipartite and Hausdorff provide the approximate upperbound on the ranking performance because the ground-truth targets are created by the minimum of the estimated GEDs of Beam and Bipartite. It is computationally very expensive to compute the exact ground-truth in these datasets, whose graphs have much higher number of nodes than the graphs in AIDS and LINUX[2]. Note that, all these methods are extremely efficient than classical methods (Bipartite and Hausdorff), which require a significant amount of computation. When the exact ground-truth can be computed (AIDS and LINUX for GED in Tables 3 and 4), GOTSim outperforms Bipartite and Hausdorff. Improvements of our models over the compared methods are statistically significant according to the corresponding paired t-tests (p-value < 0.01).

A Case Study on Graph Ranking: We present the ranking results of two example queries on the AIDS (labeled) and IMDB (unlabeled) datasets. In each example, the ground-truth ranking results (GT) are provided for comparison. In Figure 7, when GOTSim approximates the lower bound of Bipartite and Beam (IMDB dataset), GOTSim ranks several isomorphic graphs (with respect to the query) in the database on the top of the list. GOTSim's retrieved graphs are as relevant to the query as those graphs retrieved by Bipartite. However, it is clear that when the exact ground truth is used as the targets (AIDS dataset), GOTSim top-ranked, retrieved graphs are very close to the ground truths.

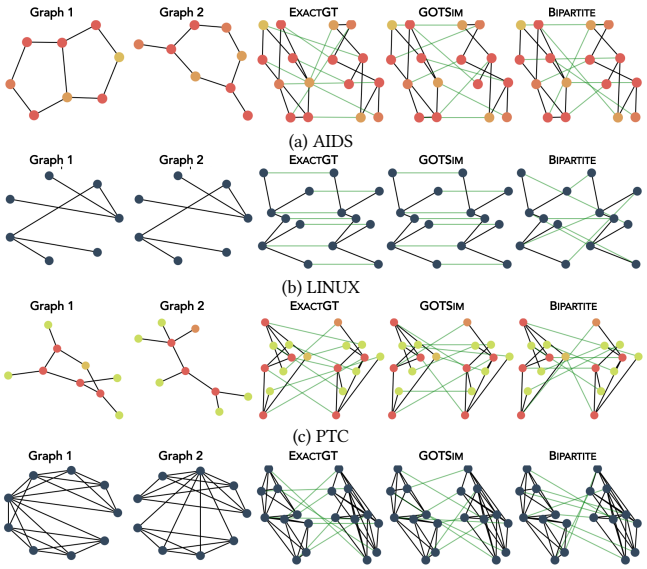
[2]Both GOTSim and GraphSim have very similar running times, which are about 3%-5% longer than SimGNN. It is expected that SimGNN is slightly faster than GOTSim and GraphSim which are more expressive models with superior performance.



(a) AIDS

(b) IMDB

Figure 7: A sample of ranking results under the GED metric on AIDS (labeled) and IMDB (unlabeled) datasets. Each query graph (first column) is presented with a ranked list of graphs. Node labels are color-coded in AIDS.



(a) AIDS

(b) LINUX

(c) PTC

(d) IMDB

Figure 8: Graph matching for GOTSim on the last GCN layer on AIDS (labeled), PTC (labeled), LINUX (unlabeled), and IMDB (unlabeled) datasets. For each pair of graphs, the ExactGT matching is provided by A^*. The cross-graph matchings are shown in green. AIDS Node labels are color-coded.

4.3 Interpretable Graph Matching (RQ2)

In this section, we demonstrate how well GOTSim learns the ground-truth node-assignment in GED estimation, similar to that of classical methods, including the exact A^* (ExactGT) and the approximate Bipartite approaches. Specifically, we visualize the node matching between a pair of graphs which has the optimal assignment cost. Figure 8 shows examples of the AIDS and PTC datasets (with labeled nodes) and examples of the LINUX and IMDB datasets (with unlabeled nodes). The selected pair of graphs have similar number of nodes for simpler visualization and discussion. As we can observe in this figure, GOTSim learns a more similar matching to ExactGT than the matching computed by Bipartite, one of the most representative classical matching algorithms. In the unlabeled graph datasets (LINUX and IMDB), GOTSim recovers almost the exact same ground-truth node matchings between a pair of graphs. In the multi-labeled graph dataset, AIDS and PTC, while it is more difficult to visually investigate the node matchings, we can still observe that GOTSim matches nodes with similar types better than that of Bipartite and is also closer to the ground-truth matchings. In the PTC and IMDB datasets, the result is even more significant because the training labels of all the learning based methods, including GOTSim, are provided by Bipartite and Beam. This implies that GOTSim generalizes better than the non-learning based approaches.

Different from soft graph matching models such as GMN [19] and other approaches [6, 21], GOTSim's matching is highly interpretable because we can easily see the injective mapping between the nodes of one graph to the other. Note that, this experiment is not possible with GraphSim and SimGNN.

5 CONCLUSION

In this paper, we studied the problem of graph similarity and proposed a learning-based graph similarity computation method. Compared to the standard graph prediction problem, graph similarity prediction poses unique challenges and has potential advantages. For example, learning the similarity function directly from the set of node embeddings is non-trivial when there does not exist a canonical ordering of the nodes in the graphs. The proposed GOTSim model directly compares graphs using their (bag of) node embedding vectors which can be learned by using a graph neural network framework. GOTSim directly solves the optimal assignment problem on a novel cost matrix formulation which accounts for node substitution, addition, and insertion. Computationally, the optimal assignment objective can be solved efficiently in polynomial time. Unlike existing optimal assignment based approaches, this objective has a stable optimal matching solution and is differentiable (which allows GOTSim to be efficiently trained using gradient learning). GOTSim also has a distinct feature: it provides an interpretable prediction. When predicting the similarity between a pair of graphs, domain experts can also understand how the model makes its predictions by visually inspecting the optimal matching between the nodes in the two graphs. To the best of our knowledge, GOTSim is the first learning-based graph-similarity method that provides such interpretable results. GOTSim can help in advancing or confirming the domain-specific knowledge in domains which involve graphs.

ACKNOWLEDGMENTS

This work was supported in part by the US National Science Foundation grant IIS-1838730, and Amazon AWS credits.

REFERENCES

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

[2] Yunsheng Bai, Hao Ding, Ken Gu, Yizhou Sun, and Wei Wang. 2020. Learning-Based Efficient Graph Similarity Computation via Multi-Scale Convolutional Set Matching.. In *AAAI*. 3219–3226.

[3] Stefano Berretti, Alberto Del Bimbo, and Enrico Vicario. 2001. Efficient matching and indexing of graph models in content-based retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23, 10 (2001), 1089–1105.

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

[5] Rainer E Burkard, Mauro Dell'Amico, and Silvano Martello. 2009. *Assignment problems*. Springer.

[6] Liqun Chen, Zhe Gan, Yu Cheng, Linjie Li, Lawrence Carin, and Jingjing Liu. 2020. Graph optimal transport for cross-domain alignment. In *International Conference on Machine Learning*. PMLR, 1542–1553.

[7] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*. 2224–2232.

[8] 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.

[9] Matthias Fey, Jan E Lenssen, Christopher Morris, Jonathan Masci, and Nils M Kriege. 2019. Deep Graph Matching Consensus. In *International Conference on Learning Representations*.

[10] Andreas Fischer, Réjean Plamondon, Yvon Savaria, Kaspar Riesen, and Horst Bunke. 2014. A hausdorff heuristic for efficient 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, 83–92.

[11] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. 2020. MAGNN: Metapath Aggregated Graph Neural Network for Heterogeneous Graph Embedding. In *Proceedings of The Web Conference 2020*. 2331–2341.

[12] Carlos Garcia-Hernandez, Alberto Fernandez, and Francesc Serratosa. 2019. Ligand-based virtual screening using graph edit distance as molecular similarity measure. *Journal of chemical information and modeling* 59, 4 (2019), 1410–1421.

[13] Carlos Garcia-Hernandez, Alberto Fernandez, and Francesc Serratosa. 2020. Learning the Edit Costs of Graph Edit Distance Applied to Ligand-Based Virtual Screening. *Current topics in medicinal chemistry* 20, 18 (2020), 1582–1592.

[14] Aude Genevay, Gabriel Peyré, and Marco Cuturi. 2018. Learning generative models with sinkhorn divergences. In *International Conference on Artificial Intelligence and Statistics*. 1608–1617.

[15] José Jiménez-Luna, Francesca Grisoni, and Gisbert Schneider. 2020. Drug discovery with explainable artificial intelligence. *Nature Machine Intelligence* 2, 10 (2020), 573–584.

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

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

[18] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=SJU4ayYgl

[19] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. 2019. Graph matching networks for learning the similarity of graph structured objects. In *International Conference on Machine Learning*. PMLR, 3835–3845.

[20] Saurav Manchanda, Da Zheng, and George Karypis. 2021. Schema-Aware Deep Graph Convolutional Networks for Heterogeneous Graphs. arXiv:2105.00644 [cs.LG]

[21] Hermina Petric Maretic, Mireille El Gheche, Giovanni Chierchia, and Pascal Frossard. 2019. GOT: an optimal transport framework for graph comparison. In *Advances in Neural Information Processing Systems*. 13876–13887.

[22] 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.

[23] Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. 2017. Matching node embeddings for graph similarity. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[24] John W Raymond, Eleanor J Gardiner, and Peter Willett. 2002. Rascal: Calculation of graph similarity using maximum common edge subgraphs. *Comput. J.* 45, 6 (2002), 631–644.

[25] 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.

[26] Ignacio Rocco, Mircea Cimpoi, Relja Arandjelović, Akihiko Torii, Tomas Pajdla, and Josef Sivic. 2018. Neighbourhood consensus networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 1658–1669.

[27] 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 *European Semantic Web Conference*. Springer, 593–607.

[28] Dennis Shasha, Jason TL Wang, and Rosalba Giugno. 2002. Algorithmics and applications of tree and graph searching. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 39–52.

[29] Charles Spearman. 1987. The proof and measurement of association between two things. *The American journal of psychology* 100, 3/4 (1987), 441–471.

[30] Hannu Toivonen, Ashwin Srinivasan, Ross D King, Stefan Kramer, and Christoph Helma. 2003. Statistical evaluation of the predictive toxicology challenge 2000–2001. *Bioinformatics* 19, 10 (2003), 1183–1193.

[31] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *6th International Conference on Learning Representations*. https://openreview.net/forum?id=rJXMpikCZ

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

[33] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous graph attention network. In *The World Wide Web Conference*. 2022–2032.

[34] Peter Willett, John M Barnard, and Geoffrey M Downs. 1998. Chemical similarity searching. *Journal of chemical information and computer sciences* 38, 6 (1998), 983–996.

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

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