Graph Edit Distance Learning via Modeling Optimum Matchings with Constraints

Yun Peng , Byron Choi , Jianliang Xu
Hong Kong Baptist University
{yunpeng, bchoi, xujl}@comp.hkbu.edu.hk

Abstract

Graph edit distance (GED) is a fundamental measure for graph similarity analysis in many real applications. GED computation has known to be NP-hard and many heuristic methods are proposed. GED has two inherent characteristics: multiple optimum node matchings and one-to-one node matching constraints. However, these two characteristics have not been well considered in the existing learning-based methods, which leads to suboptimal models. In this paper, we propose a novel GED-specific loss function that simultaneously encodes the two characteristics. First, we propose an optimal partial node matching-based regularizer to encode multiple optimum node matchings. Second, we propose a plane intersection-based regularizer to impose the one-to-one constraints for the encoded node matchings. We use the graph neural network on the association graph of the two input graphs to learn the cross-graph representation. Our experiments show that our method is 4.2x-103.8x more accurate than the state-of-the-art methods on real-world benchmark graphs.

1 Introduction

Graphs are ubiquitous and have been used in a wide range of applications, such as bioinformatics [Peng *et al.*, 2015], software engineering [Xu *et al.*, 2017], computer vision [Yan *et al.*, 2020] and so on. A fundamental problem in graph data analysis is to compute the similarity between graphs. For example, in bioinformatics, molecules with similar graphs often have similar functions. Computing the similarity of molecular graphs is widely used in virtual screening of molecules. In software engineering, the control-flow of a code fragment can be modeled as a graph. Detecting the similarity of control-flow graphs is useful for code plagiarism detection and code vulnerability search.

Many graph similarity measures are proposed to measure the similarity of graphs. Among others, graph edit distance (GED), *i.e.*, the minimum edit cost of transforming one graph to another, is widely used and domain-agnostic [Bai *et al.*, 2020; Li *et al.*, 2019; Chang *et al.*, 2020; Zhao *et al.*, 2018]. However, computing GED is NP-hard [Zeng *et al.*, 2009]. A recent work [Blumenthal and Gamper, 2020] shows that the exact GED between graphs of more than 16 nodes cannot be computed reliably by the state-of-the-art algorithms in a reasonable time. Therefore, many inexact GED computation methods have been proposed [Blumenthal *et al.*, 2020]. However, it is challenging to handcraft optimal algorithms, even by experts, for different definitions of edit cost.

Meanwhile, with the success of deep learning, learning-based graph similarity/matching computation methods have been receiving increasing research attention [Ma *et al.*, 2019; Yan *et al.*, 2020]. They learn graph similarity/matching models from data by minimizing the loss between the predicted graph similarity/matching and the ground-truth. When used for inference, the learned models can effectively predict the similarity/matching of unseen graphs.

It is well known that GED has two inherent characteristics: *multiple optimum node matchings*, *i.e.*, two graphs have many node matchings that are optimum to produce their GED, and *one-to-one node matching constraints*, *i.e.*, one node of a graph can match only one node of another graph in an optimum node matching. However, these two characteristics have not been well considered in the loss functions of existing graph similarity/matching learning works. For instance, `GraphSim` [Bai *et al.*, 2020], the state-of-the-art graph similarity learning method, employs the mean-square error (MSE) of the predicted graph similarity as its loss function. This generic loss function allows `GraphSim` to support different types of graph similarity measures. Nevertheless, `GraphSim` can only make a suboptimal prediction of GED, as the inherent characteristics of GED are not encoded. On the other hand, `GLMNet` [Jiang *et al.*, 2019] has encoded the one-to-one node matching constraints in the loss function. Let a binary matrix X denote the node matching between two input graphs and $\hat{X}$ denote the prediction. A one-to-one node matching requires that each row and column of X should have just a 1. `GLMNet` proposes the constraint regularizer $\mathcal{L}_{cr}$ that forces the product of each pair of elements in each row or column of $\hat{X}$ to be 0. $\mathcal{L}_{cr}$ itself cannot constrain the one-to-one node matching, as it says nothing to the 1's in $\hat{X}$. But, the sum of $\mathcal{L}_{cr}$ and the cross entropy between $\hat{X}$ and X can constrain the one-to-one node matching. However, GED may have multiple optimum node matchings. If we just select one optimum node matching as the ground-truth, all other optimum node

matchings are overkilled and the model trained is suboptimal. Moreover, it is non-trivial to modify the loss function of GLMNet to work with multiple optimum node matchings. The main reason is that $\mathcal{L}_{cr}$ needs the cross entropy to constrain the one-to-one node matching, but the cross entropy does not work when there are multiple optimum node matchings.

In this paper, we propose a novel GED-specific loss function to address the deficiency of existing works. First, we propose a new *partial node matching-based regularizer* $\mathcal{L}_m$ to encode multiple optimum node matchings. $\mathcal{L}_m$ is inspired by the Branch-and-Bound (BnB) search for exact GED computation and the Monte Carlo tree search (MCTS). The main ideas are as follows. i) We encode all the optimum node matchings covered by a sub-tree of the BnB tree in the ground-truth. The optimum node matchings are leaves of the BnB tree and each internal node is a partial node matching. We use the partial node matching of an internal node as the ground-truth and just penalize the mispredicted 1's in the partial node matching. All the optimum node matchings covered by the sub-tree are not penalized by the regularizer. ii) We select the optimal internal node to balance the mispenalty and the restraint of the regularizer. The reason is that with larger coverage of the internal node, fewer optimum node matchings not covered are mispenalized, which benefits the learning of the model. However, the regularizer is less restrictive, which is adverse to the learning of the model. An internal node is given a score. Inspired by the MCTS, the score is computed by random rollouts. If the rollouts from an internal node reach more optimum node matchings, the internal node has a higher score. If two internal nodes have the same score, we select the one having larger coverage. Using only one optimum node matching as the ground-truth is a special case of our regularizer.

Second, we propose a *plane intersection-based regularizer* $\mathcal{L}_c$ to encode the one-to-one constraints for the node matchings encoded by $\mathcal{L}_m$. Specifically, we leverage the constrained binary quadratic programming (CBQP) formulation of GED. The one-to-one node maching constraints in the CBQP can be modeled as the quadratic infeasibility penalty term $\mathcal{L}_{qip}$ in the loss function [Kochenberger *et al.*, 2014]. However, $\mathcal{L}_{qip}$ becomes not effective to constrain the one-to-one node matching after the binary variables in the CBQP are relaxed to be continuous in $[0, 1]$. We observe that the intersection of the solution plane of minimizing $\mathcal{L}_{qip}$ and that of minimizing the constraint regularizer $\mathcal{L}_{cr}$ is effective to constrain the one-to-one node matching. Therefore, we combine $\mathcal{L}_{qip}$ and $\mathcal{L}_{cr}$ as our plane intersection-based regularizer.

To learn the cross-graph representation of two input graphs G_1 and G_2, we build an association graph G_X from the CBQP of G_1 and G_2. The nodes in G_X model the variables of the CBQP and the edges model the relationship between the variables. G_X can also be regarded as the "product" of G_1 and G_2. The graph neural network on G_X is used to learn the embeddings of the nodes in G_X. The node embeddings are fed into fully-connected layers to predict the GED. Our experiments show that our method is 4.2x-103.8x more accurate than the existing graph similarity learning methods and heuristic GED algorithms. In summary, this paper's contributions are as follows.

- We propose a novel partial node matching-based regularizer to encode multiple optimum node matchings. An optimal partial node matching selection method is proposed.
- We propose a plane intersection-based regularizer to impose the one-to-one constraints for the node matchings encoded by the partial node matching-based regularizer.
- We conduct extensive experiments to verify that our proposed techniques significantly outperform the state-of-the-art methods on several benchmark datasets.

The rest of this paper is organized as follows. Section 2 summarizes the related work. Section 3 presents the preliminaries. The GED-specific loss function and model architecture are proposed in Section 4. Section 5 presents the experimental results and Section 6 concludes this paper.

2 Related Work

This section discusses the recent works that are closely related to this paper.

2.1 Graph Similarity Learning

Most recent graph similarity learning works use the graph neural networks (GNNs) to learn the embeddings of the graphs. The embeddings are fed into multilayer perceptrons (MLPs) or convolutional neural networks (CNNs) to predict the graph similarity. For example, the work [Wang *et al.*, 2019b] uses a Siamese GNN model to predict the graph similarity by regression. The work [Xu *et al.*, 2017] uses a Siamese GNN model to predict if two graphs are similar by binary classification. Different from fusing the embeddings of the two graphs at the very end of learning in above methods, the state-of-the-art works fuse the embeddings of the two graphs early via cross-graph learning. The work [Li *et al.*, 2019] proposes a cross-graph attention network GMN. The loss function of GMN is the classification error of predicting if two graphs are similar. The work GraphSim [Bai *et al.*, 2020] combines GNN and CNN. The loss function of GraphSim is the MSE of the predicted graph similarity. However, as the two inherent characteristics of GED have not been considered, GMN and GraphSim produce a suboptimal prediction. Interested readers are referred to [Ma *et al.*, 2019] for a recent survey.

2.2 Deep Graph Matching

Graph matching is widely studied in computer vision to match images. For example, the work [Wang *et al.*, 2019a] conducts cross-graph learning on the association graph built from the graphs extracted from two images. The embeddings of the nodes of the association graph are fed into the Sinkhorn layer to predict the node matching. The cross entropy of the predicted node matching and the ground-truth is used as the loss. In [Yu *et al.*, 2020], the Hungarian attention is proposed to pay more attention to the mismatched nodes to address the overfitting issue caused by the cross entropy loss. The work [Jiang *et al.*, 2019] introduces a constraint regularizer to encode the one-to-one node matching constraints.

The graph matching methods cannot be directly adopted for GED learning as they do not consider multiple optimum

node matchings. In addition, graph matching has an inherent characteristic of neighborhood consensus [Fey *et al.*, 2020], whereas GED does not.

There are also some works studying learning-based methods to solve the quadratic programming and other problems that can be modeled by the quadratic programming [Vesselinova *et al.*, 2020]. However, the inherent characteristics of GED are also not considered.

3 Preliminaries

This paper studies *undirected graphs*, or simply called *graphs*. A graph is denoted as $G = (V, E)$, where V and E are the node set and the edge set of G, respectively. The label of a node u and an edge (u, u') is denoted by $\ell(u)$ and $\ell(u, u')$, respectively. $N(u)$ denotes the neighbors of $u \in V$. The size of G, denoted by $|G|$, is the number of nodes in G.

A graph G_1 can be transformed to another graph G_2 by a sequence of six types of edit operations: node/edge insertion, node/edge deletion, and node/edge relabeling. The sequence of edit operations is called as the edit path. The cost of the edit path is the total cost of the edit operations in it. Graph edit distance (GED) of G_1 and G_2, denoted by $d(G_1, G_2)$, is the minimum cost of an edit path that transforms G_1 to G_2. In this paper, we assume that the cost of each edit operation is 1, yet our technique can be easily extended to other costs. In an optimum edit path, each node/edge of G_1 is edited at most for one time, as otherwise the edit path's cost is not minimum.

We assume G_1 and G_2 have the same number of nodes, as otherwise we can add dummy nodes with a special label to the smaller graph as in [Chang *et al.*, 2020]. An example of adding dummy nodes is shown in Figure 1(a)-(b).

An optimum edit path is equivalent to an optimum node matching between G_1 and G_2 [Blumenthal *et al.*, 2020]. The intuition is as follows. If a node u or an edge (u_i, u_j) of G_1 is matched to a node or an edge in G_2 that has a different label, u or (u_i, u_j) is relabeled. If an edge (u_i, u_j) of G_1 is matched to a non-existing edge of G_2, (u_i, u_j) is deleted. If a non-existing edge of G_1 is matched to an edge (v_k, v_l) of G_2, (v_k, v_l) is inserted to G_1.

Let a binary matrix X denote the node matching between G_1 and G_2. The optimum X can be computed by the constrained binary quadratic programming (CBQP) as follows.

$$\min_X dist = \sum_{\substack{u_i \in G_1 \\ v_k \in G_2}} c_{i,k} X_{i,k} + \sum_{\substack{u_i, u_j \in G_1 \\ v_k, v_l \in G_2}} c_{i,k,j,l} X_{i,k} X_{j,l} \quad (1)$$

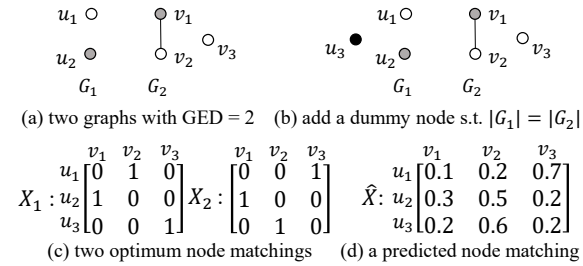
$$s.t. \sum_{v_k \in G_2} X_{i,k} = 1, \forall u_i \in G_1 \quad (2)$$

$$\sum_{u_i \in G_1} X_{i,k} = 1, \forall v_k \in G_2 \quad (3)$$

$$X_{i,k} \in \{0, 1\}, \forall u_i \in G_1, v_k \in G_2, \quad (4)$$

where $c_{i,k}$ is the edit cost of matching u_i in G_1 and v_k in G_2. $c_{i,k} = 1$ if $\ell(u_i) \neq \ell(v_k)$ and 0 otherwise. $c_{i,j,k,l}$ is the edit cost of matching the edge (u_i, u_j) in G_1 and the edge (v_k, v_l) in G_2. $c_{i,j,k,l} = 1$ if $\ell(u_i, u_j) \neq \ell(v_k, v_l)$ and 0 otherwise.

Equations (2-4) constrain that the optimum node matching must be a one-to-one node matching. Otherwise, the node matching cannot transform G_1 to G_2 or the edit cost is not the minimum. Two graphs can have multiple optimum node matchings, as shown in Figure 1(c), as there are many ways to transform a graph to another with the same minimum edit cost. A predicted node matching is shown in Figure 1(d).



(a) two graphs with GED = 2 (b) add a dummy node s.t. $|G_1| = |G_2|$

(c) two optimum node matchings (d) a predicted node matching

Figure 1: An example of two graphs and node matchings. Node labels are marked by gray levels. u_3 is a dummy node with a special label.

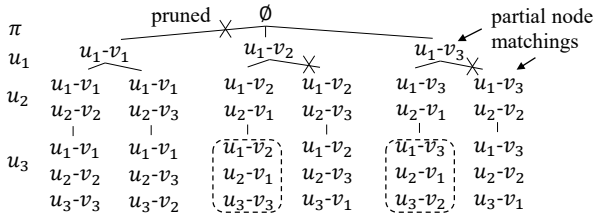


Figure 2: A Brand-and-Bound (BnB) tree for computing the GED of the two graphs in Figure 1(b). Subtrees that do not cover any optimum node matching are pruned during the BnB search. The circled leaves are optimum node matchings.

The Brand-and-Bound (BnB) tree for computing the exact GED between G_1 and G_2 in Figure 1(b) is shown in Figure 2. Each internal node is a partial node matching. π is the order to processing the nodes in G_1, which is decided by heuristics in the latest methods [Chang *et al.*, 2020].

4 Graph Edit Distance Learning

Following the CBQP of GED, we formulate a GED learning problem. The key idea is to design a *GED-specific* loss function. Then, we use a graph neural network on the association graph built from the variables of the CBQP to minimize the GED-specific loss.

4.1 Problem Transformation

We transform the CBQP to a graph learning problem, which can model the two inherent characteristics of GED. Specifically, the CBQP is equivalent to the unconstrained binary quadratic programming (UBQP) by moving the node matching constraints (Equations (2) and (3)) as the quadratic infeasibility penalty to the objective, as follows.

$$\min_X dist + \alpha \mathcal{L}_{qip} \quad (5)$$

$$\mathcal{L}_{qip} = \sum_{u_i \in G_1} (1 - \sum_{v_k \in G_2} X_{i,k})^2 + \sum_{v_k \in G_2} (1 - \sum_{u_i \in G_1} X_{i,k})^2 \quad (6)$$

where $\mathcal{L}_{qip}$ is the quadratic infeasibility penalty and α is the weight.

We further relax the binary variables in the UBQP to $[0, 1]$ to obtain an unconstrained quadratic programming (UQP) such that the variables are differentiable. Then, a GED learning problem can be obtained by minimizing the loss function corresponding to the objective of the UQP as follows.

$$\min \mathcal{L}_d + \alpha \mathcal{L}_{qip}, \tag{7}$$

where $\mathcal{L}_d$ is the mean-square error (MSE) of $dist$ and the ground-truth GED.

4.2 GED-specific Loss Function

Regularizer For Multiple Optimum Node Matchings

The main ideas of the regularizer are as follows. i) We use a partial node matching as the ground-truth, s.t. all the optimum node matchings covered by the partial node matching are not penalized. In this way, multiple optimum node matchings are encoded. ii) We select the optimal partial node matching to balance the mispenality and the restraint of the regularizer.

Specifically, let a binary matrix P denote the partial node matching of an internal node of the BnB tree. P has at most a 1 in each row and column. Our regularizer just penalizes the mispredicted 1's in P as follows.

$$\mathcal{L}_m = -P \log \hat{X}, \tag{8}$$

where $\hat{X}$ is the predicted node matching. Since the 0's in P are not penalized, all the optimum node matchings covered by the internal node are not penalized. Note that $\mathcal{L}_m$ is not the cross entropy and it is different from the Hungarian loss in [Yu *et al.*, 2020], as only the mispredicted 1's are penalized.

We propose a random rollout-based method to select the optimal P. Let X denote an optimum node matching between G_1 and G_2. X is a leaf of the BnB tree. Let $path$ denote the path from the root of the BnB tree to the leaf X. At the i-th node in $path$, denoted by $path[i]$, we conduct k random rollouts. In each rollout, we find a random path from $path[i]$ to a leaf of the BnB tree. If the leaf reached is an optimum node matching, the rollout is called positive. The score of $path[i]$ is the percentage of positive rollouts. If $path[i]$ and $path[j]$ with $i < j$ has the same score, we select $path[i]$ as the optimal internal node, because $path[i]$ is closer to the root of the BnB tree and the subtree rooted at $path[i]$ has a larger coverage.

$\mathcal{L}_m$ cannot constrain the node matchings to satisfy the one-to-one constraints. We next propose another regularizer to encode the one-to-one node matching constraints.

Regularizer To Constrain One-to-one Node Matching

When the variables in the quadratic programming are binary, $\mathcal{L}_{qip}$ can constrain the one-to-one node matching. However, after the variables are relaxed to be continuous in $[0, 1]$, $\mathcal{L}_{qip}$ is not effective. We use an example in the 3D space to show the shortcoming.

Example 1. *Suppose we have three binary variables $x, y, z \in \{0, 1\}$ and we expect there is only one 1 in x, y, z. The solution points are $sol = \{(0, 0, 1), (0, 1, 0), (1, 0, 0)\}$ in 3D space. If we use the constraint $x + y + z = 1$, the valid points are exactly sol. However, if we relax x, y, z to $[0, 1]$ and minimize $(1-(x+y+z))^2$, all points on the plane $x+y+z = 1$ are valid. sol are the intersection points of the plane with the X, Y and Z axes. It is not easy to find sol by only minimizing $(1 - (x + y + z))^2$ with $x, y, z \in [0, 1]$.*

The work [Jiang *et al.*, 2019] proposes the constraint regularizer $\mathcal{L}_{cr}$ to encode the one-to-one node matching constraints as follows.

$$\mathcal{L}_{cr} = \sum_{u_i, u_j \in G_1, v_k, v_l \in G_2} w_{i,j,k,l} X_{i,k} X_{j,l}, \tag{9}$$

where $w_{i,j,k,l} = 1$ if $i = j, k \neq l$ or $i \neq j, k = l$ and 0 otherwise. Intuitively, $\mathcal{L}_{cr}$ is to force the product of each pair of elements in each row/column in X to be 0. However, $\mathcal{L}_{cr}$ is also not effective to constrain the one-to-one node matching after the continuous relaxation, as shown by the following example.

Example 2. *Continuing Example 1, when $x, y, z \in \{0, 1\}$, sol can be obtained by minimizing $xy + xz + yz$. However, if we relax x, y, z to $[0, 1]$, minimizing $xy + xz + yz$ produces the X, Y and Z axes. It is also not easy to find sol. However, sol are exactly the intersection points of the solution of minimizing $xy+xz+yz$ and that of minimizing $(1-(x+y+z))^2$.*

From Examples 1 and 2, we can observe that after relaxing the variables to $[0, 1]$, either $\mathcal{L}_{qip}$ or $\mathcal{L}_{cr}$ does not constrain the one-to-one node matching, but the intersection of their solution planes is a desired choice. Therefore, we propose a plane intersection-based regularizer for encoding the one-to-one node matching constraints as follows.

$$\mathcal{L}_c = \mathcal{L}_{qip} + \mathcal{L}_{cr} \tag{10}$$

Finally, our overall loss function is as follows.

$$\mathcal{L} = \mathcal{L}_d + \alpha \mathcal{L}_c + \beta \mathcal{L}_m, \tag{11}$$

where α and β are two hyperparameters.

Discussion

$\mathcal{L}_c$ and $\mathcal{L}_m$ are crucial for GED learning and should be used *together* to train an effective model. We illustrate the merits of our method by examples as shown in Figure 3.

- *If we only use $\mathcal{L}_c$ and do not use $\mathcal{L}_m$*, we will obtain a suboptimal model. For example, given two graphs with GED = 1 as shown in Figure 3(a), the model converges at the node matching as shown in Figure 3(b) and the predicted GED is ∼2.3.

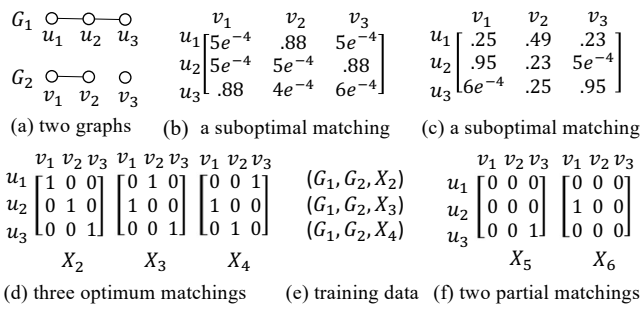


Figure 3: An example showing the effectiveness of $\mathcal{L}_c$ and $\mathcal{L}_m$

- *If we use both $\mathcal{L}_c$ and $\mathcal{L}_m$, but use the full node matching as the ground-truth in $\mathcal{L}_m$*, we will again obtain a suboptimal model. For example, if we use the samples in Figure 3(e) as the training data, the model converges at the node matching as shown in Figure 3(c) and the predicted GED is $\sim$1.5.
- *We propose to use both $\mathcal{L}_c$ and $\mathcal{L}_m$, and use the partial node matching as the ground-truth in $\mathcal{L}_m$.* We will obtain an effective model. For example, if we use samples (G_1, G_2, X_5), (G_1, G_2, X_5) and (G_1, G_2, X_6) in the training data (X_5 and X_6 are partial node matchings shown in Figure 3(f)), the predicted GED is 0.99.

4.3 Graph Neural Network-based Model

We follow the method of [Jiang *et al.*, 2019] to build an association graph of G_1 and G_2 and then use graph convolution on the association graph to learn the representation crossing G_1 and G_2. The embeddings of the nodes are fed into multilayer perceptrons (MLPs) for GED prediction.

The association graph G_X is built from the `UBQP` of G_1 and G_2. Specifically, for each variable in the `UBQP`, G_X has a node. If $c_{i,k} = 1$, the node $X_{i,k}$ is of type 1 and 0 otherwise. Let $t(\cdot)$ denote the type of a node. If $c_{i,k,j,l} = 1$, G_X has an edge $(X_{i,k}, X_{j,l})$. Assume $|V_{G_1}| \leq |V_{G_2}|$ and let $n = |V_{G_2}|$. G_X has n^2 nodes and $|E_{G_1}|n^2 + |E_{G_2}|n^2$ edges. If G_1 and G_2 are sparse, *i.e.*, $|E_{G_1}| = O(|V_{G_1}|)$ and $|E_{G_2}| = O(|V_{G_2}|)$, G_X is also sparse and G_X has $O(n^3)$ edges.

The graph convolution on G_X differentiates the types of the nodes as follows.

$$\vec{h}_u^{L+1} = \sigma\Big(\sum_{\substack{u' \in N(u) \\ t(u')=0}} W_0 \vec{h}_{u'}^{L} + \sum_{\substack{u'' \in N(u) \\ t(u'')=1}} W_1 \vec{h}_{u''}^{L}\Big), \qquad (12)$$

where u is a node in G_X, $\vec{h}_u$ is the embedding vector of u, $N(u)$ is the neighbors of u in G_X, L is the layer of graph convolution, W_0 and W_1 are trainable parameters for the nodes of type 0 and type 1, respectively, and σ is the activation function.

The time complexity of the graph convolution on G_X is $O(n^3)$. Although it is larger than $O(n^2)$ of `GMN` and `GraphSim`, our model has a higher capacity and can produce more accurate GED prediction.

5 Experimental Evaluation

We evaluate our method against a number of state-of-the-art methods for GED computation, with the major goals of addressing the following questions:

Q1: How effective is our method compared to the state-of-the-art methods of GED computation, including both heuristic algorithms and neural network based models?

Q2: How do the regularizer $\mathcal{L}_m$ for encoding multiple optimum node matchings and the regularizer $\mathcal{L}_c$ for node mapping constraints help with the GED learning?

Q3: How our method can generalize to larger graphs that have different distributions of GED and are not seen during training?

Dataset	#graphs	min	max	avg	$\lvert G_{\leq 30}\rvert$
AIDS	42687	2	222	25.6	33013
LINUX	47239	4	2764	35.5	32793
IMDB	1500	7	89	13.0	1423
PTC	344	2	109	25.6	258

Table 1: Statistics of datasets (min, max and avg are the smallest, largest and average graph size, respectively. $|G_{\leq 30}|$ and $|G_{>30}|$ are the number of graphs whose size ≤ 30 and > 30, respectively.)

Datasets. We use four real graph datasets AIDS, IMDB, LINUX and PTC that are from different domains in our experiments. The datasets are the same as those used in [Bai *et al.*, 2020]. We use $G_{\leq 30}$ and $G_{>30}$ to denote the graphs of no more than and more than 30 nodes, respectively. We use $G_{\leq 30}$ to answer Q1 and Q2, and use $G_{>30}$ to answer Q3. Table 1 shows the statistics of the datasets. The GEDs and optimum node matchings are computed by the latest exact GED method [Chang *et al.*, 2020].

Baseline methods. We follow `GraphSim` to compare with three well-known heuristic GED computation algorithms `Beam` [Neuhaus *et al.*, 2006], `Hung` [Riesen and Bunke, 2009], and `VJ` [Fankhauser *et al.*, 2011]. We also compare with two state-of-the-art GED learning methods `GMN` [Li *et al.*, 2019] and `GraphSim` [Bai *et al.*, 2020].

Parameter settings. We use two graph convolution layers and ReLU as the activation function. We use the one-hot encoding of node degree as the initial node embedding. The embedding dimensions are 32. The embedding of a node $X_{u,v}$ in the association graph G_X is fed into a fully-connected layer followed by a sigmoid function to predict $X_{u,v}$. The mean of the embeddings of all nodes in G_X is fed into a fully-connected layer to predict GED. The average mean-square error (MSE) of the predicted GED and the ground-truth on the test data is used as the performance metric. α and β are set by grid search. We set the batch size to 1 and use the Adam optimizer. The initial learning rate is 0.005 and reduced by 0.96 for each 5 epochs. We set the number of epochs to 600, and select the best model based on the lowest MSE of GED on validation data. The experiments are conducted using PyTorch on a server with Intel CPU Xeon Gold 6230R, 768G RAM, and a GPU card NVIDIA Tesla K80. The source code is available online.[1]

5.1 Comparison with Baseline Methods

To answer Q1 with this experiment, we sample 6k, 2k, 2k pairs of graphs in $G_{\leq 30}$ as the training data, the validation data and the test data, respectively. The MSE of GED prediction on the test data is shown in Table 2. From Table 2, we can observe that our method significantly outperforms the state-of-the-art GED learning methods on all datasets. Specifically, the prediction error of our method is respectively 10.6x, 30.2x, 5.4x, and 103.8x smaller than `GraphSim` and even smaller than `GMN` on AIDS, LINUX, IMDB and PTC. Our method is 21.2x, 4.2x, and 4.9x better than the approximate

[1] https://github.com/csypeng/graph_edit_distance_learning

Methods	AIDS	LINUX	IMDB	PTC
Beam	28.04	4.92	**64.16**	25.96
Hung	255.32	59.52	79.36	136.36
VJ	337.12	109.08	77.96	142.01
GMN	349.47	266.02	3712.06	567.26
GraphSim	13.96	35.14	363.20	547.19
Ours	**1.32**	**1.16**	67.55	**5.27**

Table 2: MSE of GED on test data

Methods	AIDS	LINUX	IMDB	PTC
Beam	5.59	2.13	**3.27**	3.23
Hung	**2.69**	1.56	12.41	3.08
VJ	3.15	**1.27**	14.04	**1.59**
GMN	4.69	2.45	7.34	6.55
GraphSim	6.7	6.03	6.12	6.22
Ours	18.17	21.95	20.33	18.51

Table 3: Running time on test data (ms)

algorithms on AIDS, LINUX and PTC, respectively, and is on par with them on IMDB. The IMDB dataset is harder as the graphs are much denser. The average density of IMDB is 0.83, whereas that of AIDS is only 0.09. We also compare with GLMNet. For example, the MSE of GLMNet on AIDS is 265.6. The average GED of the test data is 23.75, 22.92, 52.77, and 35.2 on AIDS, LINUX, IMDB, and PTC, respectively. Our MSEs 1.32 and 1.16 are very small comparing to the GEDs on AIDS and LINUX.

Besides the MSE, we examine the Spearman's rank correlation coefficient ρ, Kendall's rank correlation coefficient τ, and precision at 10 (p@10). For example, on AIDS, ρ, τ, and p@10 of our method is 0.89, 0.85, and 0.73, whereas those of GraphSim are 0.86, 0.81, and 0.61, respectively.

We also compare the efficiency and the result is shown in Table 3. From Table 3, we can observe that the approximate GED algorithms are faster than the learning-based methods. Our method is slower than the latest GED learning methods. The reason is that the time complexity of the graph convolution of our method is $O(n^3)$, where n is the number of nodes in the larger one of the two input graphs. The time complexities of GraphSim and GMN are $O(n^2)$. Our method is still practical as n is small in many real applications.

5.2 Ablation Study

To investigate Q2, in this experiment, we examine the effectiveness of our proposed regularizer $\mathcal{L}_m$ and $\mathcal{L}_c$. The result is shown in Table 4. From Table 4, we can observe that $\mathcal{L}_m$ is important for GED learning. The best results on all datasets are achieved with using $\mathcal{L}_m$ in the loss function. We can observe that $\mathcal{L}_c$ is also important for GED learning. On LINUX and IMDB, the best results are achieved with incorporating $\mathcal{L}_c$ in the loss function.

We also examine the effectiveness of combining $\mathcal{L}_{qip}$ and $\mathcal{L}_{cr}$. On AIDS, the MSE of using $\mathcal{L}_d + \mathcal{L}_m + \mathcal{L}_{qip}$ is 1.56 and the MSE of using $\mathcal{L}_d + \mathcal{L}_m + \mathcal{L}_{cr}$ is 1.61. It verifies that using either $\mathcal{L}_{qip}$ or $\mathcal{L}_{cr}$ is worse than combining them in the loss function.

Methods	AIDS	LINUX	IMDB	PTC
$\mathcal{L}_d$	1.89	7.46	104.84	7.11
$\mathcal{L}_d + \mathcal{L}_m$	1.73	**1.16**	97.12	**5.27**
$\mathcal{L}_d + \mathcal{L}_c$	1.85	4.46	79.53	6.63
$\mathcal{L}_d + \mathcal{L}_m + \mathcal{L}_c$	**1.32**	1.44	**67.55**	6.79

Table 4: MSE of GED prediction of different loss functions

	Beam	Hung	VJ	GraphSim	Ours
MSE	35.4	390.6	590.6	30.9	5.27
time(ms)	7.9	4.45	7.42	10.8	37.2

Table 5: Generalization performance

5.3 Generalization Analysis

We answer Q3 in this experiment. We focus on AIDS and examine the graphs whose node number is more than 30 and no more than 50, denoted by G_{30_50}, due to the time cost of computing the exact GEDs. $G_{\leq 30}$ and G_{30_50} have very different distributions of GEDs: the mean of GEDs of $G_{\leq 30}$ and G_{30_50} is 23.57 and 47.22; the standard deviation of GEDs of $G_{\leq 30}$ and G_{30_50} is 7.83 and 14.17, respectively.

For G_{30_50}, we use 6k pairs of graphs in $G_{\leq 30}$ as the training data and 2k and 2k pairs of graphs in G_{30_50} as the validation and test data, respectively. The generalization performance is shown in Table 5. Comparing with Table 2, we can observe that MSE increases with the growth of graph size. It is reasonable as the distributions of GEDs are different. Our method outperforms existing methods with a small time cost.

6 Conclusion

In this paper, we propose a deep learning-based graph edit distance (GED) computation method. A novel GED-specific loss function is proposed to model the two inherent characteristics of GED. Specifically, we propose a partial node matching-based regularizer to encode multiple optimum node matchings. A partial node matching is used as the ground-truth and all the optimum node matchings covered by the partial node matching are encoded. We can also tune the number of optimum node matchings included to balance the overkill and restraint of the regularizer. We also propose a plane intersection-based regularizer to encode the one-to-one constraints for the node matchings included by the partial node matching-based regularizer. We build an association graph of the two input graphs and use the graph neural network on the association graph to learn the cross-graph representation. Our experiments show that our method is effective and significantly outperforms the state-of-the-art GED learning methods and the heuristic GED algorithms on real-world benchmark graphs.

Acknowledgements

This research was supported in part by the GRFs 12201518, 12201119, R2002-20F, 12200819 and CRF C6030-18GF.

References

[Bai *et al.*, 2020] Yunsheng Bai, Hao Ding, Ken Gu, Yizhou Sun, and Wei Wang. Learning-based efficient graph similarity computation via multi-scale convolutional set matching. In *Proc. of AAAI Conference on Artificial Intelligence (AAAI)*, pages 3219–3226, 2020.

[Blumenthal and Gamper, 2020] David Blumenthal and Johann Gamper. On the exact computation of the graph edit distance. *Pattern Recognition Letters*, 134:46–57, 2020.

[Blumenthal *et al.*, 2020] David Blumenthal, Nicolas Boria, Johann Gamper, Sébastien Bougleux, and Luc Brun. Comparing heuristics for graph edit distance computation. *The VLDB Journal*, 29(1):419–458, 2020.

[Chang *et al.*, 2020] Lijun Chang, Xing Feng, Xuemin Lin, Lu Qin, Wenjie Zhang, and Dian Ouyang. Speeding up ged verification for graph similarity search. In *Proc. of IEEE International Conference on Data Engineering (ICDE)*, pages 793–804, 2020.

[Fankhauser *et al.*, 2011] Stefan Fankhauser, Kaspar Riesen, and Horst Bunke. Speeding up graph edit distance computation through fast bipartite matching. In *Proc. of International Workshop on Graph-Based Representations in Pattern Recognition*, pages 102–111, 2011.

[Fey *et al.*, 2020] Matthias Fey, Jan E. Lenssen, Christopher Morris, Jonathan Masci, and Nils M. Kriege. Deep graph matching consensus. In *Proc. of International Conference on Learning Representations (ICLR)*, 2020.

[Jiang *et al.*, 2019] Bo Jiang, Pengfei Sun, Jin Tang, and Bin Luo. GLMNet: Graph learning-matching networks for feature matching. *CoRR*, abs/1911.07681, 2019.

[Kochenberger *et al.*, 2014] Gary Kochenberger, Jin-Kao Hao, Fred Glover, Mark Lewis, Zhipeng Lü, Haibo Wang, and Yang Wang. The unconstrained binary quadratic programming problem: a survey. *Journal of Combinatorial Optimization*, 28(1):58–81, 2014.

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

[Ma *et al.*, 2019] Guixiang Ma, Nesreen K Ahmed, Theodore L Willke, and Philip S Yu. Deep graph similarity learning: A survey. *arXiv:1912.11615*, 2019.

[Neuhaus *et al.*, 2006] Michel Neuhaus, Kaspar Riesen, and Horst Bunke. Fast suboptimal algorithms for the computation of graph edit distance. In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 163–172, 2006.

[Peng *et al.*, 2015] Yun Peng, Zhe Fan, Byron Choi, Jianliang Xu, and Sourav S Bhowmick. Authenticated subgraph similarity searchin outsourced graph databases. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1838–1860, 2015.

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

[Vesselinova *et al.*, 2020] Natalia Vesselinova, Rebecca Steinert, Daniel F. Perez-Ramirez, and Magnus Boman. Learning combinatorial optimization on graphs: A survey with applications to networking. *IEEE Access*, 8:120388–120416, 2020.

[Wang *et al.*, 2019a] Runzhong Wang, Junchi Yan, and Xiaokang Yang. Neural graph matching network: Learning lawler's quadratic assignment problem with extension to hypergraph and multiple-graph matching. *arXiv:1911.11308*, 2019.

[Wang *et al.*, 2019b] Shen Wang, Zhengzhang Chen, Xiao Yu, Ding Li, Jingchao Ni, Lu-An Tang, Jiaping Gui, Zhichun Li, Haifeng Chen, and S Yu Philip. Heterogeneous graph matching networks for unknown malware detection. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3762–3770, 2019.

[Xu *et al.*, 2017] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. Neural network-based graph embedding for cross-platform binary code similarity detection. In *Proc. of ACM SIGSAC Conference on Computer and Communications Security*, pages 363–376, 2017.

[Yan *et al.*, 2020] Junchi Yan, Shuang Yang, and Edwin R Hancock. Learning for graph matching and related combinatorial optimization problems. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4988–4996, 2020.

[Yu *et al.*, 2020] Tianshu Yu, Runzhong Wang, Junchi Yan, and Baoxin Li. Learning deep graph matching with channel-independent embedding and hungarian attention. In *Proc. of International Conference on Learning Representations (ICLR)*, 2020.

[Zeng *et al.*, 2009] Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *Proc. VLDB Endow.*, 2(1):25–36, 2009.

[Zhao *et al.*, 2018] Xiang Zhao, Chuan Xiao, Xuemin Lin, Wenjie Zhang, and Yang Wang. Efficient structure similarity searches: A partition-based approach. *The VLDB Journal*, 27(1):53–78, 2018.