Graph Edit Distance Estimation: A New Heuristic and A Holistic Evaluation of Learning-based Methods

MOUYI XU, The University of Sydney, Australia
LIJUN CHANG, The University of Sydney, Australia

Graph edit distance (GED) is an important metric for measuring the distance or similarity between two graphs. It is defined as the minimum number of edit operations required to transform one graph into another. Computing the exact GED between two graphs is an NP-hard problem. With the success of deep learning across various application domains, graph neural networks have also been recently utilized to predict the GED between graphs. However, the existing studies on learning-based methods have two significant limitations. (1) The development of deep learning models for GED prediction has been explored in various research fields (e.g., databases, machine learning, information retrieval, and computer vision), yet cross-field evaluations have been quite limited. (2) More importantly, all these advancements have been evaluated against a simple combinatorial heuristic baseline, with their models shown to outperform it. In this paper, we aim to bridge this knowledge gap. We first conduct a holistic review of the existing learning-based methods, categorizing them into non-interpretable and interpretable GED prediction approaches, while highlighting their overarching design principles and relationships among these models. Secondly, we present a simple yet effective combinatorial heuristic algorithm App-BMao for GED estimation, adapted from an existing exact GED computation algorithm. App-BMao provides interpretable GED estimation with controlled time and space complexity. Extensive empirical evaluations on three widely used datasets show that the new heuristic algorithm App-BMao outperforms all existing learning-based approaches for both interpretable and non-interpretable GED prediction.

CCS Concepts: • **Information systems** → **Similarity measures**; • **Mathematics of computing** → **Graph algorithms**; • **Computing methodologies** → *Supervised learning by regression.*

Additional Key Words and Phrases: Graph Edit Distance, Heuristic Algorithms

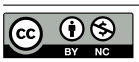
ACM Reference Format:
Mouyi Xu and Lijun Chang. 2025. Graph Edit Distance Estimation: A New Heuristic and A Holistic Evaluation of Learning-based Methods. *Proc. ACM Manag. Data* 3, 3 (SIGMOD), Article 167 (June 2025), 24 pages. https://doi.org/10.1145/3725304

1 Introduction

Graph edit distance (GED) is a fundamental metric for measuring the distance or similarity between two graphs [31]. It has been widely used in various applications including molecular chemistry, pattern recognition, graph database, computer vision and biology. The GED between two graphs is defined as the minimum number of edit operations that are needed to transform one graph into another. There are totally six allowed edit operations:

- Change the label of a node or an edge.
- Delete or insert an edge between two existing nodes.
- Delete or insert an isolated node (i.e., without adjacent edges).

Authors' Contact Information: Mouyi Xu, moxu7046@uni.sydney.edu.au, The University of Sydney, Sydney, Australia; Lijun Chang, Lijun.Chang@sydney.edu.au, The University of Sydney, Sydney, Australia.



ACM 2836-6573/2025/6-ART167
https://doi.org/10.1145/3725304

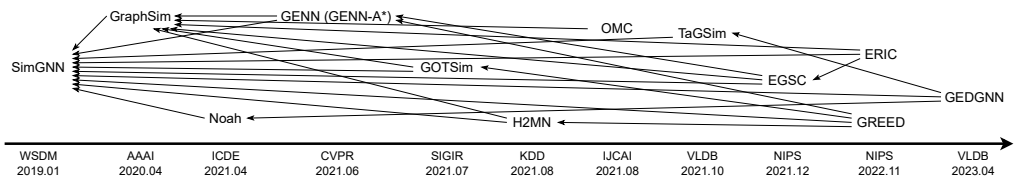


Fig. 1. A diagram summarizing existing learning-based models for GED prediction. A directed edge from one model to another means that the latter model is compared in the paper that proposes the former model. Under the long horizontal arrow shows the publication venue and date of the corresponding paper.

Although the problem of GED computation is NP-hard [41], a series of exact algorithms have been designed in the literature in view of the fundamental importance of the problem. For example, A*GED [29, 30] used best-first search with a simple label set-based lower bound for pruning. DF-GED [3, 9] and CSI-GED [17] later improved the performance by conducting a depth-first search and designing tighter lower bounds. The state-of-the-art algorithms, AStar-LSa [11] and AStar-BMao [12], again used best-first search, but proposed much tighter lower bounds than the previous ones by formulating the idea of anchor-aware lower bounds.

Due to the NP-hardness nature of the problem, even the state-of-the-art exact algorithms may take an excessive long time in processing some input graph pairs. With the development and extreme success of deep learning across various application domains such as NLP, computer vision and speech recognition, learning-based techniques (e.g., graph neural networks) have also been developed recently for graph data analysis, e.g., to predict the GED value between two graphs. Example approaches include SimGNN [5], GraphSim [6], Noah [39], GENN [36], GENN-A* [36], GOTSim [14], TaGSim [4], EGSC [26], and GEDGNN [25].

The existing studies on learning-based methods for GED prediction have two significant limitations. Firstly, the development of deep learning models for GED prediction has been explored in various research fields, e.g., database, data mining, machine learning, information retrieval, and computer vision. But, cross-field comparison and evaluation have been quite limited. For example, as shown in Figure 1, papers published in database venues (e.g., VLDB and ICDE) typically ignored papers published in other venues, and vice versa. Secondly and more importantly, the advancements achieved by the existing learning-based approaches were only evaluated against a simple combinatorial heuristic baseline algorithm (Hungarian [22] or its variants Munkre [23, 28] and VJ [15, 19]). Outperforming such a baseline algorithm is not reliable enough to show the effectiveness of the learning-based approaches compared to non-learning-based approaches for GED prediction.

In this paper, we aim to bridge this knowledge gap of studying learning-based approaches for GED estimation, by conducting a holistic experimental evaluation. We first review the existing learning-based methods, categorizing them into non-interpretable and interpretable GED prediction approaches, while highlighting their overarching design principles and relationships among these models.[1] Non-interpretable GED prediction approaches only output an estimated GED value, which could be either larger or smaller than the true GED value. Example approaches include SimGNN [5], GraphSim [6], H2MN [42], TaGSim [4], EGSC [26], GENN [36], ERIC [43], OMC [24], and GREED [27]. We show that these approaches share a general three-module framework, see Section 4.2. Module-I is a Siamese GNN that separately maps the two input graphs into both

[1]We notice that another survey was also conducted recently in [40], concurrent to our work. However, [40] does not go into the technical details of the models as we do. Also, non-learning-based methods are not discussed and experimental evaluation was lacking in [40].

node-level embeddings and graph-level embeddings. Module-II fuses together those embeddings into fused embedding vectors, which are then fed to Module-III for making the final GED prediction. Module-II typically uses neural tensor network [34] or the attention mechanism, while Module-III typically uses a multilayer perceptron.

Learning-based approaches for interpretable GED prediction outputs an edit path (more precisely, a node mapping) along with its cost as the estimated GED value. The outputted edit path serves as an evidence of the estimated GED value, and thus makes the prediction interpretable. Example approaches include GENN-A* [36], Noah [39], GOTSim [14] and GEDGNN [25]. We further divide them into two categories, tree traversal-based approaches and cost matrix-based approaches. The former, including GENN-A* and Noah, conducts a traversal over a search tree (same as exact GED computation algorithms) and invokes a neural network (e.g., one of those non-interpretable GED prediction models) to predict lower bounds during tree traversal. The latter, including GOTSim and GEDGNN, learns a cost matrix and a matching matrix and then extracts multiple candidate node mappings from them, whereas the one with the lowest cost is retained for GED estimation.

In our experimental evaluation, we further present a simple yet effective combinatorial heuristic algorithm App-BMao for interpretable GED estimation, which is adapted from the existing exact GED computation algorithm of [12]. App-BMao takes an input parameter t for controlling the time and space complexity. The time complexity of App-BMao for estimating the GED between graphs q and g is $O(t \times (|V(q)| + |V(g)|)^3)$, and its space complexity is $O(t + |V(q)| \times |V(g)|)$, where $V(q)$ and $V(g)$ are the vertex sets of q and g, respectively. We prove that App-BMao provides a more accurate GED estimation when t increases.

Our comprehensive experimental evaluations reveal the following key findings.

- Among non-interpretable GED prediction models, GREED, ERIC, EGSC and GENN generally outperform other approaches, though no clear winner among them.
- Among interpretable GED prediction approaches, the tree traversal-based approach (specifically, GENN-A*) works well on small graphs while the cost matrix-based approach (specifically, GEDGNN) perform better on large graphs (where GENN-A* runs out-of-memory).
- The new heuristic algorithm App-BMao with $t = 100$ outperforms all existing learning-based approaches (including both interpretable and non-interpretable GED prediction) as well as a state-of-the-art heuristic algorithm SDTED [7], in terms of both estimation accuracy and running efficiency.

The main contributions of our paper are summarized as follows.

- We conduct a holist review of the existing learning-based approaches for GED prediction.
- We present a new heuristic algorithm for interpretable GED estimation, with controlled time and space complexities.
- We conduct extensive experiments to evaluate the existing learning-based approaches against each other, as well as against the new heuristic algorithm App-BMao.

2 Preliminaries

We consider labeled and undirected simple graphs $g = (V(g), E(g), l)$ with node set $V(g)$ and edge set $E(g)$. Here, $l : V(g) \cup E(g) \rightarrow \Sigma$ is a labeling function that assigns each node and edge a label from a given label set Σ. That is, the label of node $v \in V(g)$ is $l(v)$ and the label of edge $(v, v') \in E(g)$ is $l(v, v')$. If a graph has no node (or edge) labels, we assign all the nodes (or edges) with the same special label $\dashv \notin \Sigma$. For presentation simplicity, we refer to a labeled and undirected simple graph as a graph.

Definition 2.1. Given two graphs $q = (V(q), E(q), l)$ and $g = (V(g), E(g), l)$ with the same number of nodes, q is **isomorphic** to g if there exists a bijective mapping $f : V(q) \rightarrow V(g)$ such that

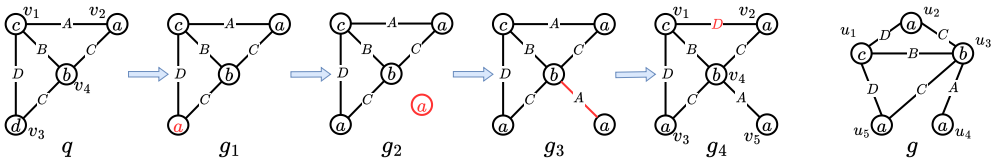


Fig. 2. An example edit path between q and g

- $\forall v \in V(q), l(v) = l(f(v))$.
- $\forall v, v' \in V(q)$, $(v, v') \in E(q)$ if and only if $(f(v), f(v')) \in E(g)$.
- $\forall (v, v') \in E(q), l(v, v') = l(f(v), f(v'))$.

For example, in Figure 2, g_4 is a graph with node set $\{v_1, v_2, \ldots, v_5\}$ and six edges, where $\Sigma = \{a, b, c, A, B, C, D\}$. g_4 is isomorphic to g with $f = \{v_1 \mapsto u_1, v_2 \mapsto u_2, v_3 \mapsto u_5, v_4 \mapsto u_3, v_5 \mapsto u_4\}$.

Definition 2.2. Given two graphs q and g, an **edit path** between q and g is a sequence of labeled graphs starting with q and ending at a graph that is isomorphic to g, such that each graph in the sequence is obtained from its predecessor by applying exactly one of the following six edit operations:

- Change the label of a node or an edge.
- Delete or insert an edge between two existing nodes.
- Delete or insert an isolated node (i.e., without adjacent edges).

For example, Figure 2 shows an edit path $P = (q, g_1, g_2, g_3, g_4)$ between q and g, where g_4 is isomorphic to g as illustrated above. Note that, equivalently, the edit path could also be defined as the sequence of edit operations that are used in the transformations.

Definition 2.3. Given two graphs q and g, the **graph edit distance** (abbreviated as **GED**) between q and g is defined as the minimum length among all possible edit paths between q and g,

$$\text{ged}(q, g) = \min_{P \in \mathcal{P}(q,g)} |P|$$

where $\mathcal{P}(q, g)$ denotes the set of all edit paths between q and g, and $|P|$ is the length of an edit path P.[2]

Note that, $\text{ged}(\cdot, \cdot)$ is a metric [31]. The GED between q and g in Figure 2 is 4, and the edit path shown in Figure 2 is a shortest edit path between q and g. However, computing the exact GED between two graphs is an NP-hard problem [41].

Problem Statement. Given two graphs q and g, we in this paper study the problem of efficiently computing a good estimation $\widehat{\text{ged}}(q, g)$ to the GED between q and g.

3 Exact GED Computation

In this section, we review the general ideas of the state-of-the-art exact GED computation algorithms [11, 12], which are also followed by the existing deep learning-based interpretable GED prediction methods that will be discussed in Section 4.3.

[2]Note that, we focus on the case of uniform edit cost in this paper, i.e., all six edit operations in Definition 2.2 have the same cost. For non-uniform edit costs, $|P|$ is the sum of the costs of all edit operations in P; this is beyond the scope of this paper.

3.1 Compute GED via Node Mapping

According to Definition 2.3, the GED between two graphs q and g can be obtained by enumerating all edit paths between them. However, the search space of all possible edit paths is extremely large. Instead, the state-of-the-art exact GED computation algorithms, e.g., [11, 12], enumerate all node mappings between q and g, by noting that (1) each edit path uses a node mapping (for determining the isomorphism between the last graph in the edit path and g), and (2) different edit paths may use the same node mapping. Thus, the number of distinct node mappings is significantly smaller than the number of distinct edit paths. Moreover, given a node mapping f, we can easily find the shortest edit path that uses this mapping in $O(|V(q)|+|V(g)|+|E(q)|+|E(g)|)$ time [11]. For example, for q and g in Figure 2 with mapping $f = \{v_1 \mapsto u_1, v_2 \mapsto u_2, v_3 \mapsto u_5, v_4 \mapsto u_3\}$, the smallest number of edit operations could be achieved by (1) relabeling node v_3 since $l(v_3) \neq l(f(v_3))$, (2) relabeling edge (v_1, v_2) since $l(v_1, v_2) \neq l(f(v_1), f(v_2))$, (3) adding a new node v_5 with label a (which is mapped to u_4), and then (4) adding an edge between v_5 and v_4 with label A. Note that, the edit path in Figure 2 conducts these four edit operations in a slightly different order.

Without loss of generality, ***we assume that*** $|V(q)| \leq |V(g)|$; if this is not the case, we can simply swap q and g since $\text{ged}(\cdot,\cdot)$ is a metric. It is proved in [11] that when $|V(q)| \leq |V(g)|$, the operation of deleting a node from q will never be used in the optimal edit path. Thus, we only need to consider ***injective node mappings*** from q to g, and simply refer to them as node mappings in the remainder of the paper. We refer to the length of the shortest edit path that uses node mapping f as the ***editorial cost*** of f, denoted $\text{edc}_f(q, g)$. Then, the GED between q and g can be computed by enumerating all node mappings from q to g, as shown in the lemma below.

Lemma 3.1. *[11] Given two graphs q and g with $|V(q)| \leq |V(g)|$, the GED between q and g satisfies*

$$\text{ged}(q, g) = \min_{f \in \mathcal{F}(q,g)} \text{edc}_f(q, g)$$

where $\mathcal{F}(q, g)$ denotes the set of all node mappings from q to g.

3.2 Find the Best Node Mapping via a Prefix-shared Search Tree

The total number of distinct node mappings, although being much smaller than that of edit paths, is still exponential, i.e., $\frac{|V(g)|!}{(|V(g)|-|V(q)|)!}$. To efficiently find the ***best node mapping*** (i.e., *the one with the smallest editorial cost*), the state-of-the-art algorithms [11, 12] organize all node mappings into a prefix-shared search tree by following a fixed matching order of $V(q)$. For example, Figure 3 shows a search tree for the matching order $\pi = (v_1, v_2, v_3, v_4)$. Each leaf state of the search tree corresponds to a ***full*** node mapping, while intermediate states represent ***partial*** node mappings. For example, $f_0 = \emptyset$, $f_1 = \{v_1 \mapsto u_1\}$, $f_5 = \{v_1 \mapsto u_1, v_2 \mapsto u_2\}$, and $f_{14} = \{v_1 \mapsto u_1, v_2 \mapsto u_2, v_3 \mapsto u_3, v_4 \mapsto u_4\}$. We say f' ***extends*** f if $f \subset f'$. Thus, the node mapping of a search tree state extends that of its parent by matching one more node. Let $|f|$ denote the number of q's nodes that are mapped by f. Then, f is a full node mapping if $|f| = |V(q)|$, and is a partial node mapping otherwise.

To avoid exhaustively visiting all node mappings, lower bounds are computed for node mappings.

Definition 3.2. The **lower bound** of a *partial* node mapping f, denoted lb_f, is defined as a value that is no larger than the minimum editorial cost among all full node mappings that extend f. The lower bound of a *full* node mapping f is defined as $\text{edc}_f(q, g)$.

The existing algorithms traverse the search tree by using different strategies, e.g., A*, depth-first-search, and beam search. The pseudocode of traversing the search tree by using A* is shown in Algorithm 1. It uses a priority queue Q to store the search frontier where each element of Q is a (either partial or full) node mapping f together with its lower bound lb_f; the ranking in Q is based on the lower bound. Based on the property of A* search and the fact that $\text{lb}_f = \text{edc}_f(q, g)$

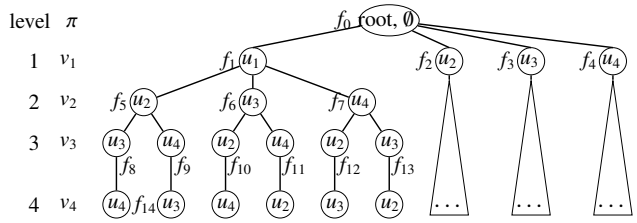


Fig. 3. Search tree [12]

Algorithm 1: GED-A*(q, g) [11]

Input: Two graphs q and g with $|V(q)| \leq |V(g)|$
Output: $\text{ged}(q, g)$

```
1 Compute a matching order π = (v_1, ..., v_|V(q)|) of V(q);
2 Q ← {(0, ∅)}; /* Initialize a min-priority queue by the root of the search tree */;
3 while true do
4     (lb_f, f) ← pop the top entry from Q;
5     if |f| = |V(q)| then return lb_f(q, g);
6     for each node u ∈ V(g) \ f do
7         f' ← f ∪ {v_|f|+1 ↦ u};
8         Compute a lower bound lb_f' of f';
9         Push (lb_f', f') into Q;
```

when f is a full node mapping, the first full node mapping popped from Q will be an optimal node mapping [11].

4 Learning-based Approaches for GED Prediction

Deep learning (i.e., neural networks) has recently shown great success in various application domains, such as NLP and computer vision. Without exception, it has also been applied to analyze graph data, e.g., to predict the GED value between two graphs. Learning-based approaches for GED prediction can be divided into two categories, non-interpretable GED prediction and interpretable GED prediction. The former simply treats the problem as a regression task, and predicts a value that could be either smaller or larger than the true GED. On the other hand, the latter outputs an edit path along with its corresponding editorial cost; as a result, the predicted value will always be no smaller than the true GED. In this section, we review the existing learning-based approaches in each category, by focusing on their overarching design principles. Before that, we first introduce graph neural networks in Section 4.1, which is the backbone of these learning-based approaches.

Notation Conventions. We use boldface lower case letters such as $\boldsymbol{h}$ to denote vectors, and boldface upper case letters such as $\boldsymbol{W}$ and $\boldsymbol{C}$ to denote matrices. All vectors used in this paper are assumed to be column vectors.

4.1 Graph Neural Networks

Graph Neural Networks (GNNs) transform individual nodes or even entire graphs into high-dimensional vector representations. A defining feature of GNNs is their use of neural message passing, where vector messages are exchanged between nodes and updated using neural networks [18]. For a given hyper-parameter K, there are K layers of message passing in a GNN, and a

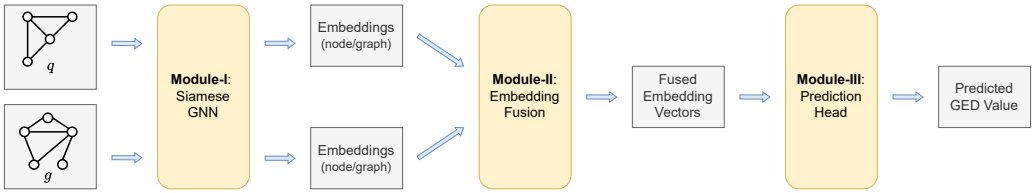


Fig. 4. General framework for non-interpretable GED prediction; Rounded rectangles represent the model (i.e., modules), while regular rectangles represent data (including input graphs, embedding vectors, and predicted GED value)

hidden embedding $\boldsymbol{h}_v^{(k)}$ is computed for each node $v \in V$ in each layer $1 \leq k \leq K$. Specifically, the message-passing updates can be expressed as

$$\boldsymbol{h}_v^{(k)} = \textsc{update}^{(k)}\left(\boldsymbol{h}_v^{(k-1)}, \textsc{agg}^{(k)}\left(\left\{\boldsymbol{h}_u^{(k-1)} : u \in N(v)\right\}\right)\right) \tag{1}$$

where $N(v)$ is the set of v's neighbors, and $\textsc{update}^{(k)}$ and $\textsc{agg}^{(k)}$ (AGG is short for aggregate) are arbitrary differentiable functions (e.g., neural networks) with learnable parameters. The initial embeddings $\{\boldsymbol{h}_v^{(0)} : v \in V\}$ are set as the nodes' input features, e.g., node labels in our problem setting. In this way, $\boldsymbol{h}_v^{(k)}$ is computed based on the input features of as well as the connections among the nodes whose shortest distances to v are at most k. In the literature, different GNN models have been designed by proposing their UPDATE and AGG functions. Examples include Graph Convolutional Network (GCN) [21], Graph Isomorphism Network (GIN) [38] and Graph Attention Network (GAT) [35].

Equation (1) computes vector embeddings for individual nodes. To obtain a vector embedding $\boldsymbol{h}_g$ for the entire graph g, a *graph pooling* operation is applied.

$$\boldsymbol{h}_g = \textsc{pool}\left(\left\{\boldsymbol{h}_v^{(k)} : v \in V, 1 \leq k \leq K\right\}\right) \tag{2}$$

In many applications, the graph pooling operation simply takes the average or sum of the node embeddings that are obtained at the last layer of the GNN, e.g., $\boldsymbol{h}_g = \frac{1}{|V|}\sum_{v \in V} \boldsymbol{h}_v^{(K)}$.

4.2 Non-interpretable GED Prediction

Designing deep learning-based approaches for non-interpretable GED prediction was pioneered by the SimGNN model [5], which is an end-to-end neural network that learns a function to map a pair of graphs into a similarity score. SimGNN consists of three modules [3] as shown in Figure 4, where the general three-module framework is also followed by later approaches. Module-I is a GNN that maps a graph into embedding vectors, as discussed in Section 4.1. Specifically, SimGNN uses GCN [21]. With learnable parameters $\boldsymbol{\theta}_{\text{GNN}}$ (here, we summarize all learnable parameters used in the GNN into $\boldsymbol{\theta}_{\text{GNN}}$), SimGNN computes both node-level embeddings $\{\boldsymbol{h}_v : v \in V(q)\}$ and a graph-level embedding $\boldsymbol{h}_q \in \mathbb{R}^d$ for graph q,

$$\left(\{\boldsymbol{h}_v : v \in V(q)\}, \boldsymbol{h}_q\right) = \text{GNN}_{\boldsymbol{\theta}_{\text{GNN}}}(q) \tag{3}$$

Here, the node-level embeddings are the ones obtained at the last layer of the GNN. Similarly, it computes $\{\boldsymbol{h}_u : u \in V(g)\}$ and $\boldsymbol{h}_g \in \mathbb{R}^d$ for graph g with the same shared parameters $\boldsymbol{\theta}_{\text{GNN}}$.

Module-II is an embedding fusion module that fuses the embeddings obtained from q and g together into fused embedding vectors. Specifically, SimGNN first uses a Neural Tensor Network

[3]Note that, the original paper of SimGNN [5] listed four modules, and we combine their second and third modules into a single module.

Table 1. Summary of learning-based approaches for non-interpretable GED prediction

	Module-I	Module-II	Module-III	Other details
SimGNN	GCN	NTN, Histogram	MLP	
GraphSim	GCN	CNN	MLP	
GENN	GCN	NTN	MLP	Attention pooling
EGSC	GIN	Attention	MLP	Attention pooling
H2MN	HGNN	Concatenation	MLP	Attention pooling
TaGSim	GAL	NTN	MLP	Predict GEV
ERIC	GIN	NTN	MLP	Regularization in training
GREED	GIN	N/A	ℓ_2 distance	Pre- and Post-MLP

(NTN) [34] to fuse the two graph-level embeddings $\boldsymbol{h}_q$ and $\boldsymbol{h}_g$ into a single embedding vector, as follows

$$\begin{aligned}\boldsymbol{h}_{\text{NTN}} &= \text{NTN}_{\boldsymbol{\theta}_{\text{NTN}}}(\boldsymbol{h}_q, \boldsymbol{h}_g) \\ &= \text{RELU}\left(\left(\boldsymbol{h}_q^\top \boldsymbol{W}_1 \boldsymbol{h}_g \parallel \cdots \parallel \boldsymbol{h}_q^\top \boldsymbol{W}_c \boldsymbol{h}_g\right) + \boldsymbol{W}_{c+1}\left(\boldsymbol{h}_q \parallel \boldsymbol{h}_g\right)\right)\end{aligned}$$

where $\boldsymbol{W}_1, \ldots, \boldsymbol{W}_c \in \mathbb{R}^{d \times d}$, $\boldsymbol{W}_{c+1} \in \mathbb{R}^{c \times 2d}$ are learnable parameters, $\parallel$ denotes vector concatenation, and $\text{RELU}(\cdot)$ is an element-wise activation function.[4] Here, we summarize all the parameters $\{\boldsymbol{W}_1, \ldots, \boldsymbol{W}_c, \boldsymbol{W}_{c+1}\}$ that are used in the NTN into $\boldsymbol{\theta}_{\text{NTN}}$, and $\boldsymbol{h}_{\text{NTN}} \in \mathbb{R}^c$ where c is a hyper-parameter. In this module, SimGNN also computes a histogram vector $\boldsymbol{h}_{\text{HIST}}$ from a similarity matrix $\boldsymbol{S}$ that is defined based on the node-level embeddings of q and g, i.e., $S_{vu} = \text{sigmoid}(\boldsymbol{h}_v^\top \boldsymbol{h}_u)$; note that, no learnable parameter is used in computing $\boldsymbol{h}_{\text{HIST}}$.

Module-III serves as a prediction head that estimates the GED value using the fused embedding vectors produced by Module-II. Specifically, SimGNN adopts a Multilayer Perceptron (MLP) to predict the similarity between q and g based on $\boldsymbol{h}_{\text{NTN}}$ and $\boldsymbol{h}_{\text{HIST}}$, as

$$\widehat{\text{sim}}(q, g) = \text{MLP}_{\boldsymbol{\theta}_{\text{MLP}}}(\boldsymbol{h}_{\text{NTN}}, \boldsymbol{h}_{\text{HIST}}) \tag{4}$$

Given a set $\mathcal{D}$ of training graph pairs with their corresponding ground-truth GED values, the neural network model of SimGNN is trained by minimizing the following loss function

$$\mathcal{L} = \frac{1}{|\mathcal{D}|} \sum_{(q,g) \in \mathcal{D}} \left(\widehat{\text{sim}}(q, g) - \text{sim}(q, g)\right)^2 \tag{5}$$

where

$$\text{sim}(q, g) = \exp\left(-\frac{2 \cdot \text{ged}(q, g)}{|V(q)| + |V(g)|}\right) \tag{6}$$

is the similarity between q and g that is obtained from the ground-truth distance $\text{ged}(q, g)$. Equivalently, the predicted GED value is

$$\widehat{\text{ged}}(q, g) = -\frac{|V(q)| + |V(g)|}{2} \ln \widehat{\text{sim}}(q, g) \tag{7}$$

Other Models. Several other models have been proposed following SimGNN, aiming to improve the prediction accuracy, e.g., GraphSim [6], GENN [36], EGSC [26], H2MN [42], TaGSim [4], ERIC [43] and GREED [27]. They follow the same general framework illustrated in Figure 4, though with a few subtle distinctions, as summarized in Table 1.

[4]For simplicity, we omit all learnable bias terms from equations in the paper.

GraphSim [6] is similar to SimGNN, but replaces NTN in Module-II with a Convolutional Neural Network (CNN) that is originally proposed for image processing. Specifically, GraphSim treats the similarity matrix $\boldsymbol{S}$, that is constructed by SimGNN for computing the histogram vector $\boldsymbol{h}_{\text{HIST}}$, as an image. Also, additional techniques are proposed in [6] for the matrix/image construction, including utilizing breadth-first-search to order nodes, and using max padding and matrix resizing to obtain a squared matrix of a fixed size.

GENN [36] also closely resembles SimGNN, with its main advancement being the introduction of a new graph pooling operation (i.e., for Equation (2)) using an attention mechanism. Specifically, for the graph q, it first obtains a global context $\boldsymbol{c}_q \in \mathbb{R}^d$ by

$$\boldsymbol{c}_q = \tanh\left(\frac{1}{|V(q)|}\boldsymbol{W}_{c,q}\sum_{v\in V(q)}\boldsymbol{h}_v\right) \tag{8}$$

where $\boldsymbol{h}_v$ is the embedding of node v obtained from the last layer of the GNN, and $\boldsymbol{W}_{c,q} \in \mathbb{R}^{d\times d}$ is a learnable parameter matrix. Then, it computes the graph-level embedding of q as

$$\boldsymbol{h}_q = \sum_{v\in V(q)} \text{sigmoid}\Big((\boldsymbol{h}_v^\top \boldsymbol{c}_q)\cdot\alpha\Big)\boldsymbol{h}_v \tag{9}$$

where $\alpha = 10$ is a scaling factor. The graph level-embedding of g, $\boldsymbol{h}_g$, is obtained similarly. Note that, a variant of GENN, referred to as GENN-A*, was also proposed in [36]; it produces an interpretable GED prediction, which will be further discussed in Section 4.3.

EGSC [26] uses GIN in Module-I and proposes a graph pooling operation similar to GENN, specifically, the same Equation (8) and Equation (9) but with $\alpha = 1$. Furthermore, EGSC computes a graph-level embedding for q and g at each layer of the GNN based on node embeddings obtained at that layer; that is, it computes $\{\boldsymbol{h}_q^{(k)} : 1 \le k \le K\}$ and $\{\boldsymbol{h}_g^{(k)} : 1 \le k \le K\}$. In Module-II, EGSC first proposes an attention mechanism to fuse the two graph-level embeddings $\boldsymbol{h}_q^{(k)}$ and $\boldsymbol{h}_g^{(k)}$ into a fused embedding vector $\boldsymbol{h}_{q,g}^{(k)}$, for each $k \in \{1,\ldots,K\}$. It then uses the attention mechanism again to fuse these K embedding vectors $\{\boldsymbol{h}_{q,g}^{(k)} : 1 \le k \le K\}$ into a single embedding vector, which is fed to Module-III for GED prediction.[5]

H2MN [42] shares similar high-level idea as EGSC, but with two main differences. Firstly, H2MN computes node embeddings by running a Hypergraph Neural Network (HGNN) on two hypergraphs that are constructed from q and g, respectively; here, the hyperedges are created via either random walk or k-hop neighborhood. Similar to EGSC, an attention mechanism is proposed to pool the node embeddings into graph embeddings, at each layer of the HGNN. Secondly, in Module-II, H2MN simply concatenates the graph embeddings of q and g, obtained at all layers of the HGNN, to obtain a single embedding vector that is fed to Module-III.

TaGSim [4] estimates the graph edit vector (GEV), instead of the single GED value as done in other approaches. Specifically, it groups the edit operations into four types: NR (node relabeling), NID (node insertion/deletion), ER (edge relabeling), and EID (edge insertion/deletion). The GEV between q and g is a quadruple,

$$\text{GEV}(q,g) = (\#\text{NR}, \#\text{NID}, \#\text{ER}, \#\text{EID}) \tag{10}$$

which are the number of corresponding type of edit operations used in an optimal (i.e., shortest) edit path; the GED then is simply a sum of these four values. TaGSim uses a separate model, similar to SimGNN, to estimate each of these four values in the GEV. Another distinctive feature of TaGSim

[5]Note that, here describes the teacher model of EGSC, and besides the teacher model, a student model with fewer parameters is also proposed in [26] aiming to improve the efficiency while maintaining the prediction accuracy through knowledge distillation. However, as shown in [26], the teacher model usually achieves a better accuracy. Thus, we only consider the teacher model in this paper, and refer to it as EGSC.

is that its Module-I is a simple Graph Aggregation Layer (GAL), which can be considered as a GNN without non-linearity nor learnable parameters, similar to [37].

ERIC [43] is similar to SimGNN, but uses GIN in Module-I and computes a graph-level embedding for q and g at each layer of the GNN (similar to EGSC). The graph embedding $\boldsymbol{h}_q$ of q (and similarly $\boldsymbol{h}_g$) is the concatenation of those obtained at all layers. Then, the similarity between q and g is predicted as

$$\alpha \cdot \text{MLP}_{\theta_{\text{MLP}}}(\text{NTN}_{\theta_{\text{NTN}}}(\boldsymbol{h}_q, \boldsymbol{h}_g)) + \beta \cdot \text{MLP}_{\theta'_{\text{MLP}}}(\exp(-\|\boldsymbol{h}_q - \boldsymbol{h}_g\|_2)) \tag{11}$$

which has the additional second term compared to SimGNN; here, α and β are trainable scalars. In addition, ERIC adds the following alignment regularization term to its loss function for model training.

$$\mathcal{L}_{\text{AReg}} = \frac{1}{K}\sum_{k=1}^{K}\left(\gamma_q^{(k)} + \gamma_g^{(k)} + \|\gamma_q^{(k)} - \gamma_g^{(k)}\|_2\right) \tag{12}$$

where $\gamma_q^{(k)} = \sum_{v \in V(q)} \|\text{DIST}(\boldsymbol{h}_v^{(k)}, \boldsymbol{h}_q^{(k)}) - \text{DIST}(\boldsymbol{h}_v^{(k)}, \boldsymbol{h}_g^{(k)})\|_2$ and $\gamma_g^{(k)}$ is defined similarly; here, $\boldsymbol{h}_v^{(k)}$ and $\boldsymbol{h}_q^{(k)}$ are, respectively, the embeddings of node v and graph q computed at layer k, and $\text{DIST}(\cdot,\cdot)$ is a distance function.

GREED [27] does not have Module-II, and directly predicts the GED between q and g as the ℓ_2 (i.e., Euclidean) distance between the graph-level embeddings of q and g obtained from Module-I. GREED's Module-I is similar to that of ERIC, but additionally applies a pre-MLP (i.e., before GIN) for transforming the input node features and a post-MLP (i.e., after the graph pooling operation of GIN) for transforming the graph embeddings.

OMC [24] formulates GED prediction through the lens of graph matching, and does not follow the framework of Figure 4. For each $v_i \in V(q)$ and $u_j \in V(g)$, let $X_{i,j} \in \{0,1\}$ be the indicating variable of whether v_i maps to u_j. Assuming $|V(q)| = |V(g)|$, the GED between q and g is estimated via a quadratic function as

$$\sum_{\substack{v_i \in V(q)\\ u_j \in V(g)}} c_{i,j} X_{i,j} + \sum_{\substack{v_i, v_{i'} \in V(q)\\ u_j, u_{j'} \in V(g)}} c_{i,i',j,j'} X_{i,j} X_{i',j'} \tag{13}$$

where $c_{i,j} \in \{0,1\}$ and $c_{i,i',j,j'} \in \{0,1\}$ are computed based on q and g as follows: $c_{i,j} = 1$ if $l(v_i) \neq l(u_j)$ and $c_{i,j} = 0$ otherwise; $c_{i,i',j,j'} = 1$ if $l(v_i, v_{i'}) \neq l(u_j, u_{j'})$ and $c_{i,i',j,j'} = 0$ otherwise; here, non-existing edges are assumed to have a special label $\perp \notin (\Sigma \cup \dashv)$. OMC relaxes $X_{i,j}$ to be in the range $[0,1]$ and computes it via a GNN that is run on an association graph constructed from q and g as follows: the association graph has a node (v_i, u_j) (corresponding to $X_{i,j}$) for each $v_i \in V(q)$ and $u_j \in V(g)$, and an edge between node (v_i, u_j) and node $(v_{i'}, u_{j'})$ if $c_{i,i',j,j'} = 1$. Then, $X_{i,j}$ is computed by running an MLP on the learned embedding of node (v_i, u_j). To train the model, OMC defines a loss function that considers (1) the difference between the estimated GED and the ground-truth GED, (2) a regularization term trying to enforce one-to-one mapping for X, and (3) a regularization term for a partial node mapping that is constructed based on a given ground-truth optimal node mapping.

4.3 Interpretable GED Prediction

Learning-based approaches for interpretable GED prediction outputs an edit path (more precisely, a node mapping) along with its corresponding editorial cost as the predicted GED value (see Section 3.1 for the definition of editorial cost). Since GED is equal to the minimum editorial cost among all possible node mappings, the predicted value then will always be no smaller than the true GED value. To find a good node mapping, the existing approaches for interpretable GED prediction

typically work together with a combinatorial algorithm and can be divided into two categories: tree traversal-based and cost matrix-based.

Tree Traversal-based Approaches. Approaches in this category conduct a traversal over a search tree (i.e., following the ideas in Section 3.2), and invoke a neural network to predict the lower bound lb_f of a partial node mapping f (see Definition 3.2) during the tree traversal. For example, if A* search is used, then the neural network is invoked at Line 8 of Algorithm 1 to predict the lower bound $\mathsf{lb}_{f'}$. The neural networks used in these approaches also generally follow the framework in Figure 4.

GENN-A* [36], which extends GENN (discussed in Section 4.2) to interpretable GED prediction, belongs to this category and uses A* search (i.e., Algorithm 1) for tree traversal. For a partial mapping f, the predicted lower bound for lb_f is computed as

$$\widehat{\mathsf{lb}}_f = \mathsf{edc}_f(q_f, g_f) + \mathsf{heu}(q_{\bar{f}}, g_{\bar{f}}) \tag{14}$$

where q_f (resp. g_f) denotes the subgraph of q (resp. g) induced by its vertices that are in f, and $q_{\bar{f}}$ (resp. $g_{\bar{f}}$) denotes the part of q (resp. g) that is not in q_f (resp. g_f). GENN-A* uses a neural network, same as GENN but with a different training process, to predict $\mathsf{heu}(q_{\bar{f}}, g_{\bar{f}})$ based on Equations (4) and (7). As the predicted $\widehat{\mathsf{lb}}_f$ is not guaranteed to be a lower bound of f, GENN-A* does not compute the exact GED. Nevertheless, the estimated GED is the editorial cost of a full node mapping (see Line 5 of Algorithm 1), and thus is interpretable.

Noah [39] is another approach belonging to this category. It shares similar high-level ideas as GENN-A*, but with two major differences. Firstly, it proposes a different model for computing the graph-level embeddings of q and g (and also subgraphs of q and g), i.e., Module-I in Figure 4. For example, when conducting message passing on one graph, it takes into account the hidden representations of nodes in the other graph. Secondly, besides the lower bound $\widehat{\mathsf{lb}}_f$ (more precisely, $\mathsf{heu}(q_{\bar{f}}, g_{\bar{f}})$ in Equation (14)), it also predicts a beam size b such that only the top-b children (i.e., partial mappings generated at Lines 6–8 of Algorithm 1) are pushed in the priority queue $\boldsymbol{Q}$, with other children being pruned. The resulting tree traversal algorithm is referred to as A*-BeamSearch.

Cost Matrix-based Approaches. Approaches in this category follow the ideas in Section 3.1, by directly extracting node mapping from some matrices. They typically first predict or compute a cost matrix $\boldsymbol{C}$ and a matching matrix $\boldsymbol{M}$ between vertices of q and vertices of g, and then obtain a predicted GED value based on $\boldsymbol{C}$ and $\boldsymbol{M}$. Note that, $\boldsymbol{C}$ is typically predicted while $\boldsymbol{M}$ could be either predicted or computed.

GOTSim [14] falls into this category. Given a GNN that outputs node embeddings $\{\boldsymbol{h}_v^{(k)} : v \in V(q), 1 \le k \le K\}$ and $\{\boldsymbol{h}_u^{(k)} : u \in V(g), 1 \le k \le K\}$, it computes a cost matrix $\boldsymbol{C}^{(k)}$ at each of the K layers. Specifically, the cost between node v of q and node u of g at layer k, $C_{v,u}^{(k)}$, is computed as the cosine distance between $\boldsymbol{h}_v^{(k)}$ and $\boldsymbol{h}_u^{(k)}$. Then, a minimum-cost matching is computed based on $\boldsymbol{C}^{(k)}$ by treating $\boldsymbol{C}^{(k)}$ as the edge weights of a bipartite graph (see Figure 5) and running the Hungarian algorithm [22], and a matching matrix $\boldsymbol{M}^{(k)}$ is constructed based on the minimum-cost matching, where $M_{v,u}^{(k)} = 1$ is v maps to u in the computed minimum-cost matching and $M_{v,u}^{(k)} = 0$ otherwise. Finally, the GED between q and g is estimated as

$$\frac{1}{K}\sum_{k=1}^{K} \boldsymbol{M}^{(k)} \circ \boldsymbol{C}^{(k)} \tag{15}$$

where $\circ$ means element-wise multiplication followed by a sum of all the elements; that is, $\boldsymbol{M}^{(k)} \circ \boldsymbol{C}^{(k)}$ is the cost of the minimum-cost matching. Note that, the GED estimated by Equation (15) is used

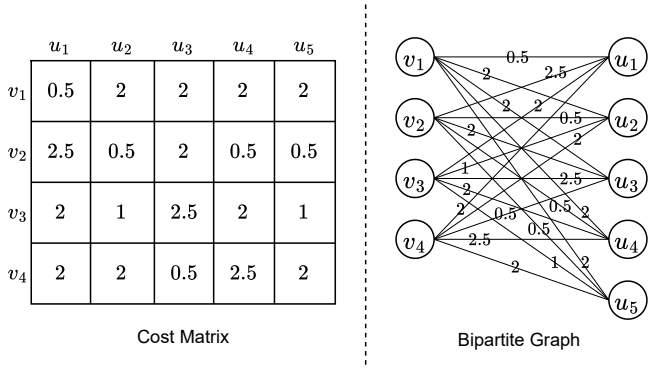


Fig. 5. Cost matrix and its corresponding bipartite graph

in both model training and in non-interpretable GED prediction. To make interpretable GED prediction, $\boldsymbol{M}^{(K)}$ is outputted as the node mapping and its editorial cost is the predicted GED value.

GEDGNN [25] is the most recent approach belonging to this category. It first adopts a GIN to obtain both node-level embeddings and graph-level embeddings of q and g as in Equation (3), and then uses the node-level embeddings to learn both a cost matrix $\boldsymbol{C}$ and a matching matrix $\boldsymbol{M}$. Specifically, the cost matrix $\boldsymbol{C}$ is learned following the idea of NTN [34], i.e.,

$$C_{v,u} = \text{MLP}\Big(\boldsymbol{h}_v^\top \boldsymbol{W}_1 \boldsymbol{h}_u \,\|\, \cdots \,\|\, \boldsymbol{h}_v^\top \boldsymbol{W}_c \boldsymbol{h}_u\Big) \in \mathbb{R} \tag{16}$$

The matching matrix $\boldsymbol{M}$ is learned similarly. Note that, all the NTNs used in learning the values of $\boldsymbol{C}$ share the same set of parameters, while all the NTNs used in learning the values of $\boldsymbol{M}$ share another set of parameters. A normalized GED between q and g, denoted $\text{ged}_\text{n}(q,g)$, is estimated as

$$\widehat{\text{ged}}_\text{n}(q,g) = \text{sigmoid}\big(\text{softmax}(\boldsymbol{M}) \circ \boldsymbol{C} + \text{MLP}(\text{NTN}(\boldsymbol{h}_q, \boldsymbol{h}_g))\big) \tag{17}$$

where the softmax function normalizes each row of $\boldsymbol{M}$ into a probability distribution, and $\text{NTN}(\boldsymbol{h}_q, \boldsymbol{h}_g)$ is another NTN that is the same as the one used in SimGNN. In training the model, it aims to minimize the loss function

$$\mathcal{L} = \text{BCELoss}(\boldsymbol{M}, \boldsymbol{M}^*) + \eta \cdot \Big(\widehat{\text{ged}}_\text{n}(q,g) - \text{ged}_\text{n}(q,g)\Big)^2 \tag{18}$$

where $\boldsymbol{M}^*$ is a ground-truth node mapping (corresponding to the GED), BCELoss is the binary cross-entropy loss, η is a hyper-parameter, and $\text{ged}_\text{n}(q,g)$ is a normalized version of $\text{ged}(q,g)$ computed as

$$\text{ged}_\text{n}(q,g) = \frac{\text{ged}(q,g)}{\max(|V(q)|,|V(g)|) + \max(|E(q)|,|E(g)|)} \tag{19}$$

Note that, $\text{ged}_\text{n}(q,g)$ is likely, but not formally guaranteed, to be between 0 and 1. To make interpretable GED prediction, GEDGNN post-processes the learned matching matrix $\boldsymbol{M}$ to extract a good node mapping. Specifically, it first extracts top-k best matchings [13] by viewing the values of $\boldsymbol{M}$ as edge weights of a bipartite graph, and then returns the one that has the smallest editorial cost.

5 Non-learning-based Approaches for GED Estimation

In this section, we first review the existing non-learning-based heuristic algorithms for GED estimation, and then present a new heuristic algorithm based on the idea of anytime algorithm.

5.1 Existing Heuristic Approaches

As summarized in a recent survey about non-learning-based heuristic approaches for GED estimation [8], most of the existing heuristic approaches are instantiations of one of three paradigms: linear sum assignment problem with error correction (LSAPE), linear programming, and local search. Among them, linear programming-based approaches tend to be extremely slow as shown in [8], while local search approaches are generally applicable for improving the estimation accuracy based on any given node mapping (e.g., those computed by the LSAPE-based approaches); thus, we omit their discussions here, interested readers can refer to [8]. LSAPE-based approaches share similarities with the cost matrix-based learning method GOTSim discussed in Section 4.3 but rely on a non-learning-based strategy to construct the cost matrix $\boldsymbol{C}$. Specifically, $C_{v,u}$ is calculated based on the local substructures surrounding v and u. The editorial cost of the minimum-cost node mapping that is extracted from $\boldsymbol{C}$ is returned as an upper bound of the GED. In addition, when $\boldsymbol{C}$ satisfies certain conditions, the value of Equation (15) with $K = 1$ is a lower bound of the GED. Different LSAPE-based approaches typically only differ in their ways of constructing the cost matrix $\boldsymbol{C}$. For example, Node [20] only considers the labels of v and u, and defines $C_{v,u} = 1$ if $l(v) \neq l(u)$ and $C_{v,u} = 0$ otherwise.[6] BP [28] additionally considers the edges adjacent to v and u, e.g., it adds $\big||N(v)| - |N(u)|\big|$ to the cost for the case of no edge labels. FBP [32] and SFBP [33] enhance BP by reducing the size of $\boldsymbol{C}$, thereby slightly improving the efficiency. Later approaches consider more complex substructures surrounding v and u, e.g., walks [16] and subgraphs [10]. The state of the art along this line is SDTED [7] that considers compressed neighborhood tree structures.

5.2 Anytime Algorithms

Another paradigm which is not considered in [8] is anytime algorithms, which quickly provide the first solution (i.e., GED estimation) to the problem, then find a list of improved solutions and eventually converge to the optimal solution.[7] Due to the last characteristic, anytime algorithms are usually obtained from exact algorithms. Specifically, for GED computation, any exact algorithm that traverses the search tree (see Figure 3) in a depth-first-search (DFS) manner is anytime in nature [2], because it visits full node mappings one-by-one at the search tree leafs. In contrast, A* search algorithms are not anytime by nature, because they stop immediately after finding the first full node mapping which is guaranteed to be optimal. Although there are techniques to convert A* search algorithms into anytime A* algorithms [2], such conversions require hyper-parameter tuning and have not been studied in the context of GED computation. Nevertheless, we observe that the A* search algorithm AStar-BMao of [12] itself is an anytime algorithm due to its way of computing the lower bound lb_f, and thus does not require conversion. As A* search has a smaller search space than DFS search for exact GED computation [11], anytime A* is expected to provide more accurate GED estimation than anytime DFS under the same time constraint; this is confirmed by our experiments in Section 6. In this section, we provide the details of anytime A* and analyze its complexities and properties, for completeness.

Given an input integer t that is used for controlling the time and space complexity, the general idea of our anytime A* algorithm is that we traverse the search tree of Figure 3 in a best-first-search manner and stop after visiting t search states (i.e., partial node mappings). For each visited partial node mapping, multiple full node mapping are generated. The smallest editorial cost among these generated full node mappings is returned as the estimated GED value, which is thus interpretable. The pseudocode of our anytime A* algorithm for GED estimation is shown in Algorithm 2, denoted

[6] Note that, here we simplified the discussion for the case of uniform edit cost.

[7] The last characteristic distinguishes anytime algorithms from local search algorithms, as the latter is not guaranteed to find the optimal solution. Also, techniques used in the two kinds of algorithms are different.

Algorithm 2: App-BMao(q, g, t)

Input: Two graphs q and g with $|V(q)| \leq |V(g)|$, and a positive integer t used for controlling the time and space complexity

Output: An approximation $\widehat{\text{ged}}(q, g)$ of $\text{ged}(q, g)$

1 Compute a matching order $\pi = (v_1, \ldots, v_{|V(q)|})$ of $V(q)$;
2 $\mathcal{S} \leftarrow \{(0, \emptyset)\}$; `/* Initialize a Set data structure for storing node mappings */`;
3 $f^* \leftarrow \emptyset$; $\widehat{\text{ged}}(q, g) \leftarrow +\infty$;
4 **for** $i \leftarrow 1$ ***to*** t **do**
5 $\quad (\text{lb}_f, f) \leftarrow$ the entry with the smallest lower bound in $\mathcal{S}$;
6 $\quad$ Remove (lb_f, f) from $\mathcal{S}$;
$\quad$ `/* The following lines generate all children of` f `*/`
7 $\quad$ **for each** *node* $v \in V(q_{\bar{f}})$ **do**
8 $\quad\quad$ **for each** *node* $u \in V(g_{\bar{f}})$ **do**
9 $\quad\quad\quad \lambda_f(v, u) \leftarrow$ the cost of mapping v to u regarding f;
10 $\quad$ **for each** $j \leftarrow 1$ ***to*** $|V(g_{\bar{f}})|$ **do**
11 $\quad\quad \sigma^* \leftarrow$ the minimum-cost perfect matching between $V(q_{\bar{f}})$ and $V(g_{\bar{f}})$ based on costs $\lambda_f(\cdot, \cdot)$;
12 $\quad\quad$ **if** $\text{edc}_{f \cup \sigma^*}(q, g) < \widehat{\text{ged}}(q, g)$ **then**
13 $\quad\quad\quad \widehat{\text{ged}}(q, g) \leftarrow \text{edc}_{f \cup \sigma^*}(q, g)$; $f^* \leftarrow f \cup \sigma^*$;
14 $\quad\quad u \leftarrow$ the node to which $v_{|f|+1}$ maps in σ^*;
15 $\quad\quad f' \leftarrow f \cup \{v_{|f|+1} \mapsto u\}$;
16 $\quad\quad \text{lb}_{f'} \leftarrow \text{edc}_f(q_f, g_f) + \sum_{v \in V(q_{\bar{f}})} \lambda_f(v, \sigma^*(v))$;
17 $\quad\quad$ **if** $\text{lb}_{f'} \geq \widehat{\text{ged}}(q, g)$ **then break**;
18 $\quad\quad$ Insert $(\text{lb}_{f'}, f')$ into $\mathcal{S}$;
19 $\quad\quad$ **if** $|\mathcal{S}| > t$ **then** Remove the entry with the largest lower bound from $\mathcal{S}$;
20 $\quad\quad \lambda_f(v_{|f|+1}, u) \leftarrow +\infty$;
21 **return** $\widehat{\text{ged}}(q, g)$;

App-BMao. We use a Set data structure $\mathcal{S}$ to store the set of candidate partial mappings to be visited. Line 1 computes a matching order π. Line 2 initializes $\mathcal{S}$ with the empty mapping, i.e., the root of the search tree. f^* stores the current best full node mapping (in terms of editorial cost), and $\widehat{\text{ged}}(q, g)$ stores the editorial cost of f^* (Line 3). Then, we iteratively visit t partial mappings of the search tree (Line 4) by popping the partial mapping with smallest lower bound from $\mathcal{S}$ (Lines 5–6). When visiting a partial mapping f, we generate up-to $|V(g)|$ full mappings (Line 10) by extending f (Lines 11–13) at the same time as we create f's children in the search tree and push them into $\mathcal{S}$ (Lines 14–18). To control the space complexity, we limit the cardinality of $\mathcal{S}$ to be at most t, by popping out partial mappings with the largest lower bounds (Line 19). In extending f to full node mappings, we construct a cost matrix between $V(q_{\bar{f}})$ and $V(g_{\bar{f}})$ — see the paragraph following Equation (14) for the definitions of q_f, g_f, $q_{\bar{f}}$ and $g_{\bar{f}}$ — such that the cost of mapping $v \in V(q_{\bar{f}})$ to $u \in V(g_{\bar{f}})$ is

$$\lambda_f(v, u) = \mathbb{1}_{l(v) \neq l(u)} + \tfrac{1}{2} \times \Upsilon\big(L_f(v), L_f(u)\big) + \textstyle\sum_{v' \in V(q_f)} \mathbb{1}_{l(v,v') \neq l(u, f(v'))} \tag{20}$$

where $\mathbb{1}_\phi$ is an indicator function that equals 1 if the expression ϕ evaluates true and 0 otherwise. $L_f(v)$ is the multi-set of edge labels between v and vertices of $q_{\bar{f}}$; $L_f(u)$ is defined similarly. $\Upsilon(S_1, S_2)$ is the distance between two multi-set, defined as $\max\{|S_1|, |S_2|\}$ minus the number of common

elements in S_1 and S_2. The second term in Equation (20) measures the difference between the adjacent edge labels of v in $q_{\bar{f}}$ and the edge labels of u in $q_{\bar{f}}$; as each such edge is considered twice, once at each end-point, there comes the coefficient $\frac{1}{2}$. The third term measures the difference between the adjacent edge labels of v in q_f and the edge labels of u in g_f. For example, Figure 5 shows the constructed matrix for $f = \emptyset$. It is proved in [12] that $\text{edc}_f(q_f, g_f)$ plus the cost of the minimum-cost matching is a valid lower bound of f.

Theorem 5.1. *The time complexity and space complexity of* App-BMao *are* $O\Big(t\times\big(|V(q)|+|V(g)|\big)^3\Big)$ *and* $O\big(t + |V(q)| \times |V(g)|\big)$*, respectively.*

Proof. The time complexity is mainly dominated by the time of constructing the cost matrix (Lines 7–8), and computing the $|V(g_{\bar{f}})|$ minimum-cost perfect matchings (Lines 10,11,20). It is shown in [12] that both can be conducted in $O(|V(q)| + |V(g)|)^3$ time. Thus, the time complexity of App-BMao follows.

The space complexity of App-BMao follows from the facts that (1) the t popped mappings (Line 6) and the Set data structure $\mathcal{S}$ take $O(t)$ space in total, and (2) the cost matrix takes $O(|V(q)|\times|V(g)|)$ space. □

From Theorem 5.1, we can see that both the time complexity and space complexity of App-BMao increase with t. We prove in the lemma below that the estimation accuracy also increases with t.

Lemma 5.2. *When t increases, the returned estimation $\widehat{\text{ged}}(q, g)$ will be non-increasing.*

Proof. This can be easily seen from the fact that all partial mappings that are visited for a smaller t will also be visited when t increases. □

Moreover, under certain conditions (as illustrated in the lemma below), we can certify that the returned estimation $\widehat{\text{ged}}(q, g)$ actually is the exact GED.

Lemma 5.3. *If the returned estimation $\widehat{\text{ged}}(q, g)$ is no larger than the lower bounds of all mappings in $\mathcal{S}$ when the algorithm terminates, then $\widehat{\text{ged}}(q, g) = \text{ged}(q, g)$.*

Proof. It is easy to see that the lemma holds when no partial mappings were ever removed from $\mathcal{S}$ at Line 19. For simplicity, let $\mathcal{S}_i$ be the version of $\mathcal{S}$ after the i-th iteration of the "for loop" at Line 4, and assume that $|\mathcal{S}_1| \geq t$. Then, $|\mathcal{S}_i| \geq t-i+1$ for $1 \leq i \leq t$. Let $key(\mathcal{S}_i, j)$ be the key value of the j-th entry in $\mathcal{S}_i$ ranked based on their keys. Then, $key(S_1, t) \geq key(S_2, t-1) \geq \cdots \geq key(S_t, 1)$. As a result, all those mappings that are removed from $\mathcal{S}$ at Line 19 will have lower bounds at least $key(S_t, 1)$. Thus, the lemma holds. □

Remarks. Technically speaking, App-BMao is not an anytime algorithm since it may not find the exact GED. But when the input parameter t is sufficiently large, App-BMao works the same way as AStar-BMao and obtains the exact GED. The main differences between App-BMao and AStar-BMao are Lines 4 and 19 of Algorithm 2. Line 4 makes App-BMao run in polynomial time and space depending on t; recall that, AStar-BMao has an exponential time and space complexity. Without Line 19, App-BMao is still correct, but its space complexity becomes $O\big((t+|V(q)|)\times|V(g)|\big)$ which is $|V(g)|$ times larger when t is sufficiently large.

6 Experiments

In this section, we empirically evaluate the existing learning-based approaches against each other, as well as against our new heuristic algorithm App-BMao. Through the experiments, we aim to answer the following research questions.

RQ1. For non-interpretable GED prediction, which learning-based approach achieves the best accuracy?
RQ2. Between the two frameworks for interpretable GED prediction (i.e., tree traversal-based v.s. cost matrix-based), which framework is more promising?
RQ3. How do the learning-based approaches perform compared with our new heuristic algorithm App-BMao?
RQ4. We found that the ground-truth GED values provided by `torch_geometric.datasets` for the IMDB dataset are not exact GED values. Will and how does this affect the model training and evaluation?

6.1 Experimental Settings

Compared Approaches. In the experiments, we compare the following approaches.

- For learning-based approaches for ***non-interpretable GED prediction***, we include SimGNN [5], TaGSim [4], EGSC [26], GENN [36], GREED [27], ERIC [43] and GEDGNN-v [25], where the last one is a variant of GEDGNN that uses Equation (17) for predicting the normalized GED.
- For learning-based approaches for ***interpretable GED prediction***, we include GENN-A* [36] and GEDGNN [25], one from each category of Section 4.3.
- For non-learning-based heuristic algorithms, we include the state-of-the-art LSAPE-based approach SDTED [7], where the weight parameter is set as 0.5 as suggested in the paper.
- For the new heuristic algorithm App-BMao, we choose values for t from $\{10, 30, 100\}$ and refer to the corresponding algorithms as App-BMao$_{10}$, App-BMao$_{30}$, App-BMao$_{100}$, respectively.
- In addition, we also include AStar-BMao [12] and modified its DFS version in the same way as App-BMao for $t = 100$ and $t = 200$, denoted DFS-BMao$_{100}$ and DFS-BMao$_{200}$, respectively.

Other learning-based approaches from Section 4 are omitted here, because their code either is incomplete or requires older packages that are incompatible with our CUDA installation. The source code of App-BMao is available at https://lijunchang.github.io/App-BMao, and other source code used in conducting the experiments is available at https://github.com/XMYMouyiXu/GED-EXP.

Table 2. Statistics of datasets

		$\|V\|$		$\|E\|$		
	#Graphs	Avg	Max	Avg	Max	#Distinct node labels
AIDS	700	8.9	10	8.8	14	29
LINUX	1000	7.6	10	6.9	13	1
IMDB	1500	13.0	89	65.9	1467	1

Datasets. We evaluate the approaches on three commonly used datasets, AIDS700nef (abbreviated as AIDS), LINUX and IMDBMulti (abbreviated as IMDB). All datasets are downloaded from `GEDDataset` of `torch_geometric` [1], which were originally prepared by the authors of SimGNN [5] and shared in a Google drive folder. Statistic of the datasets are shown in Table 2. We remark that none of these datasets come with edge labels, but at least the original AIDS dataset has edge labels (e.g., see [17]); we suspect that the absence of edge labels is mainly because GNNs at the time of [5], although can handle edge labels, does not do it well, and thus edge labels were ignored. As a result, none of the existing learning-based approaches, except TaGSim, explicitly support edge labels; note that, our new heuristic algorithm App-BMao supports edge labels.

Training and Testing Graph Pairs. The downloaded datasets also include train-test splitting (with a ratio of 4 : 1) and ground-truth GED values.[8] In particular, for each dataset, its set of graphs was split and stored into two folders, corresponding to training graphs and testing graphs. Then, ground-truth GED values are provided for all train-train graph pairs and all train-test graph pairs. As a result, the set of train-train graph pairs forms the training graph pairs, and the set of train-test graph pairs forms the testing graph pairs. In our experiments, we use these training and testing graph pairs by default, except for TaGSim, GREED and GEDGNN, which are trained on their own data provided in the source codes.

During the experiments, we observed that for the IMDB dataset, some of the provided ground-truth GED values are not exact GED values, which could be either smaller or larger than the exact GED value, especially when a graph contains more than 10 nodes. In view of this, we regenerated the ground-truth GED values for the IMDB dataset using the exact GED computation algorithm of [12], and use IMDB_{pyg} to refer to the original downloaded dataset while using IMDB to refer to the dataset with the regenerated correct ground-truth GED values. Remark that, due to the NP-hardness of computing the exact GED, we only generated ground-truth GED value for graph pairs (q, g) satisfying $|V(q)| - |V(g)| < 3$; other graph pairs are not included in either training or testing.

In addition, besides train-test graph pairs, we also generate test-test graph pairs to evaluate the models' performance on completely unseen data, which will be used in our Exp-7.

Metrics. We adopt six widely used metrics to measure and compare the performance of the different approaches.

- Mean square error (MSE)
- Mean absolute error (MAE)
- Spearman's rho
- Kendall's tau
- Precision@10
- Precision@20

MAE measures the absolute difference between the estimated GED value and the true GED value, i.e.,

$$\text{MAE}(q, g) = \left|\widehat{\text{ged}}(q, g) - \text{ged}(q, g)\right| \tag{21}$$

As there are two different normalizations of the GED value, as shown in Equation (6) and Equation (19), we consider two version of MSE, MSE_{S} and MSE_{D} that compare the similarity and normalized GED, respectively, as follows

$$\text{MSE}_{\text{S}}(q, g) = \left(\widehat{\text{sim}}(q, g) - \text{sim}(q, g)\right)^2 \tag{22}$$

$$\text{MSE}_{\text{D}}(q, g) = \left(\widehat{\text{ged}}_{\text{n}}(q, g) - \text{ged}_{\text{n}}(q, g)\right)^2 \tag{23}$$

Note that, we can easily convert between $\text{ged}(q, g)$, $\text{sim}(q, g)$ and $\text{ged}_{\text{n}}(q, g)$ as well as for their estimated versions by using Equation (6) and Equation (19). For MAE and MSE, the smaller the value, the better the estimation accuracy.

Spearman's rho (ρ) and Kendall's tau (τ) are two correlation coefficients that measure the correlation between the ranking of the graph pairs based on the predicted GED values and the ranking based on the ground-truth GED values. The last two metrics are the precision for top-10 and top-20 ranking graph pairs. For these four metrics, the larger the value, the better the accuracy.

[8]We downloaded all the files when we started the project. But the files storing the ground-truth values are no longer available in the shared Google drive folder.

Training and Evaluation. For all compared learning-based approaches, we train the model on the training graph pairs using the default hyper-parameters provided with the source code, and then evaluate it on the testing graph pairs. To get a more robust result, we run the train-and-evaluate process multiple times, and report the mean and standard deviation against each of the six metrics. The number of train-and-evaluate processes varies across the approaches, depending on how fast the process runs for an approach. Specifically, the numbers of repetitions for EGSC, GEDGNN, TaGSim, SimGNN, GREED, ERIC, GENN, GENN-A* are ten, five, five, four, three, three, two and two, respectively.

Hardware. The experiments are conducted on a machine with an Intel(R) Core(TM) i9-12900KF CPU, 128G memory and an NVIDIA GeForce RTX 4090 GPU. All learning models are trained by using the GPU, with the exception that GENN-A* uses both GPU and CPU in training. Then, all the evaluations are conducted on CPU.

6.2 Experimental Results

Exp-1: Evaluate learning-based approaches for non-interpretable GED prediction. We first evaluate the different learning-based approaches, SimGNN, TaGSim, EGSC, GREED, ERIC, GENN and GEDGNN-v, against each other for non-interpretable GED prediction. The results are shown in Figure 6. Note that, we do not have the results of TaGSim on IMDB, as we were unable to provide the correct testing data to TaGSim which requires its own proprietary format; recall that, IMDB is our regenerated dataset with correct ground-truth GED values. Nevertheless, TaGSim is outperformed by the other approaches on AIDS and LINUX. From Figure 6, we can see that there is no clear winner among them. GREED performs the best on AIDS, ERIC performs the best on LINUX, while EGSC and GENN outperform others on IMDB.

Exp-2: Evaluate learning-based approaches for interpretable GED prediction. We now compare GEDGNN with GENN-A*, both providing interpretable GED predictions but following different ideas. GENN-A* uses tree traversal and neural networks, while GEDGNN learns a cost matrix and a matching matrix and then runs a combinatorial algorithm as a post-processing to extract multiple candidate node mappings. The results are also shown in Figure 6. We can see that GENN-A* provides more accurate GED estimation than GEDGNN on AIDS and LINUX, while the latter performs better regarding the p@10 and p@20 metrics. GENN-A* runs out-of-memory (OOM) during training on IMDB. This is because (1) GENN-A* runs an A*-based exact GED computation algorithm, which has an exponential space complexity, on the training graph pairs during training, and (2) IMDB graphs are much larger than AIDS and LINUX. Also, as we will show later (i.e., in Exp-5 and shown in Table 3), GENN-A* takes significantly longer time in training and testing, also due to running the A* search algorithm. Thus, GENN-A* may produce a more accurate GED prediction but at the cost of significant more memory and time consumption, and GENN-A* ***work well on small graphs while*** GEDGNN ***is better on large graphs***. We remark that GEDGNN was not compared with GENN-A* in [25].

Exp-3: Interpretable v.s. non-interpretable GED prediction. When comparing interpretable GED prediction approaches with non-interpretable ones, there is no clear winner between GENN and its interpretable counterpart GENN-A*, and also no clear winner between GEDGNN-v and GEDGNN, in terms of prediction accuracy. Nevertheless, ***the non-interpretable GED prediction approaches*** EGSC***,*** GENN***,*** ERIC ***and*** GREED ***generally produce the most accurate GED prediction, at the cost of interpretability.***

Exp-4: Evaluate non-learning-based heuristics. We now evaluate the non-learning-based heuristics againt each other as well as against learning-based approaches. The results are also

	MSE_S ($\times 10^{-3}$) ↓	MSE_D ($\times 10^{-3}$) ↓	MAE ↓	ρ ↑	τ ↑	p@10 ↑	p@20 ↑
SimGNN	2.429(±0.092)	3.277(±0.25)	0.827(±0.049)	0.856(±0.002)	0.683(±0.003)	0.498(±0.007)	0.617(±0.001)
TaGSim	6.687(±0.785)	11.775(±1.134)	1.59(±0.097)	0.657(±0.026)	0.486(±0.021)	0.234(±0.017)	0.234(±0.017)
EGSC	1.543(±0.051)	2.38(±0.1)	0.654(±0.018)	0.897(±0.001)	0.734(±0.002)	0.652(±0.013)	0.733(±0.01)
GREED	**1.369(±0.048)**	1.898(±0.129)	**0.593(±0.017)**	**0.907(±0.001)**	**0.745(±0.001)**	**0.695(±0.009)**	**0.763(±0.004)**
ERIC	1.439(±0.039)	2.336(±0.117)	0.629(±0.013)	0.871(±0.001)	0.699(±0.001)	0.684(±0.008)	0.749(±0.001)
GENN	1.75(±0.107)	**1.16(±0.043)**	0.678(±0.019)	0.897(±0.005)	0.737(±0.007)	0.453(±0.006)	0.54(±0.007)
GEDGNN-v	2.48(±0.095)	4.402(±0.468)	0.896(±0.014)	0.865(±0.001)	0.731(±0.001)	0.568(±0.012)	0.632(±0.009)
GENN-A*	**1.258(±0.041)**	**1.23(±0.2)**	**0.507(±0.034)**	**0.94(±0.001)**	**0.823**	0.721(±0.055)	0.759(±0.043)
GEDGNN	6.031(±0.138)	12.843(±0.434)	1.438(±0.022)	0.806(±0.002)	0.702(±0.003)	**0.881(±0.008)**	**0.85(±0.006)**
SDTED	28.231	82.551	4.579	0.546	0.392	0.694	0.667
DFS-BMao$_{100}$	0.612	0.971	0.181	0.957	0.881	0.994	0.998
DFS-BMao$_{200}$	0.189	0.312	0.062	0.984	0.938	0.996	0.999
App-BMao$_{10}$	6.65	12.354	1.443	0.785	0.622	0.931	0.875
App-BMao$_{30}$	1.211	2.072	0.392	0.931	0.821	0.991	0.995
App-BMao$_{100}$	**0.104**	**0.183**	**0.052**	**0.989**	**0.941**	**0.999**	**1.0**

(a) AIDS

	MSE_S ($\times 10^{-3}$) ↓	MSE_D ($\times 10^{-3}$) ↓	MAE ↓	ρ ↑	τ ↑	p@10 ↑	p@20 ↑
SimGNN	1.638(±0.129)	1.058(±0.1)	0.326(±0.008)	0.949(±0.002)	0.818(±0.002)	0.91(±0.031)	0.909(±0.056)
TaGSim	5.891(±0.282)	3.716(±0.196)	0.535(±0.031)	0.924(±0.007)	0.779(±0.01)	0.93(±0.009)	0.929(±0.007)
EGSC	0.189(±0.028)	0.095(±0.014)	0.03(±0.005)	0.986(±0.001)	0.901(±0.003)	**0.987(±0.004)**	**0.993(±0.003)**
GREED	0.98(±0.024)	0.644(±0.012)	0.232(±0.008)	0.935(±0.001)	0.785(±0.002)	0.965(±0.007)	0.967(±0.004)
ERIC	**0.14(±0.035)**	0.109(±0.03)	**0.026(±0.007)**	**0.994(±0.001)**	**0.966(±0.006)**	0.987(±0.004)	0.992(±0.005)
GENN	0.2(±0.01)	**0.054(±0.004)**	0.038(±0.002)	0.983	0.891(±0.001)	0.504(±0.01)	0.666(±0.001)
GEDGNN-v	0.729(±0.704)	0.49(±0.496)	0.151(±0.163)	0.965(±0.006)	0.902(±0.014)	0.957(±0.014)	0.957(±0.018)
GENN-A*	**0.668(±0.641)**	**0.357(±0.297)**	**0.131(±0.102)**	**0.985(±0.014)**	**0.942(±0.043)**	0.863(±0.102)	0.863(±0.102)
GEDGNN	2.022(±0.608)	2.114(±0.644)	0.249(±0.07)	0.954(±0.012)	0.931(±0.017)	**0.978(±0.006)**	**0.976(±0.006)**
SDTED	27.28	38.103	2.248	0.739	0.576	0.953	0.954
DFS-BMao$_{100}$	1.582	1.452	0.176	0.962	0.901	**1.0**	0.993
DFS-BMao$_{200}$	0.552	0.543	0.071	0.985	0.947	1.0	0.993
App-BMao$_{10}$	2.75	2.51	0.384	0.952	0.854	0.989	0.983
App-BMao$_{30}$	0.496	0.438	0.078	0.988	0.945	1.0	**1.0**
App-BMao$_{100}$	**0.058**	**0.054**	**0.011**	**0.997**	**0.984**	1.0	1.0

(b) LINUX

	MSE_S ($\times 10^{-3}$) ↓	MSE_D ($\times 10^{-3}$) ↓	MAE ↓	ρ ↑	τ ↑	p@10 ↑	p@20 ↑
SimGNN	3.783(±0.141)	5.72(±1.936)	4.155(±0.283)	0.904(±0.017)	0.776(±0.025)	0.736(±0.024)	0.796(±0.008)
EGSC	0.423(±0.137)	3.78(±0.821)	4.493(±0.613)	**0.955(±0.007)**	**0.874(±0.015)**	**0.918(±0.009)**	**0.937(±0.01)**
GREED	2.445(±0.262)	**2.513(±0.306)**	**2.295(±0.144)**	0.936(±0.003)	0.822(±0.005)	0.748(±0.033)	0.789(±0.043)
ERIC	0.741(±0.136)	4.682(±1.981)	4.675(±2.018)	0.856(±0.02)	0.78(±0.019)	0.836(±0.077)	0.811(±0.112)
GENN	**0.268(±0.005)**	5.193(±0.331)	5.802(±0.33)	0.942(±0.018)	0.854(±0.03)	0.765(±0.086)	0.841(±0.056)
GEDGNN-v	15.978(±11.229)	51.919(±73.826)	10.9(±4.93)	0.654(±0.068)	0.571(±0.073)	0.67(±0.088)	0.698(±0.067)
GENN-A*	OOM	OOM	OOM	OOM	OOM	OOM	OOM
GEDGNN	**2.708(±0.226)**	**4.155(±0.111)**	**1.281(±0.023)**	**0.938(±0.006)**	**0.89(±0.007)**	**0.892(±0.005)**	**0.9(±0.011)**
SDTED	6.71	12.732	2.781	0.855	0.779	0.851	0.865
DFS-BMao$_{100}$	0.253	0.27	0.116	0.995	0.974	0.981	0.982
DFS-BMao$_{200}$	0.244	0.26	0.111	0.995	0.975	0.981	0.983
App-BMao$_{10}$	0.333	0.452	0.105	0.994	0.974	0.977	0.982
App-BMao$_{30}$	0.18	0.205	0.062	0.997	0.982	0.985	0.987
App-BMao$_{100}$	**0.111**	**0.128**	**0.045**	**0.998**	**0.987**	**0.99**	**0.991**

(c) IMDB

Fig. 6. Compare all approaches for GED prediction on AIDS, LINUX and IMDB. The approaches are grouped into three categories: non-interpretable GED prediction, interpretable GED prediction, and non-learning-based heuristics. For each category and each evaluation metric, the best result is highlighted in bold, while the second best result is underlined. ↓ means that the smaller the value the better, while ↑ means that the larger the value the better.

shown in Figure 6. Firstly, we can see that the estimation accuracy of our algorithm App-BMao consistently increases when t increases. This confirms with our theoretical analysis in Lemma 5.2, by noting that App-BMao provides an interpretable GED estimation that is always no smaller

Table 3. Total training time and testing time (in seconds)

	AIDS		LINUX		IMDB	
	Train	Test	Train	Test	Train	Test
SimGNN	53745	139	111112	285	111980	269
TaGSim	4176	29	9646	340	N/A	N/A
EGSC	901	15	1784	39	1093	227
GREED	1302	102	837	216	752	214
ERIC	3110	133	1374	274	2515	94
GENN	1117	8.69	1287	203	25131	17632
GEDGNN-v	4273	58	8526	207	5840	300
GENN-A*	241538	90530	34959	13671	OOM	OOM
GEDGNN	4273	20509	8526	17481	5840	25229
SDTED	0	15.83	0	15.36	0	19.22
DFS-BMao$_{100}$	0	7.59	0	5.69	0	8.22
DFS-BMao$_{200}$	0	9.92	0	7.17	0	11.54
App-BMao$_{10}$	0	2.97	0	2.91	0	5.57
App-BMao$_{30}$	0	5.29	0	3.97	0	7.54
App-BMao$_{100}$	0	8.22	0	5.58	0	13.77
AStar-BMao	0	290	0	535	0	249447

than the true GED value. Secondly, even App-BMao$_{10}$ (i.e., App-BMao with $t = 10$) consistently and significantly outperforms the state-of-the-art heuristic algorithm SDTED, demonstrating the ineffectiveness of LSAPE-based approaches for GED estimation. Thirdly, App-BMao$_{100}$ performs better than DFS-BMao$_{200}$ with comparable running time (as shown in Table 3), demonstrating that A* search is better than DFS search for GED estimation. Fourthly, App-BMao$_{100}$ outperforms all learning-based approaches including both interpretable and non-interpretable GED predictions, in terms of GED estimation accuracy. Also, App-BMao$_{30}$ outperforms the learning-based approaches in most of the cases. This demonstrates that ***the current learning-based approaches are not effective enough to outperform our combinatorial search algorithm*** App-BMao ***that provides interpretable GED prediction and has a controlled time and space complexity***. We remark that in the literature of designing learning-based approaches for GED prediction, they only compared their approaches with a very basic combinatorial algorithm (Hungarian or its variants Munkre and VJ). We urge that all future research on proposing learning-based approaches for GED prediction should compare our algorithm App-BMao as a baseline.

Exp-5: Efficiency Evaluation. In this testing, we further compare the running time of all evaluated approaches in both training and testing. The results are shown in Table 3. Note that, the reported training time is the total training time for all training graph pairs, and the reported testing time is also the total testing time for all testing graph pairs. Non-learning-based approaches do not have a training part; thus their training time is recorded as 0 in Table 3. We can see that learning-based approaches spend a significant amount of time in training their models. Moreover, they take significantly more time in inference than non-learning-based approaches. This further demonstrates the ineffectiveness of the existing learning-based approaches for GED prediction.

In addition, we also include the running time of the exact algorithm AStar-BMao in Table 3. For each graph pair, we set a time limit of 100 seconds for AStar-BMao, and record its time as 100 seconds if it exceeds the time limit. In our testing, we observe that AStar-BMao finishes within the time limit for all graph pairs of AIDS and LINUX, and exceeds the time limit for 1197 out of 150398 ($\approx 0.8\%$) graph pairs of IMDB. We can see that AStar-BMao takes a significant amount of

Table 4. Results on the IMDB_{pyg} dataset that has incorrect ground-truth GED values

	MSE_{S} ($\times 10^{-3}$) ↓	MSE_{D} ($\times 10^{-3}$) ↓	MAE ↓	ρ ↑	τ ↑	p@10 ↑	p@20 ↑
SimGNN	2.217(±0.06)	19.797(±5.106)	16.217(±1.671)	0.909(±0.001)	0.776	0.743(±0.041)	0.772(±0.023)
EGSC	0.626(±0.066)	26.359(±2.167)	25.548(±1.599)	**0.931(±0.007)**	0.817(±0.01)	**0.856(±0.011)**	**0.864(±0.01)**
GREED	1.277(±0.105)	**3.198(±0.074)**	**3.908(±0.122)**	0.927(±0.001)	**0.825**	0.671(±0.038)	0.724(±0.041)
ERIC	**0.522(±0.107)**	31.276(±11.134)	29.742(±10.945)	0.803(±0.02)	0.721(±0.017)	0.74(±0.029)	0.677(±0.084)
GENN	0.686(±0.016)	9.156(±0.967)	30.17(±1.771)	0.912(±0.001)	0.805(±0.001)	0.53(±0.022)	0.625(±0.02)
GEDGNN-v	9.393(±4.248)	312.894(±504.557)	52.64(±28.724)	0.622(±0.053)	0.543(±0.053)	0.644(±0.089)	0.659(±0.078)
GEDGNN	**1.36(±0.096)**	**7.109(±0.589)**	**4.125(±0.207)**	**0.899(±0.004)**	**0.849(±0.005)**	**0.846(±0.009)**	**0.847(±0.007)**
SDTED	2.779	12.425	5.849	0.896	0.799	0.809	0.82
DFS-BMao_{100}	0.488	2.524	2.154	0.968	0.904	**0.919**	0.927
DFS-BMao_{200}	0.489	2.529	2.156	0.968	0.904	0.918	**0.928**
App-BMao_{10}	**0.486**	**2.52**	**2.137**	0.969	0.912	0.912	0.916
App-BMao_{30}	0.499	2.59	2.185	**0.97**	0.914	0.911	0.921
App-BMao_{100}	0.506	2.624	2.2	0.97	**0.915**	0.911	0.923

time for IMDB, and is consistently much slower than non-learning-based heuristic algorithms. Nevertheless, AStar-BMao runs much faster than interpretable GED prediction models GENN-A* and GEDGNN on AIDS and LINUX where the graphs have only up-to 10 nodes.

Exp-6: IMDB v.s. IMDB_{pyg}. In this testing, we evaluate the effect of incorrect ground-truth GED value in the IMDB dataset. Recall that, we use IMDB to represent the dataset with correct ground-truth values and IMDB_{pyg} to denote the original dataset provided by [1] (with incorrect ground-truth GED values). We remark that some of the existing GED learning models (e.g., GREED and GEDGNN) used their own training data, while others (e.g., SimGNN, EGSC, ERIC, GENN and GENN-A*) directly used the incorrect GED value provided by [1] in their model training and evaluation. The results on IMDB_{pyg} are reported in Table 4; GENN-A* is excluded due to run out-of-memory. By comparing it with the results in Figure 6(c), we can see that the models generally achieve higher accuracy on the IMDB dataset than IMDB_{pyg}. This is especially true for the models that directly used the IMDB_{pyg} dataset for training, e.g., see the results for the MAE metric. Furthermore, the accuracy of our heuristic algorithm App-BMao in Table 4 becomes worse when t increases, in contradiction to our theoretical analysis in Lemma 5.2. This is because most of the incorrect GED values from IMDB_{pyg} are larger than the actual GED values, and App-BMao produces an upper bound of the GED value as the estimation. When t increases, App-BMao provides a more accurate estimation (i.e., smaller upper bound) and thus further away from the incorrect ground-truth GED value that is already larger than that computed by App-BMao_{10}. Nevertheless, the overall trend of the ranking of the different learning-based approaches are not significantly affected.

Exp-7: Train-Test v.s. Test-Test. This experiment is conducted on a set of different testing graph pairs. In all previous testings, the testing graph pairs are generated by taking one graph from the training graph set and another graph from the testing graph set, which is provided by `torch_geometric`. We suspected that as graphs from the training graph set are used in the training stage, it may affect the evaluation result. Thus, we also test the models on a new set of graph pairs, called test-test graph pairs, where both graphs in a testing graph pair are chosen from the testing graph set. The results on the new test-test graph pairs are shown in Figure 7. We can see that, for learning-based approaches, the results on train-test set show a better performance than that tested on test-test set. This is because, from the sight of the models, graphs from test-test set are totally new. The pattern learned from the training graphs will not be exactly suitable for these graphs, which leads to worse performance. Also, we observe that GREED is no longer in the list of top-two best performers among the non-interpretable GED prediction approaches, suggesting that GREED may not generalize so well compared to other approaches. For non-learning-based approaches, there is no clear pattern shown, because train-test and test-test make no difference to them.

	MSE_S ($\times 10^{-3}$) ↓	MSE_D ($\times 10^{-3}$) ↓	MAE ↓	ρ ↑	τ ↑	p@10 ↑	p@20 ↑
SimGNN	2.722(±0.036)	3.487(±0.181)	0.85(±0.033)	0.846(±0.001)	0.674(±0.002)	0.685(±0.001)	0.773(±0.001)
TaGSim	5.549(±0.809)	10.361(±1.025)	1.428(±0.081)	0.681(±0.023)	0.506(±0.021)	0.661(±0.087)	0.661(±0.087)
EGSC	1.746(±0.087)	2.676(±0.143)	**0.694(±0.019)**	**0.882(±0.003)**	**0.717(±0.003)**	**0.763(±0.012)**	**0.82(±0.006)**
GREED	1.913(±0.099)	2.661(±0.226)	0.729(±0.027)	0.871(±0.003)	0.699(±0.004)	0.74(±0.008)	0.811(±0.005)
ERIC	**1.68(±0.051)**	2.808(±0.199)	0.699(±0.017)	0.848(±0.002)	0.674(±0.003)	0.755(±0.023)	0.813(±0.007)
GENN	2.307(±0.216)	**1.431(±0.072)**	0.764(±0.019)	0.857(±0.009)	0.689(±0.01)	0.57(±0.021)	0.625(±0.009)
GEDGNN-v	2.932(±0.113)	4.294(±0.216)	0.914(±0.023)	0.82(±0.005)	0.683(±0.005)	0.66(±0.008)	0.724(±0.006)
GENN-A*	**1.352(±0.079)**	**1.147(±0.104)**	**0.478**	**0.931(±0.003)**	**0.815(±0.011)**	0.788(±0.037)	0.812(±0.029)
GEDGNN	6.682(±0.243)	14.113(±0.785)	1.517(±0.035)	0.794(±0.004)	0.693(±0.004)	**0.87(±0.006)**	**0.862(±0.002)**
SDTED	27.241	78.395	4.412	0.589	0.429	0.669	0.667
DFS-BMao$_{100}$	0.576	0.919	0.17	0.958	0.89	0.979	0.979
DFS-BMao$_{200}$	0.162	0.277	0.056	0.986	0.951	0.994	0.997
App-BMao$_{10}$	6.775	12.586	1.459	0.789	0.629	0.808	0.764
App-BMao$_{30}$	1.169	1.975	0.382	0.935	0.829	0.959	0.95
App-BMao$_{100}$	**0.103**	**0.185**	**0.053**	**0.991**	**0.954**	**0.998**	**0.998**

(a) AIDS

	MSE_S ($\times 10^{-3}$) ↓	MSE_D ($\times 10^{-3}$) ↓	MAE ↓	ρ ↑	τ ↑	p@10 ↑	p@20 ↑
SimGNN	1.494(±0.138)	0.934(±0.091)	0.317(±0.001)	0.957(±0.001)	0.837(±0.002)	0.942(±0.016)	0.968(±0.001)
TaGSim	6.187(±0.272)	3.866(±0.196)	0.533(±0.032)	0.93(±0.006)	0.791(±0.006)	0.937(±0.006)	0.86(±0.006)
EGSC	0.222(±0.028)	0.118(±0.018)	**0.034(±0.005)**	0.986(±0.001)	0.905(±0.003)	0.993(±0.002)	0.995(±0.002)
GREED	1.042(±0.039)	0.672(±0.017)	0.234(±0.011)	0.946(±0.002)	0.809(±0.003)	0.973(±0.012)	0.963(±0.021)
ERIC	**0.189(±0.046)**	0.153(±0.039)	0.036(±0.008)	**0.991(±0.002)**	**0.955(±0.009)**	**0.997**	**0.997(±0.001)**
GENN	0.258(±0.001)	**0.074(±0.003)**	0.049(±0.002)	0.985	0.902(±0.002)	0.739(±0.035)	0.837(±0.019)
GEDGNN-v	0.845(±0.669)	0.56(±0.471)	0.168(±0.157)	0.973(±0.006)	0.917(±0.014)	0.958(±0.011)	0.961(±0.023)
GENN-A*	**0.676(±0.699)**	**0.364(±0.325)**	**0.135(±0.114)**	**0.987(±0.014)**	**0.948(±0.047)**	0.974(±0.031)	0.969(±0.041)
GEDGNN	1.974(±0.6)	2.066(±0.596)	0.248(±0.06)	0.96(±0.011)	0.935(±0.014)	**0.977(±0.012)**	**0.973(±0.011)**
SDTED	28.695	40.949	2.362	0.752	0.588	0.954	0.871
DFS-BMao$_{100}$	1.462	1.326	0.167	0.966	0.913	0.985	0.978
DFS-BMao$_{200}$	0.533	0.517	0.07	0.986	0.957	0.988	0.995
App-BMao$_{10}$	2.871	2.59	0.399	0.951	0.856	0.968	0.957
App-BMao$_{30}$	0.501	0.444	0.082	0.988	0.95	0.997	0.984
App-BMao$_{100}$	**0.071**	**0.062**	**0.012**	**0.998**	**0.987**	**1.0**	**1.0**

(b) LINUX

	MSE_S ($\times 10^{-3}$) ↓	MSE_D ($\times 10^{-3}$) ↓	MAE ↓	ρ ↑	τ ↑	p@10 ↑	p@20 ↑
SimGNN	5.656(±0.096)	3.836(±0.547)	2.543(±0.243)	0.89(±0.006)	0.746(±0.011)	0.835(±0.004)	0.88(±0.018)
EGSC	1.746(±0.087)	3.131(±0.234)	**0.694(±0.019)**	0.882(±0.003)	0.717(±0.003)	0.763(±0.012)	0.82(±0.006)
GREED	2.947(±0.309)	2.603(±0.314)	2.231(±0.148)	0.909(±0.004)	0.781(±0.006)	0.798(±0.064)	0.851(±0.053)
ERIC	0.915(±0.145)	1.449(±0.208)	1.488(±0.117)	0.951(±0.006)	0.851(±0.013)	**0.837(±0.128)**	0.858(±0.081)
GENN	**0.516(±0.067)**	**0.988(±0.023)**	1.251(±0.052)	**0.96(±0.008)**	**0.868(±0.011)**	0.799(±0.01)	**0.887(±0.004)**
GEDGNN-v	18.467(±12.47)	30.36(±39.293)	6.396(±3.554)	0.792(±0.051)	0.685(±0.062)	0.777(±0.057)	0.804(±0.051)
GEDGNN	**3.995(±0.686)**	**4.876(±0.442)**	**1.191(±0.077)**	**0.937(±0.007)**	**0.895(±0.009)**	**0.918(±0.008)**	**0.926(±0.008)**
SDTED	8.569	15.538	2.65	0.851	0.781	0.871	0.856
DFS-BMao$_{100}$	2.757	0.502	0.116	0.991	0.97	0.987	0.991
DFS-BMao$_{200}$	2.737	0.484	0.11	0.991	0.971	0.988	0.991
App-BMao$_{10}$	3.093	1.279	0.139	0.987	0.967	0.978	0.991
App-BMao$_{30}$	0.405	0.553	0.088	0.991	0.974	0.985	0.995
App-BMao$_{100}$	**0.224**	**0.318**	**0.054**	**0.995**	**0.983**	**0.991**	**0.997**

(c) IMDB

Fig. 7. Evaluate all approaches for GED prediction on AIDS, LINUX and IMDB using the test-test graph pairs.

7 Conclusion

This paper bridges a knowledge gap observed in the existing studies of learning-based approaches for GED estimation. We first conducted a holistic review of the existing learning-based approaches, by categorizing them into non-interpretable GED prediction and interpretable GED prediction. We analyzed and highlighted the overarching design principles and relationships among these models. Furthermore, we presented a simple yet effective combinatorial heuristic algorithm App-BMao for interpretable GED prediction, which has a controlled time and space complexity. Extensive

empirical evaluations on three widely used datasets, AIDS, LINUX and IMDB, show that there is no clear winner among the learning-based approaches. EGSC, GREED, ERIC and GENN are generally candidates of the best performers for non-interpretable GED prediction, while GREED does not generalize as well as the other three. Nevertheless, none of the existing learning-based (interpretable or non-interpretable) GED prediction approaches is effective enough to outperform our new heuristic algorithm App-BMao that provides interpretable GED prediction and has a controlled time and space complexity. Moreover, App-BMao runs significantly faster.

Our experimental evaluations suggest that a non-learning-based approach can perform very well for GED estimation with uniform edit costs. On the other hand, learning-based (i.e., GNN-based) approaches, though has the advantage of being generally applicable to any similarity measure, have inherent limitations. For example, they currently do not handle edge labels effectively, the trained models cannot generalize to new datasets with different label vocabularies, and the cost of both training and obtaining training data is high. Thus, two possible directions of future studies on learning-based approaches for GED estimation could be overcoming these limitations and focusing on GED with non-uniform edit costs.

Acknowledgments

We thank the anonymous reviewers for their constructive comments and suggestions. The work is supported by the Australian Research Council Fundings of DP220103731.

References

[1] [n. d.]. PyTorch Geometric. https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.datasets.GEDDataset.html?highlight=geddataset#torch_geometric.datasets.GEDDataset

[2] Zeina Abu-Aisheh, Romain Raveaux, and Jean-Yves Ramel. 2016. Anytime graph matching. *Pattern Recognit. Lett.* 84 (2016), 215–224.

[3] Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. 2015. An Exact Graph Edit Distance Algorithm for Solving Pattern Recognition Problems. In *Proc. of ICPRAM'15*. 271–278.

[4] Jiyang Bai and Peixiang Zhao. 2021. TaGSim: Type-aware Graph Similarity Learning and Computation. *Proc. VLDB Endow.* 15, 2 (2021), 335–347. doi:10.14778/3489496.3489513

[5] 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 *Proc. of WSDM'19*. 384–392.

[6] 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 *Proc. of AAAI'20*. 3219–3226.

[7] Franka Bause, Christian Permann, and Nils M. Kriege. 2024. Approximating the Graph Edit Distance with Compact Neighborhood Representations. In *Proc. of ECML PKDD'24*, Vol. 14945. 300–318. doi:10.1007/978-3-031-70362-1_18

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

[9] David B. Blumenthal and Johann Gamper. 2017. Exact Computation of Graph Edit Distance for Uniform and Non-uniform Metric Edit Costs. In *Proc. of GbRPR'17*. 211–221.

[10] Vincenzo Carletti, Benoit Gaüzère, Luc Brun, and Mario Vento. 2015. Approximate Graph Edit Distance Computation Combining Bipartite Matching and Exact Neighborhood Substructure Distance. In *Proc. of GbRPR'15*, Vol. 9069. 188–197.

[11] Lijun Chang, Xing Feng, Xuemin Lin, Lu Qin, Wenjie Zhang, and Dian Ouyang. 2020. Speeding Up GED Verification for Graph Similarity Search. In *Proc. of ICDE'20*. 793–804.

[12] Lijun Chang, Xing Feng, Kai Yao, Lu Qin, and Wenjie Zhang. 2023. Accelerating Graph Similarity Search via Efficient GED Computation. *IEEE Trans. Knowl. Data Eng.* 35, 5 (2023), 4485–4498.

[13] Chandra R. Chegireddy and Horst W. Hamacher. 1987. Algorithms for finding K-best perfect matchings. *Discret. Appl. Math.* 18, 2 (1987), 155–165.

[14] Khoa D. Doan, Saurav Manchanda, Suchismit Mahapatra, and Chandan K. Reddy. 2021. Interpretable Graph Similarity Computation via Differentiable Optimal Alignment of Node Embeddings. In *Proc. of SIGIR'21*. 665–674.

[15] Stefan Fankhauser, Kaspar Riesen, and Horst Bunke. 2011. Speeding Up Graph Edit Distance Computation through Fast Bipartite Matching. In *Proc. of IAPR-TC-15'11*, Vol. 6658. 102–111. doi:10.1007/978-3-642-20844-7_11

[16] Benoit Gaüzère, Sébastien Bougleux, Kaspar Riesen, and Luc Brun. 2014. Approximate Graph Edit Distance Guided by Bipartite Matching of Bags of Walks. In *Proc. of S+SSPR'14*, Vol. 8621. 73–82.

[17] Karam Gouda and Mosab Hassaan. 2016. CSI_GED: An efficient approach for graph edit similarity computation. In *Proc. of ICDE'16*.
[18] William L. Hamilton. 2020. Graph Representation Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14, 3 (2020), 1–159.
[19] Roy Jonker and A. Volgenant. 1987. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing* 38, 4 (1987), 325–340. doi:10.1007/BF02278710
[20] Derek Justice and Alfred O. Hero III. 2006. A Binary Linear Programming Formulation of the Graph Edit Distance. *IEEE Trans. Pattern Anal. Mach. Intell.* 28, 8 (2006), 1200–1214.
[21] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proc. of ICLR'17*.
[22] Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly* 2, 1-2 (1955), 83–97.
[23] James Munkres. 1957. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics* 5, 1 (1957), 32–38.
[24] Yun Peng, Byron Choi, and Jianliang Xu. 2021. Graph Edit Distance Learning via Modeling Optimum Matchings with Constraints. In *Proc. of IJCAI'21*. 1534–1540. doi:10.24963/IJCAI.2021/212
[25] Chengzhi Piao, Tingyang Xu, Xiangguo Sun, Yu Rong, Kangfei Zhao, and Hong Cheng. 2023. Computing Graph Edit Distance via Neural Graph Matching. *Proc. VLDB Endow.* 16, 8 (2023), 1817–1829.
[26] Can Qin, Handong Zhao, Lichen Wang, Huan Wang, Yulun Zhang, and Yun Fu. 2021. Slow Learning and Fast Inference: Efficient Graph Similarity Computation via Knowledge Distillation. In *Proc. of NIPS'21*. 14110–14121.
[27] Rishabh Ranjan, Siddharth Grover, Sourav Medya, Venkat Chakravarthy, Yogish Sabharwal, and Sayan Ranu. 2022. GREED: a neural framework for learning graph distance functions. In *Proc. of NIPS'22*. Article 1636, 13 pages.
[28] Kaspar Riesen and Horst Bunke. 2009. Approximate graph edit distance computation by means of bipartite graph matching. *Image Vis. Comput.* 27, 7 (2009), 950–959. doi:10.1016/j.imavis.2008.04.004
[29] Kaspar Riesen, Sandro Emmenegger, and Horst Bunke. 2013. A Novel Software Toolkit for Graph Edit Distance Computation. In *Proc. of GbRPR'13*.
[30] Kaspar Riesen, Stefan Fankhauser, and Horst Bunke. 2007. Speeding Up Graph Edit Distance Computation with a Bipartite Heuristic. In *Proc. of MLG'07*.
[31] Alberto Sanfeliu and King-Sun Fu. 1983. A distance measure between attributed relational graphs for pattern recognition. *IEEE Trans. Systems, Man, and Cybernetics* 13, 3 (1983), 353–362.
[32] Francesc Serratosa. 2014. Fast computation of Bipartite graph matching. *Pattern Recognit. Lett.* 45 (2014), 244–250.
[33] Francesc Serratosa. 2015. Speeding up Fast Bipartite Graph Matching Through a New Cost Matrix. *Int. J. Pattern Recognit. Artif. Intell.* 29, 2 (2015), 1550010:1–1550010:17. doi:10.1142/S021800141550010X
[34] Richard Socher, Danqi Chen, Christopher D. Manning, and Andrew Y. Ng. 2013. Reasoning With Neural Tensor Networks for Knowledge Base Completion. In *Proc. of NIPS'13*. 926–934.
[35] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *Proc. of ICLR'18*.
[36] Runzhong Wang, Tianqi Zhang, Tianshu Yu, Junchi Yan, and Xiaokang Yang. 2021. Combinatorial Learning of Graph Edit Distance via Dynamic Embedding. In *Proc. of CVPR'21*. 5241–5250.
[37] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. 2019. Simplifying Graph Convolutional Networks. In *Proc. of ICML'19*, Vol. 97. 6861–6871.
[38] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *Proc. of ICLR'19*.
[39] Lei Yang and Lei Zou. 2021. Noah: Neural-optimized A* Search Algorithm for Graph Edit Distance Computation. In *Proc. of ICDE'21*. 576–587.
[40] Peilun Yang, Hanchen Wang, Jianye Yang, Zhengping Qian, Ying Zhang, and Xuemin Lin. 2024. Deep Learning Approaches for Similarity Computation: A Survey. *IEEE Trans. Knowl. Data Eng.* 36, 12 (2024), 7893–7912.
[41] Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. 2009. Comparing stars: on approximating graph edit distance. *Proc. VLDB Endow.* 2, 1 (Aug. 2009), 25–36. doi:10.14778/1687627.1687631
[42] Zhen Zhang, Jiajun Bu, Martin Ester, Zhao Li, Chengwei Yao, Zhi Yu, and Can Wang. 2021. H2MN: Graph Similarity Learning with Hierarchical Hypergraph Matching Networks. In *Proc. of KDD'21*. 2274–2284.
[43] Wei Zhuo and Guang Tan. 2022. Efficient graph similarity computation with alignment regularization. In *Proc. of NIPS'22*. Article 2188, 13 pages.

Received October 2024; revised January 2025; accepted February 2025