Computing Approximate Graph Edit Distance via Optimal Transport

Anonymous Author(s)

ABSTRACT

Given a graph pair (G^1, G^2), graph edit distance (GED) is defined as the minimum number of edit operations converting G^1 to G^2. GED is a fundamental operation widely used in many applications, but its exact computation is NP-hard, so the approximation of GED has gained a lot of attention. Data-driven learning-based methods have been found to provide superior results compared to classical approximate algorithms, but they directly fit the coupling relationship between a pair of vertices from their vertex features. We argue that while pairwise vertex features can capture the coupling cost (discrepancy) of a pair of vertices, the vertex coupling matrix should be derived from the vertex-pair cost matrix through a more well-established method that is aware of the global context of the graph pair, such as optimal transport. In this paper, we propose an ensemble approach that integrates a supervised learning-based method and an unsupervised method, both based on optimal transport. Our learning method, GEDIOT, is based on inverse optimal transport that leverages a learnable Sinkhorn algorithm to generate the coupling matrix. Our unsupervised method, GEDGW, models GED computation as a linear combination of optimal transport and its variant, Gromov-Wasserstein discrepancy, for node and edge operations, respectively, which can be solved efficiently without needing the ground truth. Our ensemble method, GEDHOT, combines GEDIOT and GEDGW to further boost the performance. Extensive experiments demonstrate that our methods significantly outperform the existing methods in terms of the performance of GED computation, edit path generation, and model generalizability.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; • **Information systems** → **Information systems applications**.

KEYWORDS

Graph edit distance, Optimal transport, Graph neural network

ACM Reference Format:

Anonymous Author(s). 2018. Computing Approximate Graph Edit Distance via Optimal Transport. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 26 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

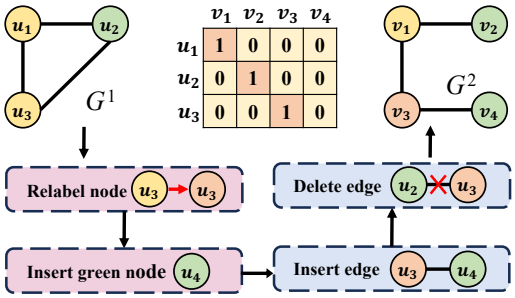


Figure 1: A toy example of graph pair (G^1, G^2)

1 INTRODUCTION

Graph edit distance (GED) is one of the most widely used graph similarity metrics, which is defined as the minimum number of edit operations that transform one graph to the other. GED has wide applications, such as graph similarity search [27, 56, 65, 70, 73], graph classification [39, 40], and inexact graph matching [7]. Scenarios include handwriting recognition [17], image indexing [58], semantic image matching [55], and investigations of antiviral drugs [71], etc. The lower part of Figure 1 illustrates the edit path (i.e., sequence of edit operations) of a graph pair (G^1, G^2) with GED = 4. The edit path with the minimum length is called Graph Edit Path (GEP), so the length of a GEP is exactly the GED.

Existing methods for GED computation can be categorized into the following three types. **(1) Exact Algorithms.** GED can be computed exactly by the A* algorithm [41], but due to being NP-hard [65], it is time-consuming even for a pair of 6-node graphs [4]. **(2) Approximate Algorithms.** To make computation tractable, approximate algorithms are proposed based on discrete optimization or combinatorial search such as A*-Beam [32], Hungarian [40] and VJ [16]. A*-beam restricts the search space of A* algorithm, which is still an exponential-time algorithm. Hungarian and VJ convert the GED computation to a linear sum assignment problem and find the optimal node matching between two graphs, which takes $O(n^3)$ time. Moreover, these heuristic methods lack a theoretical guarantee and generate results of inferior quality. **(3) Learning-based Methods.** Recent studies turn to data-driven methods based on graph neural networks (GNN) to achieve better performance [2, 3, 36, 38, 63]. Differing from the approximate algorithms, learning-based methods extract intra-graph and inter-graph information by generating node and graph embeddings, which are then used to predict GEDs with smaller errors within $O(n^2)$ time in the worst case. The two most recent works, Noah [63] and GEDGNN [36], further support generating the edit path based on A*-beam search and k-best matching, respectively, to ensure the feasibility of the predicted GED.

However, a key issue remains with these learning-based methods. Specifically, they compute a pairwise vertex discrepancy matrix $\mathbf{A}$ where each element $\mathbf{A}_{i,j}$ corresponds to the coupling cost (discrepancy) of matching vertex i in G^1 to vertex j in G^2, and $\mathbf{A}_{i,j}$ is

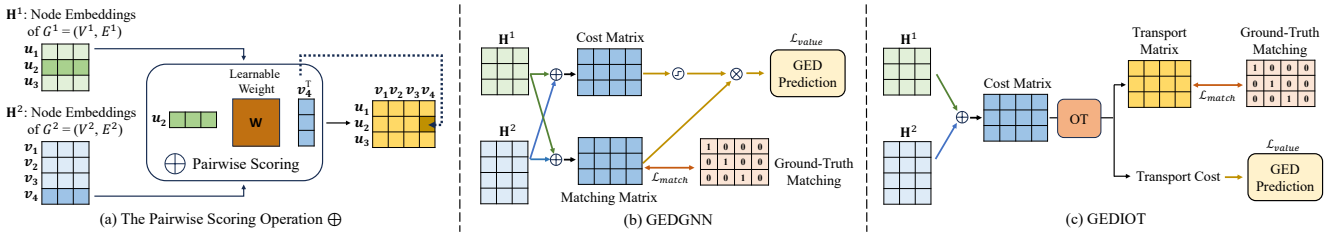


Figure 2: OT Motivation and Learning-based Model Comparison

computed only from their vertex features. As Figure 2(a) shows, a shared operation of all existing learning-based methods (including our GEDIOT) is pairwise scoring, which given two node embedding matrices obtained from G^1 and G^2 (via a graph neural network), returns a matrix $\mathbf{A}$ where element $\mathbf{A}_{i,j}$ is the pairwise score computed from the embeddings of node u_i in G^1 and node v_j in G^2. Here, we use $\oplus$ to denote the pairwise scoring operation. Existing learning-based models directly treat $\mathbf{A}$ as the vertex coupling matrix to fit the ground-truth vertex coupling relationship, but we argue that the coupling matrix should be derived from the pairwise discrepancy matrix $\mathbf{A}$ through a more well-established method that is aware of the global context of the graph pair, such as optimal transport [24]. As illustrated by the bottom branch of Figure 2(b) for GEDGNN [36], they fit $\mathbf{A}$ directly to the 0-1 ground-truth node-matching matrix for GED. Note that the optimal node matching is a global decision: node u_i in G^1 is matched to node v_j in G^2 in the GED solution not only because they have similar labels and neighborhood structures, but also because, for example, node u_i in G^1 is not as similar to the other nodes (e.g., node v_k, $k \neq j$) in G^2. However, $\mathbf{A}_{i,j}$ is computed only based on the embeddings of nodes u_i and v_j.

To fundamentally address this drawback, we propose solutions based on the foundation of the Optimal Transport (OT) theory. OT is a mathematical framework that focuses on finding the most efficient way to move and transform one distribution of mass into another, which has been successfully applied in various fields [13, 24, 61]. Laid upon rigid mathematical theory [34, 51], OT provides strong theoretical guarantees and well-understood properties. With the development of numerical algorithms, such as the Sinkhorn algorithm [14], it is particularly effective and efficient when embedding sentences or graph vertices as probabilistic distributions in the Wasserstein space derived from optimal transport [60, 61].

In this paper, we propose an ensemble approach that integrates a supervised learning-based method and an unsupervised method, both based on OT. Our learning-based method, GEDIOT, is based on inverse optimal transport (IOT) [12, 47] that leverages a learnable Sinkhorn algorithm to generate the coupling matrix. As Figure 2(c) shows, our GEDIOT model takes the cost matrix computed by pairwise scoring, and passes it through an OT module to minimize the cost of transporting masses from nodes of G^1 to nodes of G^2, which returns the learned transport matrix that considers the global cost matrix when fitting the ground-truth node-matching matrix for GED. As our experiments have shown, adding OT after the pairwise-scoring-induced cost matrix brings significant performance improvement in both GED and GEP predictions.

Based on optimal transport, we also propose an unsupervised method, GEDGW, that models GED computation as a linear combination of optimal transport and its variant, Gromov-Wasserstein (GW) discrepancy, for node and edge operations, respectively, which can be solved efficiently without the ground truth. Our ensemble method, GEDHOT, combines GEDIOT and GEDGW to further boost the performance. Our contributions are listed as follows:

- We propose a neural network architecture based on inverse optimal transport (where the cost matrix is learnable) that formulates the GED learning task as a bi-level optimization problem, named GEDIOT (GED with IOT), which introduces the OT component to capture the global context effectively.
- To make OT applicable, GEDIOT extends the learned cost matrix with a dummy row and utilizes the Sinkhorn algorithm with a learnable regularization coefficient to integrate OT with neural networks for GED computation, improving the model performance and stability. Since the coupling matrix can represent the confidence of node matching, we can also generate the edit path from it using the k-best matching algorithm of [36].
- We separate the edit operations into two types: vertex edit operations and edge edit operations. We then model the GED computation as an optimization problem combining optimal transport (for vertex edits) and its variant Gromov-Wasserstein discrepancy (for edge edits), leading to our unsupervised solution named GEDGW (Graph Edit Distance with Gromov-Wasserstein discrepancy).
- We combine GEDIOT and GEDGW into an ensemble method named GEDHOT (Graph Edit Distance with Hybrid Optimal Transport) for more accurate GED computation.
- Extensive experiments show the superior performance of proposed methods. Compared with the state-of-the-art existing method GEDGNN [36], the Mean Absolute Error (MAE) on GED computation decreases by 20.5%–63.8% with GEDIOT. Furthermore, the hybrid method GEDHOT achieves the best performance, where the MAE decreases by 31.2%–72.3% compared with GEDGNN. We also conduct experiments to verify the high-quality edit path generation and superior generalizability of our methods.

The rest of this paper is organized as follows. Section 2 reviews the related work, and Section 3 defines our problem and presents the background of OT. Then, Section 4 describes the proposed learning-based method GEDIOT, and Section 5 further proposes the unsupervised method GEDGW and the ensemble method GEDHOT, and analyzes the time complexity of our methods. Finally, Section 6 reports our experiments, and Section 7 concludes this paper.

2 RELATED WORK

GED Computation. Classical exact algorithms [4, 8] seek the exact graph edit distance for each graph pair. Due to the NP-hardness of GED computation, they fail to generate solutions in a limited time when the graph size increases. To make computation tractable,

plenty of heuristic algorithms are proposed, including A*-Beam [32], Hungarian [40] and VJ [16], all of which provide an approximate GED in polynomial time. Recently, graph neural networks (GNN) have become popular since the extracted node and graph embeddings can greatly help the performance in various tasks [25, 54, 59, 66, 67, 69, 74]. A number of GNN-based methods, such as SimGNN [3], TaGSim [2], Noah [63], MATA* [29] and GEDGNN [36], have also been proposed to generate embeddings for GED computation with adequate training data, which achieve the best performance in approximate GED computation. For a more detailed review of heuristic and GNN-based methods, please see Section A in our online supplementary file [1].

Graph Similarity Search. Given a query graph and a threshold, graph similarity search retrieves all graphs from a database with GED to the query graph within the given threshold. An important step in this task is to verify whether the GEDs of graph pairs are smaller than the threshold. A series of works [8, 9, 19, 22, 23, 28, 72] are proposed to speed up the GED verification process between the database and the query graph. It is related to, but also distinct from, GED computation. They focus on the filtering technique of search space based on the threshold, while GED computation seeks the difference between a pair of graphs and has no threshold available for filtering. However, when setting the similarity threshold to infinity, the verification step can also be extended for GED computation [8, 9].

Optimal Transport. The goal of optimal transport (OT) [34] is to minimize the cost of transporting mass from one distribution to another. It has been applied in various fields, including image and signal processing [24], natural language processing [61], and domain adaptation [13]. Inverse optimal transport (IOT) [12, 47] is an inverse process to the classical optimal transport, which calculates the cost matrix from the coupling matrix. Recent studies [44, 46] interpret classical contrastive learning as inverse optimal transport. DB-OT [45] applies inverse optimal transport to long-tailed classification. Legal case matching algorithms are proposed in [64] via inverse optimal transport. They all use the general inverse optimal transport with the cross-entropy loss to build an OT-assisted neural network model, and the relation between inverse optimal transport and graphs remains rarely studied as it requires careful design for different graph problems. While our proposed GEDIOT model is also based on IOT, as Section 4.2 will describe, in order for the Sinkhorn algorithm to be applicable to GED prediction, we need to modify the OT constraints by incorporating a dummy supernode.

A few works have also applied OT and its variants to other graph problems (but not GED) [10, 15, 33, 52, 53]. One of the most important variants is Gromov-Wasserstein discrepancy (GW) [30, 35], a measure used to compare two metric spaces, capturing the differences in their intrinsic geometric structures. GW has been applied for graph partitioning and graph matching [60]. Fused GW [49] is a combination of optimal transport and GW, which has been successfully applied in graph classification and clustering. However, the optimization objective of Fused GW does not consider the edit costs of unmatched vertices in GED computation, but the size of G^1 and G^2 may not match for a given graph pair (G^1, G^2), so as Section 5 will describe, our proposed GEDGW model first needs to add dummy nodes to incorporate such costs into the objective.

Table 1: Notations

Notation	Description
G	a labeled undirected graph
$V,\ E,\ L$	the node, edge and label sets of G
(G^1, G^2)	the graph pair for GED computation
$\mathbf{M}$	node label matching matrix of (G^1, G^2)
$\mathbf{A}^1,\ \mathbf{A}^2$	adjacency matrices of G^1 and G^2
$\mathbf{H}^1,\ \mathbf{H}^2$	final node embeddings of G^1 and G^2
$GED(G^1, G^2)$	the GED of graph pair (G^1, G^2)
$GEP(G^1, G^2)$	the GEP of graph pair (G^1, G^2)
$\boldsymbol{\pi}$	the coupling matrix between G^1 and G^2
$\boldsymbol{\pi}^*,\ GED^*(G^1, G^2)$	ground truths of the graph pair (G^1, G^2)
$\mathbf{1}_n,\ \mathbf{0}_n$	the n-dimensional vectors full of 1 and 0
$\cdot \Vert \cdot$	the concatenation operator
$\cdot \oslash \cdot$	the element-wise division
$\langle \mathbf{P}, \mathbf{Q} \rangle$	the Frobenius dot-product $\sum_i \sum_j (P_{i,j} Q_{i,j})$
$\mathcal{L}(\mathbf{C}^1, \mathbf{C}^2)$	the 4-th order tensor $\left((\mathbf{C}^1_{i,j} - \mathbf{C}^2_{k,l})^2\right)_{i,j,k,l}$
$\mathcal{L} \otimes \mathbf{B}$	the matrix $\left(\sum_{j,l} \mathcal{L}_{i,j,k,l} \mathbf{B}_{j,l}\right)_{i,k}$

3 PRELIMINARIES

This section introduces Graph Edit Distance (GED), Graph Edit Path (GEP), and the fundamental concepts of Optimal Transport (OT) on graphs. All vectors default to column vectors unless otherwise specified. Table 1 summarizes important notations for quick lookup.

3.1 Problem Statement

We consider two tasks: GED computation and GEP generation between two node-labeled undirected graphs $G^1 = (V^1, E^1, L^1)$ and $G^2 = (V^2, E^2, L^2)$. We discuss GED computation of edge-labeled graphs in Section H.1 [1] due to space limit. We denote $|V^1| = n_1, |E^1| = m_1$ and $|V^2| = n_2, |E^2| = m_2$. We assume that $n_1 \le n_2$ as otherwise, we can swap G^1 and G^2.

Graph Edit Distance (GED). Given the graph pair (G^1, G^2), graph edit distance $GED(G^1, G^2)$ is the minimum number of edit operations that transform G^1 to G^2. An edit operation is an insertion/deletion of a node/edge or the relabeling of a node.

Graph Edit Path (GEP). The edit path of the graph pair (G^1, G^2) is a sequence of edit operations that transform G^1 to G^2. The graph edit path $GEP(G^1, G^2)$ is the shortest one with length $GED(G^1, G^2)$.

Figure 1 shows a GEP of the graph pair (G^1, G^2), where different colors denote different vertex labels and $GED(G^1, G^2) = 4$.

Node Matching. The node matching (hereafter we use the terms "node" and "vertex" interchangeably) of (G^1, G^2) is an $n_1 \times n_2$ binary matrix, denoted by $\boldsymbol{\pi} \in \{0, 1\}^{n_1 \times n_2}$, where $\boldsymbol{\pi}_{i,k} = 1$ if the node $u_i \in V^1$ matches $v_k \in V^2$, and $\boldsymbol{\pi}_{i,k} = 0$ otherwise. Since we assume that $n_1 \le n_2$, $\boldsymbol{\pi}$ satisfies the following constraints

$$\boldsymbol{\pi} \mathbf{1}_{n_2} = \mathbf{1}_{n_1}, \quad \boldsymbol{\pi}^\top \mathbf{1}_{n_1} \le \mathbf{1}_{n_2}, \quad \mathbf{1}_{n_1}^\top \boldsymbol{\pi} \mathbf{1}_{n_2} = n_1, \tag{1}$$

where $\mathbf{1}_n$ is the n-dimensional vector full of 1, and $\mathbf{a} \le \mathbf{b}$ denotes that $\mathbf{a}_i \le \mathbf{b}_i$ for the i^{th} elements of $\mathbf{a}$ and $\mathbf{b}$ for all i. Intuitively, the constraints ensure that each of the n_1 vertices of G^1 is matched to a vertex in G^2 since Eq. (1) allows exactly one "1" in each row and at most one "1" in each column. As illustrated in the 0-1 matrix in Figure 1, nodes u_1, u_2 and u_3 in G^1 are matched to v_1, v_2 and v_3 in G^2, respectively, and v_4 in G^2 is not matched.

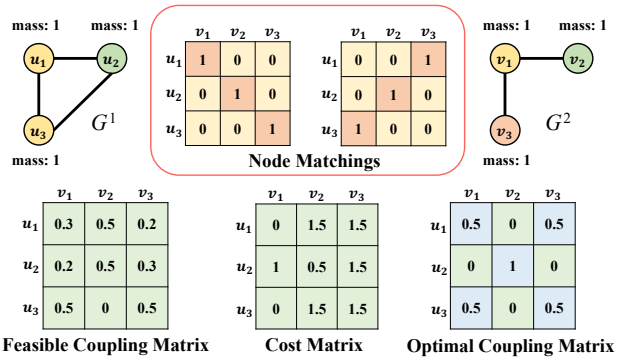


Figure 3: Example of Cost Matrix and Coupling Matrices

With a given node matching between G^1 and G^2, the edit path can be generated by traversing and comparing the differences between the labels and edges of all matching nodes in G^1 and G^2. Specifically, (i) we first check each node in G^2 to see if it is matched and if it has the same label as that of its matched node in G^1, if not, we add one to the Edit Distance (ED). This takes $O(n_2)$ time. In Figure 1, since v_3 and u_3 have different labels (hints a node relabeling), and v_4 (hints a node insertion) is not matched, we add 2 to ED. (ii) We then check each edge in G^1 to see if the corresponding edge (decided by the two matched end-nodes) exists in G^2, and vice versa; if not, we add one to ED. This takes $O(m_1 + m_2)$ time. In Figure 1 edge (u_2, u_3) exists in G^1 but the corresponding (v_2, v_3) based on node matching does not exist in G^2 (hints an edge deletion), and edge (v_3, v_4) exists in G^2 but there is no corresponding edge in G^1 (hints an edge insertion), so we add 2 to ED. Overall, the number of edit operations is 4. Note that the time complexity is linear (i.e., $O(n_2 + m_1 + m_2)$). The pseudo-code is shown in Algorithm 3 in Section C [1].

By relaxing the binary constraints of $\boldsymbol{\pi} \in \{0,1\}^{n_1 \times n_2}$ to $\boldsymbol{\pi} \in [0,1]^{n_1 \times n_2}$, node matching can be connected with the optimal transport theory to be introduced as follows.

3.2 Background of Optimal Transport

Optimal Transport (OT). The optimal transport problem seeks the most efficient way of transporting one distribution of mass into another. Given a graph pair (G^1, G^2), where $G^1 = (V^1, E^1, L^1)$ and $G^2 = (V^2, E^2, L^2)$, we assume there are two pre-defined mass distributions $\boldsymbol{\mu} = \{\boldsymbol{\mu}_i\}_{i=1}^{n_1}$ and $\boldsymbol{\nu} = \{\boldsymbol{\nu}_j\}_{j=1}^{n_2}$ on nodes of G^1 and G^2, respectively. For instance, when $n_1 = n_2$, for all u_i in G^1 and v_j in G^2, we can set their masses as $\boldsymbol{\mu}_i = 1$ and $\boldsymbol{\nu}_j = 1$, which puts the same importance weight on every node. Figure 3 shows our mass distributions on G^1 and G^2 where every node has mass 1. Coupling matrix $\boldsymbol{\pi} \in \mathbb{R}^{n_1 \times n_2}$ is a node-to-node mass transport matrix between G^1 and G^2, where each element $\boldsymbol{\pi}_{i,k}$ denotes the amount of mass transported from node $u_i \in V^1$ to $v_k \in V^2$. In our work, $\boldsymbol{\pi}_{i,k}$ is in the range $[0,1]$ reflecting the confidence that u_i matches v_k. The feasible set of coupling matrices of (G^1, G^2) is denoted by:

$$\Pi(\boldsymbol{\mu}, \boldsymbol{\nu}) = \left\{ \boldsymbol{\pi} \in \mathbb{R}^{n_1 \times n_2} \mid \boldsymbol{\pi} \mathbf{1}_{n_2} = \boldsymbol{\mu},\ \boldsymbol{\pi}^\top \mathbf{1}_{n_1} = \boldsymbol{\nu},\ \boldsymbol{\pi} \geq 0 \right\}.$$

The lower-left corner of Figure 3 shows an example of a feasible $\boldsymbol{\pi}$. The feasible set of $\boldsymbol{\pi}$ basically relaxes Eq. (1) to allow values in $[0,1]$, but still requires that elements in a row (resp. column) sum up to 1 (if $n_1 \neq n_2$, dummy nodes need to be added as we will describe in Section 4.2. So we are basically generalizing Eq. (1) for the case when G^1 and G^2 have the same size, where $\boldsymbol{\pi}^\top \mathbf{1}_{n_1} = \mathbf{1}_{n_2}$).

Algorithm 1: Sinkhorn algorithm

Input: cost matrix $\mathbf{C}$, mass distributions $\boldsymbol{\mu}$, $\boldsymbol{\nu}$, regularization coefficient ε, maximum iteration *maxiter*

$\mathbf{K} \leftarrow \exp(-\mathbf{C}/\varepsilon)$, $\boldsymbol{\varphi} \leftarrow \mathbf{1}_{n_1}$
for $m = 1$ *to maxiter* **do**
  $\boldsymbol{\psi} \leftarrow \boldsymbol{\nu} \oslash (\mathbf{K}^\top \boldsymbol{\varphi})$
  $\boldsymbol{\varphi} \leftarrow \boldsymbol{\mu} \oslash (\mathbf{K}\, \boldsymbol{\psi})$
$\boldsymbol{\pi} \leftarrow \mathrm{diag}(\boldsymbol{\varphi})\, \mathbf{K}\, \mathrm{diag}(\boldsymbol{\psi})$
$w \leftarrow \langle \mathbf{C}, \boldsymbol{\pi} \rangle$
return $\boldsymbol{\pi}$, w

With a given inter-graph node-to-node cost matrix $\mathbf{C} \in \mathbb{R}^{n_1 \times n_2}$, where $\mathbf{C}_{i,j}$ denotes the cost of transporting a unit of mass from $u_i \in V^1$ to $v_j \in V^2$, OT finds the optimal coupling matrix $\boldsymbol{\pi}$ between G^1 and G^2 as follows:

$$\min_{\boldsymbol{\pi} \in \Pi(\boldsymbol{\mu}, \boldsymbol{\nu})} \langle \mathbf{C}, \boldsymbol{\pi} \rangle, \tag{2}$$

where $\langle \mathbf{C}, \boldsymbol{\pi} \rangle = \sum_i \sum_j \mathbf{C}_{i,j} \boldsymbol{\pi}_{i,j}$ is the Frobenius dot-product of $\mathbf{C}$ and $\boldsymbol{\pi}$, and the optimal value is the so-called Wasserstein Distance or Earth Mover's Distance. The lower center of Figure 3 shows a simple hand-crafted cost matrix $\mathbf{C}$ between graphs G^1 and G^2, defined as follows. Initially, we assume matrix $\mathbf{C}$ is all-zero. If the labels of u_i in G^1 and v_j in G^2 are different, we increase $\mathbf{C}_{i,j}$ by 1. Let d_i and d_j be the degrees of u_i and v_j. We further increase $\mathbf{C}_{i,j}$ by $|d_i - d_j|/2$, since the difference between degrees is associated with the number of edge insertions/deletions. The constant $1/2$ is used to avoid double-counting of an edge on its two endpoints. Solving OT over this cost matrix gives the coupling matrix shown in the lower-right corner of Figure 3, which indicates that u_2 is mapped to v_2, but u_1 can be mapped to either v_1 or v_3 with 50% probability each. This directly corresponds to the two optimal node matchings illustrated in Figure 3, which give a GED value of 2. A simple yet efficient method to solve Eq. (2) is by introducing an entropy regularization term into the optimization objective [57]:

$$\min_{\boldsymbol{\pi} \in \Pi(\boldsymbol{\mu}, \boldsymbol{\nu})} \langle \mathbf{C}, \boldsymbol{\pi} \rangle + \varepsilon H(\boldsymbol{\pi}), \tag{3}$$

where $H(\boldsymbol{\pi}) = \sum_i \sum_j \boldsymbol{\pi}_{i,j} \left(\log \boldsymbol{\pi}_{i,j} - 1 \right) = \langle \boldsymbol{\pi}, \log(\boldsymbol{\pi}) - 1 \rangle$ is the entropy function and $\varepsilon > 0$ is the regularization coefficient. Leveraging the duality theory [5] and strict convexity of Eq. (3), the unique solution can be solved by the Sinkhorn algorithm as shown in Algorithm 1 [14], which alternately updates the dual variables $\boldsymbol{\psi}$ and $\boldsymbol{\varphi}$ to fit the specified mass distribution $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$. For more details, please see Section B.1 [1].

Gromov-Wasserstein Discrepancy (GW). In practice, it is challenging to define a reasonable node-to-node cost matrix $\mathbf{C} \in \mathbb{R}^{n_1 \times n_2}$ without specified node embeddings for the two graphs G^1 and G^2. To address this issue, Gromov-Wasserstein discrepancy (GW) [30, 68] is introduced for graph alignment tasks [50, 60] as an extension of optimal transport. GW only requires the distances between nodes in the same graph, not inter-graph node distances. Specifically, GW

is the optimal value of the following optimization objective:

$$\min_{\boldsymbol{\pi}\in\Pi(\boldsymbol{\mu},\boldsymbol{\nu})} \sum_{i,j,k,l} (\mathbf{C}^1_{i,j} - \mathbf{C}^2_{k,l})^2 \boldsymbol{\pi}_{i,k}\boldsymbol{\pi}_{j,l}, \tag{4}$$

where $\mathbf{C}^1$ and $\mathbf{C}^2$ are the pre-defined cost matrices (e.g., adjacency matrices, all-pair shortest paths) of graphs G^1 and G^2, respectively. Here, we choose the typical option of $(\mathbf{C}^1_{i,j} - \mathbf{C}^2_{k,l})^2$ to measure the mismatch between two edges $(i,j) \in E^1$ and $(k,l) \in E^2$, but more choices can be found in [35]. Intuitively, $\boldsymbol{\pi}_{i,k}$ (resp. $\boldsymbol{\pi}_{j,l}$) represents the probability of matching nodes $u_i \in V^1$ and $v_k \in V^2$ (resp. $u_j \in V^1$ and $v_l \in V^2$), and Eq. (4) computes the expectation of edge-pair mismatch.

Let $\mathcal{L}(\mathbf{C}^1, \mathbf{C}^2)$ be the 4-th order tensor $\left((\mathbf{C}^1_{i,j} - \mathbf{C}^2_{k,l})^2\right)_{i,j,k,l}$ and $\mathcal{L} \otimes \boldsymbol{\pi}$ denotes the matrix $\left(\sum_{j,l} \mathcal{L}_{i,j,k,l}\boldsymbol{\pi}_{j,l}\right)_{i,k}$. Then the objective function can be rewritten into the following simple form:

$$\min_{\boldsymbol{\pi}\in\Pi(\boldsymbol{\mu},\boldsymbol{\nu})} \left\langle \boldsymbol{\pi}, \mathcal{L}(\mathbf{C}^1, \mathbf{C}^2) \otimes \boldsymbol{\pi} \right\rangle, \tag{5}$$

which can be solved with the conditional gradient algorithm [6, 52].

4 LEARNING-BASED METHOD: GEDIOT

In this section, we introduce **GEDIOT**, our neural network for GED computation based on inverse optimal transport. The training is an inverse process of OT to find (i.e., fit) the cost matrix given the ground-truth node coupling matrix of (G^1, G^2), $\boldsymbol{\pi}^*$, as supervision.

Motivation of introducing OT. Recall that a node matching satisfies the constraints in Eq. (1). Let us denote its feasible set by

$$U(\mathbf{1}_{n_1}, \mathbf{1}_{n_2}) = \left\{ \boldsymbol{\pi} \geq 0 \mid \boldsymbol{\pi}\mathbf{1}_{n_2} = \mathbf{1}_{n_1},\ \boldsymbol{\pi}^\top \mathbf{1}_{n_1} \leq \mathbf{1}_{n_2},\ \mathbf{1}_{n_1}^\top \boldsymbol{\pi}\mathbf{1}_{n_2} = n_1 \right\}. \tag{6}$$

Previous learning-based models predict GED and node matching via the interaction information of node/graph embeddings [2, 3, 36, 63], but they directly fit the predicted node matching with the ground-truth node coupling using binary cross-entropy loss, without considering all the constraints in $U(\mathbf{1}_{n_1}, \mathbf{1}_{n_2})$ during the training process.

We propose a novel neural architecture, GEDIOT, for GED computation and GEP generation, which predicts only the node-to-node cost matrix $\mathbf{C}$ from the interaction information of node/graph embeddings, and relies on OT to obtain the node matching from $\mathbf{C}$ so that all the constraints in $U(\mathbf{1}_{n_1}, \mathbf{1}_{n_2})$ are taken into consideration.

The training process is constructed as a bi-level optimization as formulated in Eq. (7), where the inner minimization computes the coupling matrix $\widehat{\boldsymbol{\pi}}$ satisfying the constraints in $U(\mathbf{1}_{n_1}, \mathbf{1}_{n_2})$ by solving an entropy-regularized OT problem that can be evaluated with our learnable Sinkhorn module, and the outer minimization calculates the difference between the coupling matrix and the ground truth to update the cost matrix $\widehat{\mathbf{C}}$ via backpropagation.

$$\begin{aligned} &\min_{\widehat{\mathbf{C}}}\ \mathcal{L}_m\left(\boldsymbol{\pi}^*, \widehat{\boldsymbol{\pi}}\right) + \mathcal{L}_v\left(GED^*, \widehat{GED}\right), \\ &\text{where } \widehat{\boldsymbol{\pi}} = \operatorname*{argmin}_{\boldsymbol{\pi}\in U(\mathbf{1}_{n_1},\mathbf{1}_{n_2})} \left\langle \widehat{\mathbf{C}}, \boldsymbol{\pi} \right\rangle + \varepsilon H(\boldsymbol{\pi}), \\ &\widehat{GED} = \left\langle \widehat{\mathbf{C}}, \widehat{\boldsymbol{\pi}} \right\rangle. \end{aligned} \tag{7}$$

Here, $\boldsymbol{\pi}^*$ and GED^* are the ground truth coupling matrix and GED of graph pair (G^1, G^2), respectively. Note that $\boldsymbol{\pi}^* \in \{0,1\}^{n_1\times n_2}$ is a one-to-one mapping and there are $(n_2 - n_1)$ full-zero columns. During test, computing GED is simply to solve the (inner) entropy-regularized OT problem, which is thus effective and interpretable.

In Eq. (7), $\widehat{\mathbf{C}} \in \mathbb{R}^{n_1\times n_2}$ is a learnable cost matrix that encodes the cost of matching each vertex pair across G^1 and G^2, and $\widehat{\boldsymbol{\pi}} \in \mathbb{R}^{n_1\times n_2}$ denotes the coupling matrix induced from $\widehat{\mathbf{C}}$ by minimizing the inner optimization problem. Recall from Section 3.1 that when relaxing the binary constraints of $\boldsymbol{\pi} \in \{0,1\}^{n_1\times n_2}$ to $\boldsymbol{\pi} \in [0,1]^{n_1\times n_2}$, Eq. (1) basically defines $\boldsymbol{\pi}_{i,j}$ to be the probability mass transported from $u_i \in V^1$ to $v_j \in V^2$, and the row $\boldsymbol{\pi}_i$ defines the distribution of transported probability mass from u_i to vertices of V^2. In Eq. (7), the GED value is approximated with $\left\langle \widehat{\mathbf{C}}, \widehat{\boldsymbol{\pi}} \right\rangle = \sum_{i=1}^{n_1}\sum_{j=1}^{n_2} \widehat{\mathbf{C}}_{i,j}\widehat{\boldsymbol{\pi}}_{i,j}$ since $\sum_{j=1}^{n_2} \widehat{\mathbf{C}}_{i,j}\widehat{\boldsymbol{\pi}}_{i,j}$ is the expected cost to transport mass from u_i, so $\left\langle \widehat{\mathbf{C}}, \widehat{\boldsymbol{\pi}} \right\rangle$ is the expected cost to transport all mass from G^1. In the special case when $\boldsymbol{\pi} \in \{0,1\}^{n_1\times n_2}$, $\left\langle \widehat{\mathbf{C}}, \widehat{\boldsymbol{\pi}} \right\rangle$ basically adds up the costs of transporting mass from u_i to its matched target in V^2 for all $u_i \in V^1$; and the first term in the outer minimization encourages a sparse $\widehat{\boldsymbol{\pi}}$ since the ground-truth $\boldsymbol{\pi}^*$ is sparse.

The objective of the outer optimization contains two terms designed for our two tasks: GED computation and GEP generation. Specifically, $\mathcal{L}_m$ is the matching loss for GEP generation, which we use Binary Cross-Entropy (BCE) loss between the ground truth $\boldsymbol{\pi}^*$ and the learned coupling matrix $\widehat{\boldsymbol{\pi}}$ that is then fed into the k-best matching framework [11, 36] as described in Section 4.5. $\mathcal{L}_v$ is the value loss for GED computation, which we adopt Mean-Squared Error (MSE) between the ground truth GED and the learned one obtained from both node and graph embeddings. The inner entropy-regularized OT of Eq. (7) provides an optimal coupling matrix with the current cost matrix $\widehat{\mathbf{C}}$. Then, the outer minimization fits the learned coupling matrix and GED to the ground truths to optimize the neural parameters in the model. Notably, we will formulate $\widehat{\mathbf{C}}$ further using the node features extracted from (G^1, G^2) by GNN (see Figure 2), so "$\min_{\widehat{\mathbf{C}}}$" in the outer optimization actually optimizes on the parameters of feature extraction network. More analysis of the process of Eq. (7) can be found in Section B.2 [1], where we delve into the gap between the learned and ground truth.

Model Overview. Figure 4 illustrates the network architecture of GEDIOT, including three main components: (1) node embedding component, (2) learnable OT component, and (3) graph discrepancy component. We highlight our new OT module with a red dashed frame in Figure 4. In the node embedding component, a GNN is employed to generate node embeddings with multiple graph convolution layers. For both graphs (G^1 and G^2), the node embeddings outputted by all layers are concatenated to aggregate information from neighbors of different hops (instead of only the last-hop neighbors), to alleviate the GNN over-smoothing issue [37, 42]. The concatenated embeddings are then passed through an MLP to derive the final node embeddings of the desired dimension d, denoted by $\mathbf{H}^1 \in \mathbb{R}^{n_1\times d}$ (for G^1) and $\mathbf{H}^2 \in \mathbb{R}^{n_2\times d}$ (for G^2). Subsequently, the learnable OT component extracts the node-matching matrix from the node embedding matrices through the OT process. This component includes a cost matrix layer that utilizes $\mathbf{H}^1$ and $\mathbf{H}^2$ to

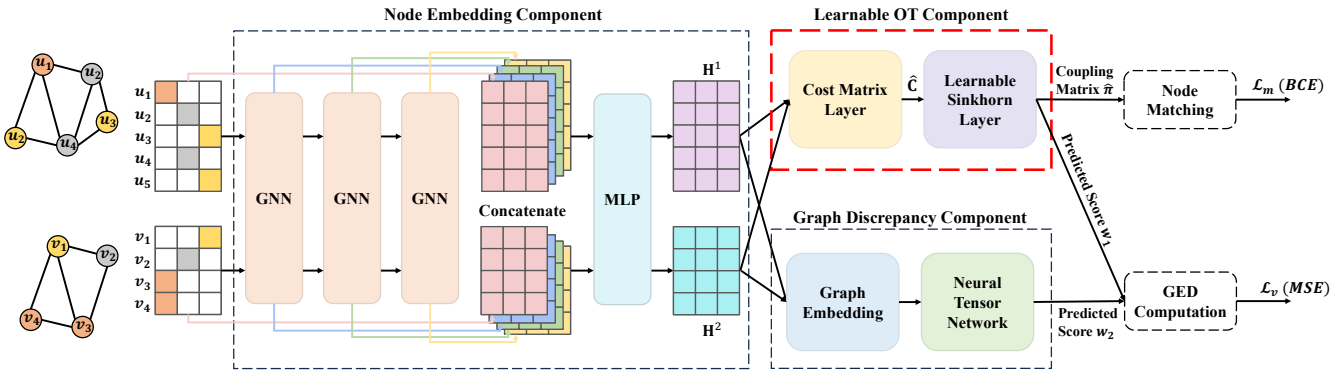


Figure 4: The architecture of GEDIOT

measure the node-to-node cost matrix $\widehat{\mathbf{C}}$, and a learnable Sinkhorn layer to read out the learned coupling matrix $\widehat{\boldsymbol{\pi}}$ via the Sinkhorn algorithm with a learnable regularization coefficient. This component also provide a GED score $w_1 = \left\langle \widehat{\mathbf{C}}, \widehat{\boldsymbol{\pi}} \right\rangle$. Additionally, a graph discrepancy component is employed to measure the edit operations of unmatched nodes (e.g., the $(n_2 - n_1)$ nodes in G^2) from the graph-to-graph level, which outputs another score w_2 for GED prediction. This component includes a neural network to generate the graph embeddings and a neural tensor network (NTN) [3] to calculate the predicted score w_2. Finally, scores w_1 and w_2 are combined to compute $GED(G^1, G^2)$. Note that all the sizes of parameters are user-defined (e.g., embedding dimension d) and independent of graph sizes (see more details in Section H.2 [1]). Figure 4 marks the learnable parts ($\mathbf{H}^1, \mathbf{H}^2, \widehat{\mathbf{C}}, \widehat{\boldsymbol{\pi}}$) in GEDIOT for ease of understanding.

4.1 Node Embedding Component

In this component, a GNN and an MLP are employed to capture the graph topology information and generate the final node embedding.

GNN Module. We adopt a siamese GNN to generate node embeddings by graph convolution operations, following previous graph similarity learning models [29, 36, 38]. Given the graph pair (G^1, G^2), nodes in both G^1 and G^2 are embedded with the shared network through node feature propagation and aggregation.

Concretely, Graph Isomorphism Network (GIN) [62] is adopted to capture the graph topology, since GIN has been shown to be as powerful as the Weisfeiler-Lehman (WL) graph isomorphism test in differentiating different graph structures [43]. For a graph $G = (V, E, L)$, we initialize the node embedding $\mathbf{h}^{(0)}(u)$ for $u \in V$ as the one-hot encoding of its label. If graphs are unlabeled, we set each initial node embedding as a constant number following previous works [3, 36]. In the i^{th} layer, the embedding of node u, denoted by $\mathbf{h}^{(i)}(u)$, is updated from itself and its neighbors as

$$\mathbf{h}^{(i)}(u) = \text{MLP}\left(\left(1 + \delta^{(i)}\right)\mathbf{h}^{(i-1)}(u) + \sum_{v \in \mathcal{N}(u)} \mathbf{h}^{(i-1)}(v)\right) \tag{8}$$

where $\delta^{(i)}$ is a learnable parameter of each layer and $\mathcal{N}(u)$ is the set of neighbors of u.

MLP Module. As the features propagate via GIN, higher-order graph structural information is fused into node embeddings, which may cause over-smoothed node embeddings at the last layer. Note that various GIN layers contain different orders of topological information: $\mathbf{h}^{(0)}(u)$ represents the features of u itself whereas $\mathbf{h}^{(i)}(u)$ contains the feature information from its i^{th}-hop neighbors. To obtain sufficiently rich node embeddings for more accurate GED computation, we concatenate the node embeddings from all GIN layers: $\mathbf{h} = \left[\mathbf{h}^{(0)} \| \mathbf{h}^{(1)} \| \cdots \| \mathbf{h}^{(k)}\right]$. The concatenated embedding $\mathbf{h}$ is then fed to an MLP to produce the final node embedding $\mathbf{H} \in \mathbb{R}^{n \times d}$:

$$\mathbf{H} = \text{MLP}(\mathbf{h}) = \text{MLP}\left(\left[\mathbf{h}^{(0)} \| \mathbf{h}^{(1)} \| \cdots \| \mathbf{h}^{(k)}\right]\right). \tag{9}$$

Suppose that the size of input $\mathbf{h}$ is $n \times D$, then we use an MLP with three dense layers of $D \times 2D$, $2D \times D$ and $D \times d$, respectively, to reduce the input $\mathbf{h}$ to the final node embeddings $\mathbf{H} \in \mathbb{R}^{n \times d}$.

4.2 Learnable OT Component

This component includes a cost matrix layer to extract the cost matrix from node embeddings $\mathbf{H}^1$ and $\mathbf{H}^2$ extracted by the node embedding component introduced in Section 4.1, and a learnable Sinkhorn layer to implement the inner entropy-regularized OT of Eq. (7) to generate the node matching from the cost matrix.

Cost Matrix Layer. This layer measures the node-to-node cost matrix $\widehat{\mathbf{C}} \in \mathbb{R}^{n_1 \times n_2}$ for the graph pair (G^1, G^2), by multiplying the final node embeddings $\mathbf{H}^1, \mathbf{H}^2$ with a trainable parameter matrix:

$$\widehat{\mathbf{C}} = f\left(\mathbf{H}^1 \mathbf{W} (\mathbf{H}^2)^\top\right),$$

where $\widehat{\mathbf{C}}_{i,j} = f(\mathbf{H}^1_i \mathbf{W} (\mathbf{H}^2_j)^T) = \sum_{k=1}^{d} \sum_{l=1}^{d} f(\mathbf{H}^1_{i,k} \mathbf{W}_{k,l} \mathbf{H}^2_{l,j})$, $\mathbf{W} \in \mathbb{R}^{d \times d}$ is a learnable interaction matrix, and f is an element-wise activation function. $\mathbf{W}_{k,l}$ can be regarded as a correlation weight for the k^{th} dimension in embedding $\mathbf{H}^1$ and the l^{th} dimension in embedding $\mathbf{H}^2$. In this work, we use tanh as the activation function:

$$\widehat{\mathbf{C}} = \tanh\left(\mathbf{H}^1 \mathbf{W} (\mathbf{H}^2)^\top\right). \tag{10}$$

Learnable Sinkhorn Layer. This layer is designed to solve the entropy-regularized OT numerically with the Sinkhorn algorithm in Algorithm 1. It takes the learned cost matrix $\mathbf{C}$ as input to generate the coupling matrix $\widehat{\boldsymbol{\pi}}$ and the predicted score w_1.

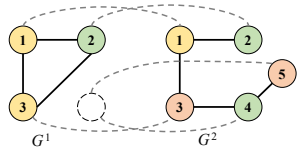


Figure 5: Illustration of the Dummy Supernode

Recall that the core process of the Sinkhorn algorithm is the alternate update of dual variables as shown in Lines 3 and 4 in Algorithm 1:

$$\boldsymbol{\psi} \leftarrow \boldsymbol{\nu} \oslash (\mathbf{K}^\top \boldsymbol{\varphi}), \quad \boldsymbol{\varphi} \leftarrow \boldsymbol{\mu} \oslash (\mathbf{K}\,\boldsymbol{\psi}),$$

where $\mathbf{K} = \exp(-\mathbf{C}/\varepsilon)$ is related to the learned cost matrix $\mathbf{C}$ and regularization coefficient ε, $\boldsymbol{\varphi}$ and $\boldsymbol{\psi}$ are the dual variables, and $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ are the pre-defined mass distributions (e.g., all-1 vectors). However, the constraint set $U(\mathbf{1}_{n_1}, \mathbf{1}_{n_2})$ in Eq. (6) has an inequality constraint $\boldsymbol{\pi}^\top \mathbf{1}_{n_1} \le \mathbf{1}_{n_2}$, which hinders applying the Sinkhorn algorithm directly, since the derivation of Sinkhorn as detailed in Section B.1 [1] only allows equality constraints (with inequality constraints, the dual formulation would introduce additional conditions that require the Lagrangian multipliers to be non-negative for $\boldsymbol{\pi}^\top \mathbf{1}_{n_1} \le \mathbf{1}_{n_2}$). To address this issue, we reconstruct an equivalent standard-form OT without the inequality constraint by extending the cost matrix $\mathbf{C}$ with a dummy row filled with 0 and redefining mass distributions as $\widetilde{\boldsymbol{\mu}}, \widetilde{\boldsymbol{\nu}}$ as follows:

$$\widetilde{\mathbf{C}} = \begin{bmatrix} \widehat{\mathbf{C}} \\ \mathbf{0}_{n_2}^\top \end{bmatrix}, \quad \widetilde{\boldsymbol{\mu}} = [\mathbf{1}_{n_1}^\top, n_2 - n_1]^\top, \quad \widetilde{\boldsymbol{\nu}} = \mathbf{1}_{n_2}.$$

Accordingly, we denote the new constraint set by

$$\Pi(\widetilde{\boldsymbol{\mu}}, \widetilde{\boldsymbol{\nu}}) = \left\{ \boldsymbol{\pi} \in \mathbb{R}^{(n_1+1)\times n_2} \mid \boldsymbol{\pi}\mathbf{1}_{n_2} = \widetilde{\boldsymbol{\mu}}, \quad \boldsymbol{\pi}^\top \mathbf{1}_{n_1+1} = \widetilde{\boldsymbol{\nu}}, \quad \boldsymbol{\pi} \ge 0 \right\},$$

and the standard-form OT is formulated as follows:

$$\min_{\boldsymbol{\pi} \in \Pi(\widetilde{\boldsymbol{\mu}}, \widetilde{\boldsymbol{\nu}})} \left\langle \widetilde{\mathbf{C}}, \boldsymbol{\pi} \right\rangle. \tag{11}$$

Intuitively, the dummy row in $\widetilde{\mathbf{C}}$ basically adds a dummy supernode in G^1 to match $(n_2 - n_1)$ nodes in G^2, as Figure 5 illustrates. We set the cost of matching the dummy supernode as 0 since our learnable OT component only accounts for the edit operations related to node matching (i.e., matching each of the n_1 node in G^1 towards G^2); while the edit cost induced by these $(n_2 - n_1)$ nodes in G^2 will be handled by the graph discrepancy component (see Section 4.3).

By adding entropy regularization $\varepsilon H(\boldsymbol{\pi})$ to Eq. (11), we can solve for $\boldsymbol{\pi}$ by the Sinkhorn algorithm. In each iteration, we update the dual variables $\widetilde{\boldsymbol{\psi}} \in \mathbb{R}^{n_2}$ and $\widetilde{\boldsymbol{\varphi}} \in \mathbb{R}^{n_1+1}$ alternately via:

$$\widetilde{\boldsymbol{\psi}} \leftarrow \widetilde{\boldsymbol{\nu}} \oslash \left(\widetilde{\mathbf{K}}^\top \widetilde{\boldsymbol{\varphi}}\right), \quad \widetilde{\boldsymbol{\varphi}} \leftarrow \widetilde{\boldsymbol{\mu}} \oslash \left(\widetilde{\mathbf{K}}\,\widetilde{\boldsymbol{\psi}}\right), \tag{12}$$

where $\widetilde{\mathbf{K}} \leftarrow \exp(-\widetilde{\mathbf{C}}/\varepsilon)$ is the element-wise exponential of $-\widetilde{\mathbf{C}}/\varepsilon$. We stack the two operations as feedforward layers to implement the iterations. When the iterative updates converge, $\widetilde{\boldsymbol{\pi}} = \mathrm{diag}(\widetilde{\boldsymbol{\varphi}})\,\widetilde{\mathbf{K}}\,\mathrm{diag}(\widetilde{\boldsymbol{\psi}})$, and the learned coupling matrix $\widehat{\boldsymbol{\pi}}$ is exactly $\widetilde{\boldsymbol{\pi}}$ with the last row removed [10]. The predicted score w_1 is $\left\langle \widehat{\mathbf{C}}, \widehat{\boldsymbol{\pi}} \right\rangle$, which estimates the optimal cost of edit operations induced by node matching.

A question remains: how to set a proper regularization coefficient ε? While a smaller ε leads to a closer approximation of the exact OT solution (without regularization). However, it also introduces a greater risk of numerical instability, which may lead to a divide-by-zero error. A straightforward approach is to set different ε for different datasets manually to achieve satisfactory performance. Nevertheless, the selection of an appropriate ε is costly.

Rather than fixing ε in advance for different datasets, we treat it as a learnable parameter and optimize it by gradient descent during training. The regularization coefficient ε is tuned for different datasets adaptively towards the optimal value, avoiding time-consuming manual adjustments. This is where the term "learnable" in the layer name originated (as Eq. (12) is parameter-free).

We also provide a concrete example from real-world datasets in Section D [1] to further illustrate our method.

4.3 Graph Discrepancy Component

Recall that before the learnable Sinkhorn layer, we add a dummy supernode to G^1; when the layer completes and outputs $\widehat{\pi}$, we remove the last row that corresponds to the dummy supernode. The learnable OT component captures only the edit operations induced by the node matching (from the node-to-node level), and some edit operations are not accounted for since $n_1 \le n_2$. We thus adopt another graph discrepancy component to supplement the unencoded information from the embedding of unmatched $(n_2 - n_1)$ nodes in G^2 from the graph-to-graph level. It includes a graph embedding layer to learn the embeddings of G^1 and G^2, and a neural tensor network (NTN) [3] that reads out the graph discrepancy information from the graph embeddings to enhance GED prediction.

Specifically, we first generate the graph-level embeddings with the node attentive mechanism [3]. Given a graph G (can be either G^1 or G^2) with node embedding matrix $\mathbf{H} \in \mathbb{R}^{n\times d}$ (can be either $\mathbf{H}^1$ or $\mathbf{H}^2$) extracted by our node embedding component, we first calculate the global graph context vector

$$\mathbf{h}_c = \tanh\left(\mathbf{W}_1 \left(\frac{1}{n}\left(\sum_{i=1}^{n} \mathbf{H}_i\right)^\top\right)\right), \tag{13}$$

which averages node features for all nodes of G followed by a non-linear transformation, where $\mathbf{W}_1 \in \mathbb{R}^{d\times d}$ is a learnable weight matrix and $\mathbf{H}_i$ is the i^{th} row of $\mathbf{H} \in \mathbb{R}^{n\times d}$. Then, the attention weight of each node v_i is computed as the inner product between $\mathbf{h}_c$ and $\mathbf{H}_i$ and normalized to the range $(0, 1)$, giving the node weight vector: $\mathbf{a} = \sigma(\mathbf{H}\mathbf{h}_c) \in \mathbb{R}^n$, where σ is the sigmoid function. Finally, the graph embedding $\mathbf{h}_G \in \mathbb{R}^d$ is computed as the weighted sum of node embeddings: $\mathbf{h}_G = \sum_{i=1}^{n} \mathbf{a}_i \mathbf{H}_i$.

Now that we have obtained graph embeddings for G^1 and G^2, we use an NTN to calculate the graph-to-graph interaction vector $\mathbf{s}(G^1, G^2) \in \mathbb{R}^L$ where L denotes the output dimension of NTN.

$$\mathbf{s}(G^1, G^2) = \mathrm{ReLU}\left(\mathbf{h}_{G^1}^\top \mathbf{W}_2^{[1:L]} \mathbf{h}_{G^2} + \mathbf{W}_3 [\mathbf{h}_{G^1}^\top \| \mathbf{h}_{G^2}^\top]^\top + \mathbf{b}\right), \tag{14}$$

where $\mathbf{W}_2^{[1:L]} \in \mathbb{R}^{L\times d\times d}$, $\mathbf{W}_3 \in \mathbb{R}^{L\times 2d}$ and $\mathbf{b} \in \mathbb{R}^L$ are learnable, and $\mathbf{h}_{G^1}^\top \mathbf{W}_2^{[1:L]} \mathbf{h}_{G^2}$ denotes the following L-dimensional vector:

$$\left[\mathbf{h}_{G^1}^\top \mathbf{W}_2^{(1)} \mathbf{h}_{G^2}, \quad \mathbf{h}_{G^1}^\top \mathbf{W}_2^{(2)} \mathbf{h}_{G^2}, \quad \ldots, \quad \mathbf{h}_{G^1}^\top \mathbf{W}_2^{(L)} \mathbf{h}_{G^2}\right]^\top,$$

where $\mathbf{W}_2^{(i)}$ is the i^{th} learnable weight matrix of $\mathbf{W}_2^{[1:L]}$.

Finally, we apply an MLP to progressively reduce the dimension of $\mathbf{s}(G^1, G^2)$ to a scalar, which outputs the predicted score w_2 to measure the edit operations of the unmatched nodes.

4.4 Model Training

GEDIOT is supervised by the ground-truth $GED^*(G^1, G^2)$ and the corresponding coupling matrix $\boldsymbol{\pi}^*$ for node matching between two graphs G^1 and G^2 during the training process. As shown in Eq. (7), the loss function consists of two parts: a value loss $\mathcal{L}_v$ to predict the GED and a matching loss $\mathcal{L}_m$ to predict the coupling matrix. The final loss function of GEDIOT is defined as

$$\mathcal{L} = \lambda \mathcal{L}_v + (1-\lambda)\mathcal{L}_m, \tag{15}$$

where we use a hyperparameter λ to balance $\mathcal{L}_v$ and $\mathcal{L}_m$.

Since the range of $GED(G^1, G^2)$ is too large to train a neural network effectively, we normalize the ground-truth GED to the range $[0, 1]$, and the normalized ground-truth GED is given by:

$$\mathrm{n}GED^*(G^1, G^2) = \frac{GED^*(G^1, G^2)}{\max(n_1, n_2) + \max(m_1, m_2)},$$

where the denominator on the right is the maximum number of edit operations that modify all nodes and edges to transform G^1 to G^2. To predict this normalized GED, we define the function:

$$\mathrm{score}(G^1, G^2) = \sigma(w_1 + w_2),$$

where $w_1 = \left\langle \widehat{\mathbf{C}}, \widehat{\boldsymbol{\pi}} \right\rangle$ is the predicted score from the learnable OT component, and w_2 is the predicted score from NTN [3]. Here, σ is the sigmoid function to ensure that the prediction is within $(0, 1)$.

We use MSE as the loss function for value:

$$\mathcal{L}_v = \left(\mathrm{score}(G^1, G^2) - \mathrm{n}GED^*(G^1, G^2)\right)^2,$$

and we fit the predicted coupling matrix with the ground-truth 0-1 matrix $\boldsymbol{\pi}^*$, by minimizing the binary cross-entropy loss (BCE) between the learned coupling matrix $\widehat{\boldsymbol{\pi}}$ and ground truth $\boldsymbol{\pi}^*$:

$$\mathcal{L}_m = \frac{1}{n_1 n_2} BCE\left(\boldsymbol{\pi}^* | \widehat{\boldsymbol{\pi}}\right),$$

where

$$\begin{aligned} BCE\left(\boldsymbol{\pi}^* | \widehat{\boldsymbol{\pi}}\right) &= \sum_{i=1}^{n_1}\sum_{j=1}^{n_2} \boldsymbol{\pi}^*_{i,j} \log \widehat{\boldsymbol{\pi}}_{i,j} + \left(1 - \boldsymbol{\pi}^*_{i,j}\right) \log\left(1 - \widehat{\boldsymbol{\pi}}_{i,j}\right) \\ &= \left\langle \boldsymbol{\pi}^*, \log(\widehat{\boldsymbol{\pi}}) \right\rangle + \left\langle 1 - \boldsymbol{\pi}^*, \log(1 - \widehat{\boldsymbol{\pi}}) \right\rangle. \end{aligned}$$

4.5 GEP Generation

Although we fit $\widehat{\boldsymbol{\pi}}$ to the ground-truth node matching $\boldsymbol{\pi}^* \in \{0, 1\}^{n_1 \times n_2}$, in practice when the model is trained, the learned coupling matrix $\widehat{\boldsymbol{\pi}}$ outputted by GEDIOT is not perfect but in the range $\boldsymbol{\pi}^* \in [0, 1]^{n_1 \times n_2}$ representing the confidence of node-to-node matching.

During inference, we adopt the k-best matching framework of [36] to generate $\widehat{GEP}(G^1, G^2)$ from the learned coupling matrix $\widehat{\boldsymbol{\pi}}$, which utilizes the solution space splitting method [11] to obtain a candidate set of k-best bipartite node matchings (based on the matching cost specified by the learned coupling matrix $\widehat{\boldsymbol{\pi}}$) and searches for the one with the shortest edit path as $\widehat{GEP}(G^1, G^2)$. Specifically, let S be a set of node matchings (Figure 6 shows two graphs G^1 and G^2 each having 3 nodes and all 6 possible node matchings in S), in which we can find the best and second-best node matchings according to the matching cost from $\widehat{\boldsymbol{\pi}}$, denoted by

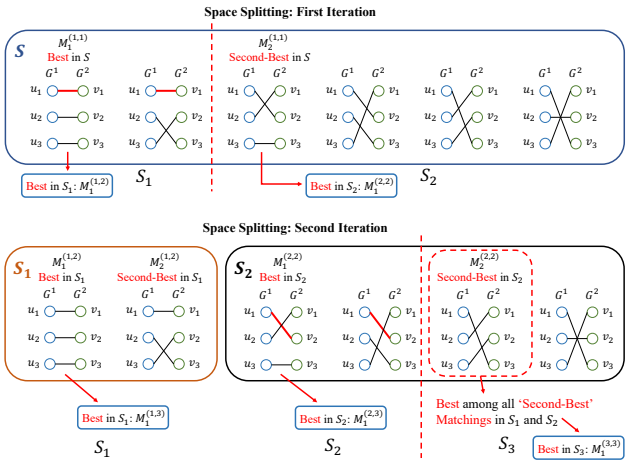


Figure 6: Example of Space Splitting of k-Best Matching

$M_1^{(1,1)}$ and $M_2^{(1,1)}$, respectively, in $O(n^3)$ time [11]. The first (resp. second) "1" in the superscript $(1, 1)$ means that the two matchings are in the first partition (resp. obtained in the first iteration). Let (u, v) be a node pair in $M_1^{(1,1)}$ but not in $M_2^{(1,1)}$ where $u \in V^1$ and $v \in V^2$. We can split S into two subspaces S_1 and S_2, such that a node matching of S is in S_1 if it contains (u, v), and otherwise it is in S_2. As shown in the upper part of Figure 6, u_1 matches v_1 in the best matching in S, but u_1 does not match v_1 in the second-best matching in S. Then, we split S into S_1 and S_2 according to whether u_1 matches v_1 in the first iteration. Note that $M_1^{(1,1)}$ (resp. $M_2^{(1,1)}$) becomes the best node matching in S_1 (resp. S_2) after splitting, which we denote as $M_1^{(1,2)}$ (resp. $M_1^{(2,2)}$). We also search the new second-best node matchings in S_1 and S_2, denoted by $M_2^{(1,2)}$ and $M_2^{(2,2)}$, respectively. The entire node matching space is partitioned by repeatedly selecting a partition to split in this manner. Assuming that there are t partitions and each has its best and second-best node matching $M_1^{(r,t)}$ and $M_2^{(r,t)}$, where $r = 1, 2, .., t$, the $(t+1)^{\text{th}}$ best node matching is $M_2^{(t^*,t)}$ of the partition t^* with the best 'second-best' node matching, so partition t^* is selected for splitting. Consider the lower part of Figure 6, where we assume the second-best matching $M_2^{(2,2)}$ in S_2 is better than the second-best matching $M_2^{(1,2)}$ in S_1. Since the best and second-best matchings in S_2 differ based on whether u_1 is matched to v_2, we further split S_2 accordingly. After splitting, the second-best matching $M_2^{(2,2)}$ in the original S_2 becomes the best matching in S_3, which we denote as $M_1^{(3,3)}$. This process is repeated until k partitions are reached, and GED lower-bound-based pruning [8, 18] is integrated to prune the unfruitful branches. Finally, $2k$ node matchings (2 from each partition) are collected as the candidate set to find the shortest edit path. More details can be found in Section C [1].

5 UNSUPERVISED METHOD: GEDGW

Currently, learning-based methods [2, 3, 36, 63] show the best performance of approximate GED computation, but they need ground truth for training set. This section presents our unsupervised optimization approach, **GEDGW**, that is able to achieve performance comparable to learning-based methods. GEDGW is based on the

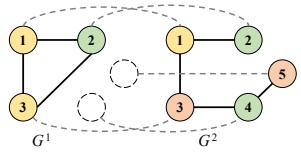


Figure 7: Illustration of Adding Dummy Nodes in G^1

Gromov-Wasserstein discrepancy, which bridges GED computation and optimal transport from an optimization perspective.

5.1 Formulation of GEDGW

Recall that the total edit operations that transform G^1 to G^2 can be determined with a given node matching between G^1 and G^2, where GED is the smallest one. Consequently, the GED computation of the graph pair (G^1, G^2) can be formulated as an optimization problem related to node matching.

Since there can be $(n_2 - n_1)$ nodes in G^2 that do not match any nodes in G^1, we add $(n_2 - n_1)$ dummy nodes in G^1 without any labels and edges following previous works [21, 41], as Figure 7 illustrates. This ensures that the two graphs have the same number of nodes without affecting the GED computation. For simplicity, we abuse the notations to still denote the graph after adding dummy nodes by G^1 and let $n = n_2 = \max\{n_1, n_2\}$ in this section.

Given a node matching, we can derive its induced edit operations into those on nodes and edges. Accordingly, GED computation can be derived by solving the following quadratic programming problem where the first (resp. second) term in the objective models the cost of node (resp. edge) edit operations. Section B.3 [1] provides a detailed illustration of the GEDGW formulation.

$$\min_{\boldsymbol{\pi}} \sum_{i,k} \mathbf{M}_{i,k}\boldsymbol{\pi}_{i,k} + \frac{1}{2}\sum_{i,j,k,l} (\mathbf{A}^1_{i,j} - \mathbf{A}^2_{k,l})^2 \boldsymbol{\pi}_{i,k}\boldsymbol{\pi}_{j,l}, \tag{16}$$
$$\text{s.t. } \boldsymbol{\pi}\mathbf{1}_n = \mathbf{1}_n, \quad \boldsymbol{\pi}^\top \mathbf{1}_n = \mathbf{1}_n, \quad \boldsymbol{\pi} \in \{0,1\}^{n\times n}.$$

Here, $\mathbf{M} \in \{0,1\}^{n\times n}$ is the node label matching matrix between nodes of G^1 and G^2, where $\mathbf{M}_{i,k} = 1$ if nodes $u_i \in V^1$ and $v_k \in V^2$ have the same label; otherwise $\mathbf{M}_{i,k} = 0$. Matrices $\mathbf{A}^1 \in \{0,1\}^{n\times n}$ and $\mathbf{A}^2 \in \{0,1\}^{n\times n}$ are the adjacency matrices of G^1 and G^2, respectively. The factor $\frac{1}{2}$ in the second term is to avoid the double counting of $\boldsymbol{\pi}_{i,k}\boldsymbol{\pi}_{j,l}$ and $\boldsymbol{\pi}_{j,l}\boldsymbol{\pi}_{i,k}$ since the graphs are undirected.

More concretely, the first linear term of Eq. (16) measures the cost of the node edit operations, including (1) The insertion/deletion of a node as indicated by matching a node in G^2 and a dummy node in G^1, and (2) the relabeling operation as represented by matching a node in G^2 to an original node in G^1 whose labels are different.

The second quadratic term measures the cost of edge insertion/deletion since each element $(\mathbf{A}^1_{i,j} - \mathbf{A}^2_{k,l})^2\boldsymbol{\pi}_{i,k}\boldsymbol{\pi}_{j,l}$ in the sum measures whether edge $(u_i, u_j) \in E^1$ and edge $(v_k, v_l) \in E^2$ exist simultaneously when u_i matches v_k and u_j matches v_l.

After relaxing the binary constraint to allow elements of $\boldsymbol{\pi}$ to take values in $[0, 1]$, the solution $\boldsymbol{\pi}$ represents the confidence of node-to-node matching between G^1 and G^2. Note that Eq. (16) with relaxation on binary variables can be regarded as a linear combination of optimal transport (OT) and Gromov-Wasserstein Discrepancy (GW), where the first linear term models the edit operations on nodes as an OT problem (the right part of Figure 9 in Section B.3 [1]) and the second quadratic term models the edit operations on edges as a GW problem (the left part of Figure 9 in Section B.3 [1]). So we call this method GEDGW. The optimization problem of GEDGW is reformulated as follows:

$$\min_{\boldsymbol{\pi}\in\Pi(\mathbf{1}_n,\mathbf{1}_n)} \langle \boldsymbol{\pi}, \mathbf{M}\rangle + \frac{1}{2}\left\langle \boldsymbol{\pi}, \mathcal{L}(\mathbf{A}^1, \mathbf{A}^2)\otimes\boldsymbol{\pi}\right\rangle \tag{17}$$

where $\Pi(\mathbf{1}_n, \mathbf{1}_n) = \left\{\boldsymbol{\pi} \in \mathbb{R}^{n\times n} \mid \boldsymbol{\pi}\mathbf{1}_n = \mathbf{1}_n,\ \boldsymbol{\pi}^\top\mathbf{1}_n = \mathbf{1}_n, \boldsymbol{\pi} \geq 0\right\}$ is the feasible set of coupling matrices. We exploit the Conditional Gradient (CG) method [6, 52] to solve GEDGW, which is presented in detail in Section B.4 [1]. An example in Section D [1] further illustrates our GEDGW method.

5.2 Further Improvement by Ensembling

Recall that GEDGW and GEDIOT model the GED computation from two different perspectives via optimal transport. To achieve better performance, we combine these two OT-based methods into an ensemble **GEDHOT** (GED with Hybrid Optimal Transport), which combines the results from GEDGW and GEDIOT to enhance the performance of GED computation and GEP generation during test.

Specifically, given an input of graph pair (G^1, G^2), we run GEDGW and GEDIOT to get the GEDs and coupling matrices denoted by $\widehat{GED}_{\text{GW}}(G^1, G^2)$ and $\widehat{\boldsymbol{\pi}}_{\text{GW}}$, $\widehat{GED}_{\text{IOT}}(G^1, G^2)$ and $\widehat{\boldsymbol{\pi}}_{\text{IOT}}$, respectively. Since GED is the minimum number of edit operations, we choose the smaller of $\widehat{GED}_{\text{GW}}(G^1, G^2)$ and $\widehat{GED}_{\text{IOT}}(G^1, G^2)$ as $\widehat{GED}(G^1, G^2)$.

$$\widehat{GED}(G^1, G^2) = \min\left\{\widehat{GED}_{\text{GW}}(G^1, G^2), \widehat{GED}_{\text{IOT}}(G^1, G^2)\right\}.$$

For GEP generation, we generate the best edit paths via the k-best matching framework [36] from $\widehat{\boldsymbol{\pi}}_{\text{GW}}$ and $\widehat{\boldsymbol{\pi}}_{\text{IOT}}$, respectively, and then choose the shorter one.

5.3 Time Complexity Analysis

Due to space limitation, we provide a comprehensive analysis of the time complexity of our proposed methods in Section E [1].

In a nutshell, for GEDIOT, since the model training can be done offline given a graph dataset, we consider the computation cost of its forward propagation, the time complexity of which is given by

$$O\left(N(md + nd^2 + nN^2d^2) + Ld^2 + nd^2 + n^2d + Mn^2\right) \approx O(n^2),$$

where we assume that the number of GNN layers is N, the dimension of hidden layers of GNN and MLP is d, the output dimension of NTN is L, $n = n_2$, $m = \max(m_1, m_2)$ and M denotes the number of iterations of the Sinkhorn algorithm. As the hyperparameters are regarded as constants, the time complexity can be simplified to $O(n^2)$, which is the same as previous learning-based methods in the worst case. For GEP generation via the k-best matching framework, the time complexity is $O(kn^3)$.

For GEDGW, we use the CG method [6, 52] (see Section B.4 [1] for details) to solve Eq. (17). The time complexity of CG is bounded by $O(Kn^3)$, where K is the number of iterations. For GEDHOT, the time complexity is $O(n^2 + Kn^3) = O(Kn^3)$, and the time complexity to generate GEP using the k-best matching framework is $O(kn^3)$. Note that both GEDGW and GEDHOT have the same time complexity as the classical heuristic algorithms (e.g. Hungarian and VJ).

6 EXPERIMENT

This section evaluates the performance of our proposed methods and compares with existing approximate GED computing methods.

Table 2: Statistics of Graph Datasets

$\mathcal{D}$	$\lvert\mathcal{D}\rvert$	$\lvert V\rvert_{avg}$	$\lvert E\rvert_{avg}$	$\lvert V\rvert_{max}$	$\lvert E\rvert_{max}$	$\lvert L\rvert$
AIDS	700	8.9	8.8	10	14	29
LINUX	1000	7.6	6.9	10	13	1
IMDB	1500	13	65.9	89	1467	1

6.1 Datasets

We use three real-world graph datasets: AIDS, Linux, and IMDB. Table 2 summarizes their statistics including the number of graphs ($|\mathcal{D}|$), the average number of nodes ($|V|_{avg}$) and edges ($|E|_{avg}$), the maximum number of nodes ($|V|_{max}$) and edges ($|E|_{max}$), and the number of labels ($|L|$). For graph pairs with no more than 10 nodes, we use the A* algorithm [41] to generate the exact ground truth, and for the remaining graphs with more than 10 nodes, we use the ground-truth generation technique in [2, 36] to generate 100 synthetic graphs for each graph. For each dataset, we sample 60% graphs and pair every two of them to create graph pairs of the training set. As for the test set, we sample 20% graphs; for each selected graph, 100 graphs are randomly chosen from the training graphs to generate 100 graph pairs for the test set. The validation set is formed in the same manner as the test set. Section F.1 [1] describes the datasets, data preprocessing, and dataset partitions in detail.

6.2 Compared Methods

Recall that GEDGW is a non-learning approximation algorithm, GEDIOT is a learning-based method, and GEDHOT is a combination of both. We compare them with the classical approximation algorithms and learning-based methods.

Classical Algorithms. We select three representative classical approximate algorithms for GED computation. (1) **Hungarian** [40] is based on the Hungarian method for weighted graph matching which takes cubic time. (2) **VJ** [16] is based on bipartite graph matching which takes cubic time. (3) **Classic** runs both Hungarian and VJ to find the GEPs, and takes the better GEP. We do not include the heuristic A*-beam algorithm [32] since Noah in the paragraph below is an optimized version of A*-beam with better performance.

Learning-based Methods. We choose four state-of-the-art learning-based methods for GED computation. (1) **SimGNN** [3] is the very first learning method applying GNN for GED computation. (2) **Noah** and **GPN** [63]. Noah employs the well-designed graph path network (GPN) to optimize the search direction of the A*-Beam algorithm [32] to find GEP. Additionally, GPN can also be utilized independently for GED computation only. (3) **TaGSim** [2] categories edit operations to four different types, and learns the number of edit operations in each type to achieve competitive GED approximation. (4) **GEDGNN** [36] is the latest method for both GED computation and GEP generation. See Section 2 for a detailed review.

Our Methods. We propose **GEDIOT**, **GEDGW**, and **GEDHOT** for comparison. The detailed setup can be found in Section F.2 [1].

6.3 Evaluation metrics

We consider four kinds of metrics to evaluate the performance, which have been widely used [2, 3, 36, 63].

Metrics for GED Computation. (1) **Mean Absolute Error** (MAE) measures the average absolute error between ground-truth GEDs and approximate GEDs. For a graph pair (G^1, G^2), it is formulated as $|GED^*(G^1, G^2) - \widehat{GED}(G^1, G^2)|$. (2) **Accuracy** measures the ratio of approximate GEDs that equal the ground-truth GEDs after rounding to the nearest integer. (3) **Feasibility** measures the ratio that the approximate GEDs are no less than the ground-truth GEDs, so that a GEP of this length is feasible (i.e., can be found).

Metrics for Ranking. These metrics measure the matching ratio between the ranking results of the approximate GED and the ground truth. They include (4) **Spearman's Rank Correlation Coefficient** (ρ). (5) **Kendall's Rank Correlation Coefficient** (τ). (6) **Precision at** k ($p@k$). The first two metrics focus on global ranks while the last focuses on top k. We use $p@10$ and $p@20$.

Metrics for Path. These metrics measure how well the generated edit path GEP matches the ground-truth GEP^*. They include (7) $Recall = \frac{|GEP \cap GEP^*|}{|GEP^*|}$, (8) $Precision = \frac{|GEP \cap GEP^*|}{|GEP|}$, and (9) F1 score defined as $F1 = 2 \cdot \frac{Recall \cdot Precision}{Recall + Precision}$.

Metrics for Efficiency. (10) Running Time ($sec/100p$), where p = "pairs". It records the time for every 100 graph pairs during test.

6.4 Experimental Results

We evaluate the performance of various methods for both GED computation and GEP generation.

Performance of GED Computation. We first compare our proposed methods (i.e., GEDGW, GEDIOT, and GEDHOT) with the six baselines mentioned in Section 6.2 (Hungarian and VJ are dominated by Classic and are hence omitted due to space limit). We categorize the methods into three types: learning-based methods, non-learning methods, and hybrid methods. We count Noah also as a hybrid method since it combines GPN with A*-Beam.

Table 3 reports the results. We can see that among the learning-based methods, GEDGNN achieves the best performance on all three datasets for value, ranking, and feasibility metrics. Meanwhile, GEDIOT significantly outperforms GEDGNN (as well as the other learning-based baselines) in terms of value and ranking metrics with comparable time consumption. For instance, compared with the state-of-the-art method GEDGNN, the MAE of our proposed GEDIOT is 23.9%, 63.8%, 20.5% smaller on AIDS, Linux, and IMDB, respectively; also, on AIDS, the accuracy of GEDGNN and our GEDIOT is 40.4% and 49.7%, respectively. Note that TaGSim is the most time-efficient (e.g., on AIDS, the training time for an epoch of TaGSim is 151 s, while that of GEDIOT is 581 s) but cannot return high-quality results. We train TaGSim for more epochs so that the total training time of TaGSim is roughly equal to GEDIOT, and the results are similar to that reported in Table 3. On AIDS, Linux, and IMDB, the MAE and accuracy of TaGSim with more training time are 0.816 and 37.9%, 0.316 and 70.6%, 4.962 and 11.4% respectively, which are still worse than our model. It demonstrates that our experimental setup is sufficient to converge.

For the non-learning methods, Classic and GEDGW, it is obvious that GEDGW achieves much better performance on all the value and ranking metrics with up to 14× faster computational speed. More surprisingly, on AIDS and IMDB, GEDGW even achieves a higher accuracy than the state-of-the-art learning-based method GEDGNN. Note that the training phase for the learning-based methods always

Table 3: Performance Evaluations of GED Computation.

Datasets	Methods	Value		Ranking				Feasibility ↑	Time ↓
		MAE ↓	Accuracy ↑	ρ ↑	τ ↑	p@10 ↑	p@20 ↑		($sec/100p$)
AIDS	SimGNN	0.880	34.7%	0.841	0.704	0.632	0.741	61.5%	0.279
	GPN	0.924	35.6%	0.816	0.680	0.606	0.713	66.5%	0.245
	TaGSim	0.807	37.4%	0.862	0.730	0.669	0.754	66.2%	**0.087**
	GEDGNN	0.763	40.4%	0.870	0.742	0.716	0.774	72.1%	0.307
	GEDIOT	0.581	49.7%	0.922	0.813	0.814	0.853	73.9%	0.318
	Classic	6.594	3.3%	0.529	0.418	0.545	0.614	**100%**	1.463
	GEDGW	1.247	41.2%	0.789	0.670	0.752	0.765	**100%**	0.430
	Noah	3.164	5.6%	0.704	0.585	0.681	0.721	**100%**	161.023
	GEDHOT	**0.484**	**59.3%**	**0.936**	**0.838**	**0.863**	**0.885**	73.9%	0.745
Linux	SimGNN	0.408	63.3%	0.939	0.856	0.911	0.916	75.6%	0.278
	GPN	0.142	87.1%	0.959	0.896	0.947	0.974	90.5%	0.265
	TaGSim	0.346	69.6%	0.937	0.859	0.888	0.910	85.9%	**0.069**
	GEDGNN	0.094	91.6%	0.961	0.897	0.980	0.976	95.9%	0.282
	GEDIOT	0.034	97.2%	0.969	0.911	0.992	0.995	98.5%	0.326
	Classic	2.471	21.5%	0.785	0.707	0.762	0.835	**100%**	0.915
	GEDGW	1.198	48.1%	0.817	0.705	0.827	0.811	**100%**	0.382
	Noah	1.736	8.4%	0.870	0.798	0.906	0.936	**100%**	71.646
	GEDHOT	**0.026**	**97.9%**	**0.970**	**0.915**	**0.994**	**0.997**	98.5%	0.754
IMDB	SimGNN	1.191	40.4%	0.735	0.648	0.759	0.799	68.1%	0.291
	GPN	1.614	28.2%	0.742	0.668	0.669	0.708	34.3%	0.229
	TaGSim	5.247	14.8%	0.496	0.441	0.666	0.699	47.7%	**0.095**
	GEDGNN	0.735	59.6%	0.859	0.781	0.838	0.856	80.2%	0.305
	GEDIOT	0.584	65.3%	0.930	0.858	0.902	0.912	78.6%	0.347
	Classic	12.980	62.8%	0.764	0.718	0.837	0.831	**100%**	3.483
	GEDGW	0.818	**83.0%**	0.926	0.896	0.968	0.951	93.6%	0.247
	Noah	10.467	38.4%	0.717	0.688	0.755	0.795	**100%**	4816.67
	GEDHOT	**0.506**	69.9%	**0.956**	**0.899**	**0.978**	**0.972**	73.1%	0.607

↑: higher is better, ↓: lower is better **Bold**: best, Underline: runner-up.

Table 4: Performance Evaluations of GEP Generation.

Datasets	Methods	Value		Ranking				Path			Time ↓
		MAE ↓	Accuracy ↑	ρ ↑	τ ↑	p@10 ↑	p@20 ↑	*Recall* ↑	*Precision* ↑	$F1$ ↑	($sec/100p$)
AIDS	Classic	6.594	3.3%	0.529	0.418	0.545	0.614	0.572	0.345	0.423	**1.752**
	Noah	3.164	5.6%	0.704	0.585	0.681	0.721	0.609	0.505	0.548	163.153
	GEDGNN	1.503	42.2%	0.795	0.690	0.849	0.838	0.715	0.646	0.675	56.439
	GEDIOT	1.266	49.9%	0.814	0.715	0.881	0.858	0.756	0.692	0.719	57.857
	GEDGW	0.829	53.2%	0.862	0.774	0.842	0.858	0.715	0.675	0.692	57.102
	GEDHOT	**0.440**	**71.2%**	**0.923**	**0.864**	**0.951**	**0.935**	**0.809**	**0.786**	**0.796**	112.161
Linux	Classic	2.471	21.5%	0.785	0.707	0.762	0.835	0.770	0.541	0.623	**0.954**
	Noah	1.736	8.4%	0.870	0.798	0.906	0.936	0.851	0.772	0.802	73.018
	GEDGNN	0.156	93.5%	0.970	0.954	0.987	0.980	0.917	0.904	0.909	19.317
	GEDIOT	0.114	95.4%	0.976	0.965	0.988	0.987	0.924	0.914	0.918	19.514
	GEDGW	0.591	72.2%	0.898	0.836	0.925	0.887	0.837	0.780	0.802	26.788
	GEDHOT	**0.033**	**98.4%**	**0.994**	**0.990**	**0.992**	**0.996**	**0.928**	**0.924**	**0.926**	47.523
IMDB	Classic	12.980	62.8%	0.764	0.718	0.837	0.831	0.833	0.628	0.654	**3.663**
	Noah	10.467	38.4%	0.717	0.688	0.755	0.795	0.845	0.670	0.682	4864.38
	GEDGNN	3.574	79.6%	0.888	0.859	0.924	0.924	0.907	0.808	0.826	93.893
	GEDIOT	3.638	82.0%	0.903	0.878	0.923	0.928	0.907	0.816	0.831	93.091
	GEDGW	0.374	93.2%	0.969	0.955	0.988	0.983	0.763	0.736	0.744	81.948
	GEDHOT	**0.254**	**95.0%**	**0.983**	**0.972**	**0.995**	**0.993**	**0.946**	**0.927**	**0.933**	170.412

↑: higher is better, ↓: lower is better **Bold**: best, Underline: runner-up.

takes several hours, while GEDGW does not need that phase and directly outputs results within a second. Moreover, all the learning-based methods need the ground truths of GED and node matching for model training. The performance of GEDGW suggests that it is possible to approximate high-quality GEDs in a non-learning way.

Finally, for the two hybrid methods, Noah and GEDHOT, we can see that compared to Noah, the MAE of our GEDHOT is up to 20× smaller with hundreds of times smaller computational time (recall that Noah runs the expensive A* algorithm). In addition, in Table 3, GEDHOT clearly outperforms all the other methods, followed by the proposed GEDIOT and GEDGW with a consistent second-best performance on all three datasets. For instance, on AIDS, the accuracies of GEDIOT, GEDGW and GEDHOT are 49.7%, 41.2%, and 59.3% respectively, while that of GEDGNN is only 40.4%. This shows that GEDHOT can combine the merits of both GEDIOT and GEDGW to get better results.

Performance of GEP Generation. We next compare the performance of GEP generation of the methods above. Note that among the learning-based baselines, Noah and GEDGNN are the only two that can generate GEP, so we include Noah, GEDGNN, and Classic as baselines in Table 4 for comparison. We can see that Classic takes the shortest computational time, but the MAE is several times larger

Table 5: GED Computation of Unseen Graph Pairs.

Datasets	Methods	Value		Ranking				Feasibility ↑	Time ↓
		MAE ↓	Accuracy ↑	ρ ↑	τ ↑	p@10 ↑	p@20 ↑		($sec/100p$)
AIDS	SimGNN	0.925	34.4%	0.808	0.668	0.631	0.731	63.6%	0.284
	GPN	1.038	33.4%	0.771	0.631	0.578	0.683	64.5%	0.235
	TaGSim	0.880	34.8%	0.832	0.694	0.674	0.739	66.0%	**0.093**
	GEDGNN	0.826	38.0%	0.831	0.696	0.702	0.750	69.4%	0.298
	GEDIOT	**0.684**	**44.5%**	**0.897**	**0.776**	**0.791**	**0.835**	**71.3%**	0.313
Linux	SimGNN	0.399	63.2%	0.953	0.877	0.934	0.918	77.6%	0.288
	GPN	0.147	86.6%	0.973	0.916	0.948	0.967	90.5%	0.279
	TaGSim	0.347	69.3%	0.951	0.877	0.878	0.905	87.4%	**0.079**
	GEDGNN	0.122	89.8%	0.965	0.904	0.968	0.973	95.1%	0.291
	GEDIOT	**0.051**	**96.1%**	**0.976**	**0.925**	**0.983**	**0.990**	**97.6%**	0.336
IMDB	SimGNN	1.236	39.3%	0.733	0.642	0.755	0.801	67.4%	0.307
	GPN	1.635	27.7%	0.741	0.664	0.670	0.710	33.9%	0.226
	TaGSim	4.811	15.4%	0.501	0.445	0.665	0.700	47.2%	**0.107**
	GEDGNN	0.743	59.2%	0.858	0.777	0.842	0.857	**79.8%**	0.294
	GEDIOT	**0.595**	**65.5%**	**0.925**	**0.850**	**0.903**	**0.913**	78.5%	0.353

↑: higher is better, ↓: lower is better **Bold**: best, Underline: runner-up.

than other methods. Among the other four methods, similar to GED results in Table 3, GEDHOT achieves the best performance for value and ranking metrics on all the three datasets. For example, on AIDS, the accuracy of GEDGNN and GEDHOT is 42.2% and 71.2% respectively; also, on Linux, GEDHOT obtains 4.7×, 17.9×, 3.5× smaller MAE compared with GEDGNN, GEDGW, GEDIOT, respectively. Moreover, the second-best is either GEDIOT or GEDGW.

Note that the computational time of GEDHOT is about twice as large as the time of the other three methods except for Classic. Even if a smaller time cost is preferred, our proposed GEDGW and GEDIOT are preferred compared to GEDGNN, which is the latest method for GEP generation. It is worth noting that on AIDS and IMDB, the non-learning method GEDGW even achieves 1.8× and 9.6× smaller MAE than the learning-based method GEDGNN.

Regarding path quality metrics, Recall, Precision, and F1 score, Table 4 shows that GEDHOT consistently performs the best, and GEDIOT is consistently the second-best.

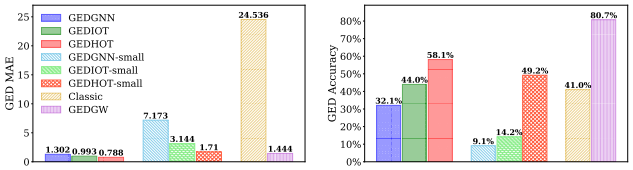
We also study the contribution of GEDIOT and GEDGW for the ensemble method GEDHOT. For example, on AIDS, for GED computation, most graph pairs (80.8%) use the results from GEDIOT instead of GEDGW. For GEP generation, 63.1% of the graph pairs use the results from GEDIOT, and 36.9% of the graph pairs use the results from GEDGW. More results can be found in Section G.2 [1].

Note that GED is a distance metric, satisfying the triangle inequality. Without loss of generality, we conduct experiments on AIDS and Linux to evaluate the fraction of triangle inequality violations in the predicted GEDs. The results shown in Section G.2 [1] indicate that our methods satisfy this property in most cases (> 95%).

6.5 Generalizability

Since all the learning-based methods require training data supervision, it is interesting to explore how they generalize beyond the training data distribution, including our GEDIOT model.

Modeling GED Computation of Unseen Graphs. Recall that we prepared the test set by sampling 100 training graphs for each test graph, which models the graph similarity search task. To evaluate the generalizability, now we instead sample 100 test graphs (rather than training graphs) for each test graph, so that both graphs in a graph pair of the test set are unseen during training.



(a) IMDB - MAE (b) IMDB - Accuracy

Figure 8: Generalizability for Large Unseen Graphs on IMDB

Table 5 shows the results of the five learning-based methods, where GEDIOT still significantly outperforms GEDGNN and the others in terms of value and ranking metrics. For example, on Linux, the MAE of GEDIOT is 2.4× smaller than GEDGNN, and the accuracy reaches 96.1% while that of GEDGNN is below 90%.

Compared with the results in Table 3, the performance of all methods decreases since the test set is more challenging. Nevertheless, the amount of degradation is not significant. For example, the accuracy of GEDIOT decreases by 10.5% and 1.1% on AIDS and Linux, respectively, which demonstrates its generalizability.

Generalization to Large Unseen Graphs. Ground truth is crucial for supervised learning-based methods. In GED computation, ground truth is difficult to obtain for large graphs due to the NP-hardness of the problem. For instance, there are plenty of graphs with more than 10 nodes in the IMDB dataset, and it is too expensive to calculate the GEDs of these graph pairs with exact algorithms. Therefore, we consider training the model only with small graphs and testing the performance of the learning-based methods on large unseen graphs. More concretely, we select the graph pairs from the training set of IMDB that are formed by the small graphs (at most 10 nodes) to build a new training set. All the methods trained on it are appended with the "-small" suffix, i.e., **GEDGNN-small**, **GEDIOT-small** and **GEDHOT-small**. To evaluate generalizability, we also construct a new test set, which consists of the graph pairs from the test set of IMDB that are formed by the large graphs (more than 10 nodes). The results are shown in Figure 8, where GEDGNN, GEDIOT, and GEDHOT denote the methods trained on the complete training set of IMDB. We can see that models trained on small graphs have an inferior performance compared to training on complete training set. However, GEDHOT-small and GEDIOT-small are

Table 6: Ablation Study of GEDIOT Components.

Method	AIDS						Linux					
	MAE ↓	Accuracy ↑	ρ ↑	τ ↑	p@10 ↑	p@20 ↑	MAE↓	Accuracy ↑	ρ ↑	τ ↑	p@10 ↑	p@20 ↑
GEDIOT	0.581	49.7%	0.922	0.813	0.814	0.853	0.034	97.2%	0.969	0.911	0.992	0.995
GEDIOT (w/ GCN)	0.578	49.1%	0.917	0.805	0.794	0.838	0.064	93.8%	0.967	0.909	0.980	0.985
GEDIOT (w/o MLP)	0.854	35.9%	0.814	0.677	0.599	0.678	0.158	85.9%	0.958	0.889	0.934	0.956
GEDIOT (w/o Cost)	0.794	38.4%	0.870	0.741	0.692	0.765	0.132	87.5%	0.964	0.901	0.953	0.966
GEDIOT (w/o learnable ε)	0.767	38.5%	0.906	0.790	0.801	0.831	0.063	94.7%	0.967	0.910	0.988	0.991

still significantly better than GEDGNN-small in terms of MAE and accuracy. Notably, GEDGW achieves the highest accuracy of 80.7% since it is unsupervised, demonstrating its robustness as compared to learning-based methods that face generalizability challenges.

We further discuss how the generalizability is impacted when synthesizing test graph pairs with larger GEDs. Similarly, GEDGW achieves the best performance and our neural model outperforms GEDGNN. Detailed results are shown in Figure 12 in Section G.1 [1].

We notice that the state-of-the-art methods Nass [22] and AStar-BMao [9] for graph similarity search (introduced in Section 2) can be applied for exact GED computation by setting the similarity threshold to infinity. As indicated in [36], exact methods suffer from huge computation costs when the graph size increases. We compare our method GEDIOT with Nass and AStar-BMao on two large real-world datasets. The detailed setup and the running time of the three methods can be found in Section G.3 [1]. We find that the running time of the two exact methods Nass and AStar-BMao is quite sensitive w.r.t. the graph size and the GED value. Our method GEDIOT shows a consistent advantage compared to the two exact algorithms, particularly for larger graphs and GEDs, since the time complexity of GEDIOT is only $O(n^2)$, whereas AStar-BMao and Nass are still exponential-time algorithms.

We also generate synthetic power-law graphs of various sizes (from 50 to 400 nodes). The results are reported in Section G.4 [1], where we find that the GED relative error of our GEDGW and GEDHOT is nearly 0 while that of GEDGNN is always almost 2, and the computational time of learning-based methods is orders of magnitude faster than the exact algorithms.

6.6 Ablation and Parameter Study

We conduct ablation study to verify the effectiveness of various modules in GEDIOT, and to show the robustness of GEDIOT w.r.t. hyperparameters by varying their values.

Effect of Modules in GEDIOT. In this ablation study, we modify GEDIOT into four variants and compare their performance with GEDIOT. Table 6 shows the results, where we use "w/ GCN" to denote the variant substituting GIN with GCN in GEDIOT, and use "w/o MLP", "w/o Cost", and "w/o learnable ε" to denote GEDIOT that removes the MLP in the node embedding component, that replaces cost matrix module in the learnable OT component with $\mathbf{H}^1(\mathbf{H}^2)^\top$ (i.e., to model node interactions with simple inner product of their embeddings), and that fixes the regularization coefficient ε in the learnable Sinkhorn layer as $\varepsilon_0 = 0.05$, respectively.

As Table 6 shows, replacing or removing a module in GEDIOT can significantly degrade the performance of both value and ranking metrics, which verifies the effectiveness of our proposed components for GED computation. For instance, on AIDS, if fixing the regularization coefficient ε, the accuracy decreases from 49.7% to 38.5% and MAE increases from 0.581 to 0.767.

Varying Parameters in the Sinkhorn Algorithm. We study how the performance of GEDIOT is impacted as the initial value of the regularization coefficient, denoted by ε_0 and the number of iterations varies in the learnable Sinkhorn layer. The results are presented in Section G.5 [1]. We find that both MAE and accuracy are stable with various ε_0, which shows the robustness of the learnable regularization method to ε_0. Moreover, we observe that the MAE decreases and the accuracy increases as the number of iterations increases, but after 15 (resp. 10) iterations on AIDS (resp. Linux), the MAE and accuracy become fairly stable as the Sinkhorn algorithm converges. Note that the computational time also increases when conducting more iterations. Considering the time-accuracy tradeoff, we set the iteration number to 5 by default.

Varying λ in the Loss Function. As presented in Section G.5 [1], we also discuss the effect of varying λ in Eq. (15) (from 0 to 1) that balances the two terms $\mathcal{L}_m$ and $\mathcal{L}_v$ of the loss function. The results show that the performance improves with the increase of λ in $[0, 1]$ and becomes stable when λ is around 0.8.

Varying the Size of Training Set. In this experiment, we evaluate the effect of varying the training set size. Concretely, we randomly sample 10%-100% of the original training set of AIDS and Linux to retrain GEDIOT. The results in Section G.5 [1] describe its influence on training time, MAE, and accuracy of GEDIOT. It can be observed that as the training set size increases, the MAE decreases and the accuracy increases, while the training time increases linearly. Furthermore, the observed trends of MAE and accuracy with increasing training set size appear to be flattening, which shows that training set size is sufficient.

k-Best Matching. We further verify the effect of k in k-best matching for GEP generation. As depicted in Section G.5 [1], the MAE constantly decreases and the accuracy increases as the parameter k increases. Nevertheless, computational time also increases with the increase of k since the search space becomes larger.

7 CONCLUSION

In this paper, we proposed novel optimal-transport-based methods for graph edit distance computation and graph edit path generation from both learning and optimization perspectives. We first proposed a neural network with inverse optimal transport called GEDIOT. By modeling the node edit operations and edge edit operations as optimization problems, we also proposed an unsupervised method GEDGW to approximate the GED value without the need of training. Additionally, we combine the two methods and propose an ensemble method GEDHOT which achieves a higher performance. Experiments demonstrate that our methods outperform the state-of-the-art methods for GED computation and GEP generation with remarkable result quality and generalizability.

REFERENCES

[1] 2024. Online Supplementary File. https://anonymous.4open.science/r/GED-via-Optimal-Transport-0F6F/Online_Supplementary_File.pdf.

[2] Jiyang Bai and Peixiang Zhao. 2021. TaGSim: Type-Aware Graph Similarity Learning and Computation. *PVLDB* 15, 2 (2021), 335–347.

[3] 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 *WSDM*. 384–392.

[4] David B Blumenthal and Johann Gamper. 2020. On The Exact Computation of The Graph Edit Distance. *Pattern Recognition Letters* 134 (2020), 46–57.

[5] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press.

[6] Gábor Braun, Alejandro Carderera, Cyrille W Combettes, Hamed Hassani, Amin Karbasi, Aryan Mokhtari, and Sebastian Pokutta. 2022. Conditional Gradient Methods. *arXiv preprint arXiv:2211.14103* (2022).

[7] Horst Bunke and Gudrun Allermann. 1983. Inexact Graph Matching for Structural Pattern Recognition. *Pattern Recognition Letters* 1, 4 (1983), 245–253.

[8] Lijun Chang, Xing Feng, Xuemin Lin, Lu Qin, Wenjie Zhang, and Dian Ouyang. 2020. Speeding up GED Verification for Graph Similarity Search. In *ICDE*. 793–804.

[9] Lijun Chang, Xing Feng, Kai Yao, Lu Qin, and Wenjie Zhang. 2022. Accelerating Graph Similarity Search via Efficient GED Computation. *IEEE Transactions on Knowledge and Data Engineering* 35, 5 (2022), 4485–4498.

[10] Laetitia Chapel, Mokhtar Z Alaya, and Gilles Gasso. 2020. Partial Optimal Transport with Applications on Positive-unlabeled Learning. *NeurIPS* 33 (2020), 2903–2913.

[11] Chandra R Chegireddy and Horst W Hamacher. 1987. Algorithms for Finding k-Best Perfect Matchings. *Discrete Applied Mathematics* 18, 2 (1987), 155–165.

[12] Wei-Ting Chiu, Pei Wang, and Patrick Shafto. 2022. Discrete Probabilistic Inverse Optimal Transport. In *ICML*. 3925–3946.

[13] Nicolas Courty, Rémi Flamary, Devis Tuia, and Alain Rakotomamonjy. 2016. Optimal Transport for Domain Adaptation. *TPAMI* 39, 9 (2016), 1853–1865.

[14] Marco Cuturi. 2013. Sinkhorn Distances: Lightspeed Computation of Optimal Transport. *NeurIPS* 26 (2013), 2292–2300.

[15] Yihe Dong and Will Sawin. 2020. COPT: Coordinated Optimal Transport on Graphs. *NeurIPS* 33 (2020), 19327–19338.

[16] Stefan Fankhauser, Kaspar Riesen, and Horst Bunke. 2011. Speeding up Graph Edit Distance Computation Through Fast Bipartite Matching. In *International Workshop on Graph-Based Representations in Pattern Recognition*. 102–111.

[17] Andreas Fischer, Ching Y. Suen, Volkmar Frinken, Kaspar Riesen, and Horst Bunke. 2013. A Fast Matching Algorithm for Graph-Based Handwriting Recognition. In *International Workshop on Graph-Based Representations in Pattern Recognition (Lecture Notes in Computer Science, Vol. 7877)*. 194–203.

[18] Karam Gouda and Mona Arafa. 2015. An Improved Global Lower Bound for Graph Edit Similarity Search. *Pattern Recognition Letters* 58 (2015), 8–14.

[19] Karam Gouda and Mosab Hassaan. 2016. CSI_GED: An Efficient Approach for Graph Edit Similarity Computation. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 265–276.

[20] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. 2019. Strategies for Pre-training Graph Neural Networks. *arXiv preprint arXiv:1905.12265* (2019).

[21] Derek Justice and Alfred Hero. 2006. A Binary Linear Programming Formulation of The Graph Edit Distance. *TPAMI* 28, 8 (2006), 1200–1214.

[22] Jongik Kim. 2021. Boosting Graph Similarity Search through Pre-computation. In *Proceedings of the 2021 International Conference on Management of Data*. 951–963.

[23] Jongik Kim, Dong-Hoon Choi, and Chen Li. 2019. Inves: Incremental Partitioning-Based Verification for Graph Similarity Search.. In *EDBT*. 229–240.

[24] Soheil Kolouri, Se Rim Park, Matthew Thorpe, Dejan Slepcev, and Gustavo K Rohde. 2017. Optimal Mass Transport: Signal Processing and Machine-Learning Applications. *IEEE Signal Processing Magazine* 34, 4 (2017), 43–59.

[25] Ling Li, Siqiang Luo, Yuhai Zhao, Caihua Shan, Zhengkui Wang, and Lu Qin. 2023. COCLEP: Contrastive Learning-based Semi-Supervised Community Search. In *ICDE*. 2483–2495.

[26] Ruilin Li, Xiaojing Ye, Haomin Zhou, and Hongyuan Zha. 2019. Learning to Match via Inverse Optimal Transport. *Journal of Machine Learning Research* 20, 80 (2019), 1–37.

[27] Yongjiang Liang and Peixiang Zhao. 2017. Similarity Search in Graph Databases: A Multi-Layered Indexing Approach. In *ICDE*. 783–794.

[28] Yongjiang Liang and Peixiang Zhao. 2017. Similarity Search in Graph Databases: A Multi-layered Indexing Approach. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 783–794.

[29] Junfeng Liu, Min Zhou, Shuai Ma, and Lujia Pan. 2023. MATA*: Combining Learnable Node Matching with A* Algorithm for Approximate Graph Edit Distance Computation. In *CIKM*. 1503–1512.

[30] Facundo Mémoli. 2011. Gromov-Wasserstein Distances and The Metric Approach to Object Matching. *Foundations of Computational Mathematics* 11, 4 (2011), 417–487.

[31] James Munkres. 1957. Algorithms for the Assignment and Transportation Problems. *Journal of the society for industrial and applied mathematics* 5, 1 (1957), 32–38.

[32] Michel Neuhaus, Kaspar Riesen, and Horst Bunke. 2006. Fast Suboptimal Algorithms for The Computation of Graph Edit Distance. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. 163–172.

[33] Hermina Petric Maretic, Mireille El Gheche, Giovanni Chierchia, and Pascal Frossard. 2019. GOT: An Optimal Transport Framework for Graph Comparison. *NeurIPS* 32 (2019), 13899–13910.

[34] Gabriel Peyré, Marco Cuturi, et al. 2019. Computational Optimal Transport: With Applications to Data Science. *Foundations and Trends® in Machine Learning* 11, 5-6 (2019), 355–607.

[35] Gabriel Peyré, Marco Cuturi, and Justin Solomon. 2016. Gromov-Wasserstein Averaging of Kernel and Distance Matrices. In *ICML*. 2664–2672.

[36] Chengzhi Piao, Tingyang Xu, Xiangguo Sun, Yu Rong, Kangfei Zhao, and Hong Cheng. 2023. Computing Graph Edit Distance via Neural Graph Matching. *PVLDB* 16, 8 (2023), 1817–1829.

[37] Shaima Qureshi et al. 2023. Limits of Depth: Over-Smoothing and Over-Squashing in GNNs. *Big Data Mining and Analytics* 7, 1 (2023), 205–216.

[38] Rishabh Ranjan, Siddharth Grover, Sourav Medya, Venkatesan Chakaravarthy, Yogish Sabharwal, and Sayan Ranu. 2022. Greed: A Neural Framework for Learning Graph Distance Functions. In *NeurIPS*. 22518–22530.

[39] Kaspar Riesen and Horst Bunke. 2008. IAM Graph Database Repository for Graph Based Pattern Recognition and Machine Learning. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR) (Lecture Notes in Computer Science, Vol. 5342)*. 287–297.

[40] Kaspar Riesen and Horst Bunke. 2009. Approximate Graph Edit Distance Computation by Means of Bipartite Graph Matching. *Image and Vision Computing* 27, 7 (2009), 950–959.

[41] Kaspar Riesen, Sandro Emmenegger, and Horst Bunke. 2013. A Novel Software Toolkit for Graph Edit Distance Computation. In *International Workshop on GraphBased Representations in Pattern Recognition*. 142–151.

[42] T Konstantin Rusch, Michael M Bronstein, and Siddhartha Mishra. 2023. A Survey on Oversmoothing in Graph Neural Networks. *arXiv preprint arXiv:2303.10993* (2023).

[43] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. 2011. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research* 12, 9 (2011), 2539–2561.

[44] Liangliang Shi, Jack Fan, and Junchi Yan. 2024. OT-CLIP: Understanding and Generalizing CLIP via Optimal Transport. In *ICML*. 1–22.

[45] Liangliang Shi, Zhaoqi Shen, and Junchi Yan. 2024. Double-Bounded Optimal Transport for Advanced Clustering and Classification. In *AAAI*, Vol. 38. 14982–14990.

[46] Liangliang Shi, Gu Zhang, Haoyu Zhen, Jintao Fan, and Junchi Yan. 2023. Understanding and Generalizing Contrastive Learning from The Inverse Optimal Transport Perspective. In *ICML*. 31408–31421.

[47] Andrew M Stuart and Marie-Therese Wolfram. 2020. Inverse Optimal Transport. *SIAM J. Appl. Math.* 80, 1 (2020), 599–619.

[48] Vayer Titouan, Nicolas Courty, Romain Tavenard, and Rémi Flamary. 2019. Optimal Transport for Structured Data with Application on Graphs. In *ICML*. 6275–6284.

[49] Titouan Vayer, Laetitia Chapel, Rémi Flamary, Romain Tavenard, and Nicolas Courty. 2020. Fused Gromov-Wasserstein Distance for Structured Objects. *Algorithms* 13, 9 (2020), 212.

[50] Titouan Vayer, Nicolas Courty, Romain Tavenard, Laetitia Chapel, and Rémi Flamary. 2019. Optimal Transport for Structured Data with Application on Graphs. In *ICML*, Vol. 97. PMLR, 6275–6284.

[51] Cédric Villani et al. [n. d.]. *Optimal Transport: Old and New*. Vol. 338. Springer.

[52] Cédric Vincent-Cuaz, Rémi Flamary, Marco Corneli, Titouan Vayer, and Nicolas Courty. 2021. Semi-Relaxed Gromov-Wasserstein Divergence and Applications on Graphs. In *ICLR*. 1–14.

[53] Hanchen Wang, Rong Hu, Ying Zhang, Lu Qin, Wei Wang, and Wenjie Zhang. 2022. Neural Subgraph Counting with Wasserstein Estimator. In *SIGMOD*. 160–175.

[54] Jianwei Wang, Kai Wang, Xuemin Lin, Wenjie Zhang, and Ying Zhang. 2024. Neural Attributed Community Search at Billion Scale. *PACMMOD* 1, 4 (2024), 1–25.

[55] Runzhong Wang, Tianqi Zhang, Tianshu Yu, Junchi Yan, and Xiaokang Yang. 2021. Combinatorial Learning of Graph Edit Distance via Dynamic Embedding. In *CVPR*. 5241–5250.

[56] Xiaoli Wang, Xiaofeng Ding, Anthony K. H. Tung, Shanshan Ying, and Hai Jin. 2012. An Efficient Graph Indexing Method. In *ICDE*. 210–221.

[57] Alan Geoffrey Wilson. 1969. The Use of Entropy Maximising Models, in the Theory of Trip Distribution, Mode Split and Route Split. *Journal of Transport Economics and Policy* (1969), 108–126.

[58] Bing Xiao, Xinbo Gao, Dacheng Tao, and Xuelong Li. 2008. HMM-Based Graph Edit Distance for Image Indexing. *International Journal of Imaging Systems and Technology* 18, 2-3 (2008), 209–218.
[59] Shunxin Xiao, Shiping Wang, Yuanfei Dai, and Wenzhong Guo. 2022. Graph Neural Networks in Node Classification: Survey and Evaluation. *Machine Vision and Applications* 33, 1 (2022), 4–22.
[60] Hongteng Xu, Dixin Luo, and Lawrence Carin. 2019. Scalable Gromov-Wasserstein Learning for Graph Partitioning and Matching. In *NeurIPS*. 3046–3056.
[61] Jingjing Xu, Hao Zhou, Chun Gan, Zaixiang Zheng, and Lei Li. 2021. Vocabulary Learning via Optimal Transport for Neural Machine Translation. In *ACL*. 1–13.
[62] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How Powerful Are Graph Neural Networks? *arXiv preprint arXiv:1810.00826* (2018).
[63] Lei Yang and Lei Zou. 2021. Noah: Neural-Optimized A* Search Algorithm for Graph Edit Distance Computation. In *ICDE*. 576–587.
[64] Weijie Yu, Zhongxiang Sun, Jun Xu, Zhenhua Dong, Xu Chen, Hongteng Xu, and Ji-Rong Wen. 2022. Explainable Legal Case Matching via Inverse Optimal Transport-based Rationale Extraction. In *SIGIR*. 657–668.
[65] Zhiping Zeng, Anthony KH Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. 2009. Comparing Stars: On Approximating Graph Edit Distance. *PVLDB* 2, 1 (2009), 25–36.
[66] Muhan Zhang. 2022. Graph Neural Networks: Link Prediction. *Graph Neural Networks: Foundations, Frontiers, and Applications* (2022), 195–223.
[67] Muhan Zhang and Yixin Chen. 2018. Link Prediction Based on Graph Neural Networks. *NeurIPS* 31 (2018), 5171–5181.
[68] Wei Zhang, Zihao Wang, Jie Fan, Hao Wu, and Yong Zhang. 2024. Fast Gradient Computation for Gromov-Wasserstein Distance. *Journal of Machine Learning* 3, 3 (2024), 282–299.
[69] Kangfei Zhao, Jeffrey Xu Yu, Hao Zhang, Qiyan Li, and Yu Rong. 2021. A Learned Sketch for Subgraph Counting. In *SIGMOD*. 2142–2155.
[70] Xiang Zhao, Chuan Xiao, Xuemin Lin, Qing Liu, and Wenjie Zhang. 2013. A Partition-Based Approach to Structure Similarity Search. *PVLDB* 7, 3 (2013), 169–180.
[71] Xiang Zhao, Chuan Xiao, Xuemin Lin, and Wei Wang. 2012. Efficient Graph Similarity Joins with Edit Distance Constraints. In *ICDE*. IEEE, 834–845.
[72] Xiang Zhao, Chuan Xiao, Xuemin Lin, Wenjie Zhang, and Yang Wang. 2018. Efficient Structure Similarity Searches: A Partition-based Approach. *The VLDB Journal* 27, 1 (2018), 53–78.
[73] Weiguo Zheng, Lei Zou, Xiang Lian, Dong Wang, and Dongyan Zhao. 2013. Graph Similarity Search with Edit Distance Constraint in Large Graph Databases. In *CIKM*. 1595–1600.
[74] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2020. Graph Neural Networks: A Review of Methods and Applications. *AI Open* 1 (2020), 57–81.

A REVIEW OF APPROXIMATE GED COMPUTATION

A.1 Review of Heuristic Algorithms

There are plenty of heuristic algorithms including A*-Beam [32], Hungarian [40], and VJ [16], all of which provide an approximate GED in polynomial time. A*-Beam [32] bounds the search space in the exact algorithm A* with a user-defined beam size for efficiency. Hungarian [40] constructs a cost matrix to estimate the number of edit operations induced by matching two nodes across two graphs and model the computation of GED as a linear sum assignment problem. It produces a matching matrix and approximate GED by solving it with the Hungarian algorithm [31]. In [40], the solution to a linear sum assignment problem concerning node matching is regarded as the approximation of the GED value. VJ [16] improves upon the primal-dual method used in the Hungarian algorithm and incorporates more effective search strategies to reduce the computational overhead.

A.2 Review of GNN-based Methods

Recently, graph neural networks (GNN) have become popular since the extracted node and graph embeddings can greatly help the performance in node classification [59, 74], link prediction [66, 67], and other classical graph problems [25, 54, 69], etc. Consequently, a number of GNN-based methods, such as SimGNN [3], TaGSim [2], Noah [63], MATA* [29] and GEDGNN [36], have also been proposed to generate embedding for GED computation with adequate training data, which achieve best performance in approximate GED computation. SimGNN [3] proposes to simply aggregate node embeddings into a graph embedding with attention mechanism for each graph, and then generate features for a given graph pair (G^1, G^2) with their embeddings using a neural tensor network. The features are then used to predict the GED for regression. TaGSim [2] categorizes the graph edit operations into different types and predicts the number of operations in each type more precisely. Noah [63] applies the heuristic A*-beam algorithm [32] guided by a GNN model called graph path network (GPN) to find small feasible edit paths. MATA* [29] employs a structure-enhanced GNN to learn the differentiable top-k candidate matching vertices which prunes the unpromising search directions of A*LSa [8] for approximate GED computation. Finally, GEDGNN [36] utilizes two separate cross-matrix modules to generate a cost matrix $\mathbf{A}_{\text{cost}}$ and a vertex-matching matrix $\mathbf{A}_{\text{match}}$, respectively, from GNN-extracted vertex features, where $\mathbf{A}_{\text{match}}$ is used for edit path generation, and both matrices are used to regress the GED. However, the correlation between $\mathbf{A}_{\text{cost}}$ and $\mathbf{A}_{\text{match}}$ is not captured, and $\mathbf{A}_{\text{match}}$ is directly used to fit the ground-truth vertex coupling relationship. In contrast, our GEDIOT model explicitly captures their correlation as $\mathbf{A}_{\text{match}} = \text{OT}(\mathbf{A}_{\text{cost}})$, where OT is our learnable Sinkhorn layer to be introduced in Section 4.2 that ensures the matching constraints to be established in Eq. (1).

B THEORETICAL ANALYSIS

B.1 Sinkhorn Algorithm for OT

In this section, we derive the Sinkhorn algorithm of optimal transport (OT) with Lagrange duality theory.

Recall that Sinkhorn is to solve the entropy relaxation of OT as specified in Eq. (3). The Lagrangian of Eq. (3) can be written as

$$\begin{aligned} L(\boldsymbol{\pi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &= \sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\left(\mathrm{C}_{i,j}\boldsymbol{\pi}_{i,j} + \boldsymbol{\pi}_{i,j}(\log \boldsymbol{\pi}_{i,j} - 1)\right) \\ &\quad + \sum_{i=1}^{n_1}\boldsymbol{\alpha}_i\left(\sum_{j=1}^{n_2}\boldsymbol{\pi}_{i,j} - \boldsymbol{\mu}_i\right) + \sum_{j=1}^{n_2}\boldsymbol{\beta}_j\left(\sum_{i=1}^{n_1}\boldsymbol{\pi}_{i,j} - \boldsymbol{\nu}_j\right) \\ &= \langle \mathbf{C}, \boldsymbol{\pi}\rangle + \varepsilon H(\boldsymbol{\pi}) + \left\langle \boldsymbol{\alpha}, \boldsymbol{\pi}\mathbf{1}_{n_2} - \boldsymbol{\mu}\right\rangle + \left\langle \boldsymbol{\beta}, \boldsymbol{\pi}^\top \mathbf{1}_{n_1} - \boldsymbol{\nu}\right\rangle \end{aligned}$$

Taking the derivative of the above Lagrangian with respect to $\boldsymbol{\pi}_{i,j}$ and setting it to zero, we get

$$\boldsymbol{\pi}_{i,j} = \boldsymbol{\varphi}_i \, \mathbf{K}_{i,j} \, \boldsymbol{\psi}_j,$$

where $\mathbf{K} \in \mathbb{R}^{n_1 \times n_2}$ is the kernel matrix with $\mathbf{K}_{i,j} = \exp\left(-\mathrm{C}_{i,j}/\varepsilon\right)$; $\boldsymbol{\varphi} \in \mathbb{R}^{n_1}$ and $\boldsymbol{\psi} \in \mathbb{R}^{n_2}$ are the dual variables with $\boldsymbol{\varphi}_i = \exp\left(-\boldsymbol{\alpha}_i/\varepsilon\right)$ and $\boldsymbol{\psi}_j = \exp\left(-\boldsymbol{\beta}_j/\varepsilon\right)$. Taking the derivatives of the Lagrangian with respect to $\boldsymbol{\alpha}_i$ and $\boldsymbol{\beta}_j$, and setting them to zero, we obtain

$$\boldsymbol{\mu}_i = \boldsymbol{\varphi}_i \sum_j^{n_2} \mathbf{K}_{i,j}\boldsymbol{\psi}_j, \quad \boldsymbol{\nu}_j = \boldsymbol{\psi}_j \sum_i^{n_1} \mathbf{K}_{i,j}\boldsymbol{\varphi}_i.$$

The Sinkhorn algorithm is to update the dual variables $\boldsymbol{\varphi}$ and $\boldsymbol{\psi}$ via the element-wise computation:

$$\begin{aligned} \boldsymbol{\psi} &= \boldsymbol{\nu} \oslash (\mathbf{K}^\top \boldsymbol{\varphi}), \\ \boldsymbol{\varphi} &= \boldsymbol{\mu} \oslash (\mathbf{K}\,\boldsymbol{\psi}), \end{aligned}$$

where the notation $\oslash$ is element-wise division. Note that the element $\mathbf{K}_{i,j}$ in $\mathbf{K}$ is strictly positive, and thus the denominators $\mathbf{K}^\top\boldsymbol{\varphi}$ and $\mathbf{K}\,\boldsymbol{\psi}$ are always non-zero.

B.2 Error Analysis of GEDIOT

In this section, we analyze the solution of our proposed GEDIOT during training.

The following theorem shows that in an ideal situation, the well-trained GEDIOT can output a coupling matrix that is the same as the ground truth node matching.

Theorem B.1. *There exists a cost matrix $\widehat{\mathbf{C}}^*$, such that the optimal coupling matrix $\widehat{\boldsymbol{\pi}}^*$ of the optimization problem*

$$\min_{\boldsymbol{\pi} \in U(\mathbf{1}_{n_1}, \mathbf{1}_{n_2})} \left\langle \widehat{\mathbf{C}}^*, \boldsymbol{\pi}\right\rangle + \varepsilon \left\langle \boldsymbol{\pi}, \log \boldsymbol{\pi} - 1\right\rangle$$

is exactly the ground truth node matching $\boldsymbol{\pi}^*$.

Proof. First, following Section 4.2, we add a dummy row and consider the optimization problem

$$\min_{\boldsymbol{\pi} \in \Pi(\widetilde{\boldsymbol{\mu}}, \widetilde{\boldsymbol{\nu}})} \left\langle \widetilde{\mathbf{C}}, \boldsymbol{\pi}\right\rangle + \varepsilon \left\langle \boldsymbol{\pi}, \log \boldsymbol{\pi} - 1\right\rangle, \tag{18}$$

$$\Pi(\widetilde{\boldsymbol{\mu}}, \widetilde{\boldsymbol{\nu}}) = \left\{\boldsymbol{\pi} \in \mathbb{R}^{(n_1+1)\times n_2} \mid \boldsymbol{\pi}\mathbf{1}_{n_2} = \widetilde{\boldsymbol{\mu}}, \ \ \boldsymbol{\pi}^\top \mathbf{1}_{n_1+1} = \widetilde{\boldsymbol{\nu}}, \ \ \boldsymbol{\pi} \geq 0\right\}.$$

Given arbitrary $\mathbf{a} \in \mathbb{R}^{n_1+1}$ and $\mathbf{b} \in \mathbb{R}^{n_2}$, let

$$\widetilde{\mathbf{C}}_{i,j} = -\left(\mathbf{a}_i + \mathbf{b}_j + \varepsilon \log \boldsymbol{\pi}^*_{i,j}\right), \text{ for } i = 1, \cdots, n_1, j = 1, \cdots, n_2,$$

and $\widetilde{\mathbf{C}}_{n_1+1,j} = 0$, for $j = 1, \cdots, n_2$. The Lagrange duality of Eq. (18) is

$$L(\boldsymbol{\pi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \left\langle \widetilde{\mathbf{C}}, \boldsymbol{\pi} \right\rangle + \varepsilon \left\langle \boldsymbol{\pi}, \log \boldsymbol{\pi} - 1 \right\rangle + \left\langle \boldsymbol{\alpha}, \boldsymbol{\pi} \mathbf{1}_{n_2} - \widetilde{\boldsymbol{\mu}} \right\rangle + \left\langle \boldsymbol{\beta}, \boldsymbol{\pi}^\top \mathbf{1}_{n_1+1} - \widetilde{\boldsymbol{\nu}} \right\rangle$$

Verifying the KKT condition [5]

$$\frac{\partial L}{\partial \boldsymbol{\pi}_{i,j}} = \widehat{\mathbf{C}}_{i,j} + \varepsilon \log \boldsymbol{\pi}_{i,j} + \boldsymbol{\alpha}_i + \boldsymbol{\beta}_j = -\left(\mathbf{a}_i + \mathbf{b}_j + \varepsilon \log \boldsymbol{\pi}^*_{i,j}\right) + \varepsilon \log \boldsymbol{\pi}_{i,j} + \boldsymbol{\alpha}_i + \boldsymbol{\beta}_j = 0,$$

Thus $\widehat{\boldsymbol{\pi}}^* = \boldsymbol{\pi}^*$, $\widehat{\boldsymbol{\alpha}}^* = \mathbf{a}$, and $\widehat{\boldsymbol{\beta}}^* = \mathbf{b}$ is a group of optimal solutions, where $\widehat{\boldsymbol{\pi}}^*$ is the optimal coupling matrix of Eq. (18) without the last row, and $\widehat{\boldsymbol{\alpha}}^*$ and $\widehat{\boldsymbol{\beta}}^*$ are corresponding optimal dual variables. Then, the first term of the objective function in the outer minimization in Eq. (7) reaches 0.

Particularly, according to Eq. (16), the approximate GED value is exactly the ground truth GED value when

$$\widehat{\mathbf{C}}^* = \mathbf{M} + \frac{1}{2}\mathcal{L}(\mathbf{A}^1, \mathbf{A}^2) \otimes \boldsymbol{\pi}^*. \tag{19}$$

□

Then based on the analysis in [26], we show the relation between errors in the learned cost matrix and errors in the learned coupling matrix during training.

Theorem B.2. *We assume that the ground truth node-matching matrix is $\boldsymbol{\pi}^*$ and one of the corresponding cost matrices is $\mathbf{C}^*$ (defined in Eq. (19)). During training, the coupling matrix and cost matrix are denoted as $\widehat{\mathbf{C}}$ and $\widehat{\boldsymbol{\pi}}$ respectively. Let $\Delta\mathbf{C} = \mathbf{C}^* - \widehat{\mathbf{C}}$ and $\Delta \log \boldsymbol{\pi} = \log \boldsymbol{\pi}^* - \log \widehat{\boldsymbol{\pi}}$, then*

$$\|\Delta\mathbf{C}\|_F \geq \varepsilon^2 \left(\|\Delta \log \boldsymbol{\pi}\|_F\right) - \mathbf{f}^\top \mathbf{A}^\dagger \mathbf{f},$$
$$\|\Delta \log \boldsymbol{\pi}\|_F \geq \varepsilon^{-2} \left(\|\Delta\mathbf{C}\|_F - \mathbf{g}^\top \mathbf{A}^\dagger \mathbf{g}\right),$$

where $\mathbf{A} = \begin{bmatrix} n_2\mathbf{I}_{n_1\times n_1} & \mathbf{1}_{n_1}\mathbf{1}_{n_2}^\top \\ \mathbf{1}_{n_2}\mathbf{1}_{n_1}^\top & n_1\mathbf{I}_{n_2\times n_2} \end{bmatrix}$, *$\mathbf{A}^\dagger$ is the Moore-Penrose inverse of matrix $\mathbf{A}$, Frobenius norm* $\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\mathbf{A}_{ij}^2}$, $\mathbf{f} = \left[(\Delta \log \boldsymbol{\pi}\mathbf{1})^\top, \mathbf{1}^\top(\Delta \log \boldsymbol{\pi})\right]^\top$, $\mathbf{g} = \left[(\Delta\mathbf{C}\mathbf{1})^\top, \mathbf{1}^\top(\Delta\mathbf{C})\right]^\top$, *and ε is the regularization coefficient.*

Proof. For the sake of simplicity, we assume a dummy row has already been added to the cost matrix. According to the KKT condition, given the cost matrix $\mathbf{C}$ and the coupling matrix $\boldsymbol{\pi}$, there exist $\boldsymbol{\alpha}, \boldsymbol{\beta}$ such that

$$\boldsymbol{\pi}_{i,j} = \exp\left(-\left(\mathbf{C}_{i,j} + \boldsymbol{\alpha}_i + \boldsymbol{\beta}_j\right)/\varepsilon\right).$$

Thus, there exist $\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*$ and $\widehat{\boldsymbol{\alpha}}, \widehat{\boldsymbol{\beta}}$ such that

$$\mathbf{C}^*_{i,j} = -\varepsilon \log \boldsymbol{\pi}^*_{i,j} - \boldsymbol{\alpha}_i - \boldsymbol{\beta}_j,$$
$$\widehat{\mathbf{C}}_{i,j} = -\varepsilon \log \widehat{\boldsymbol{\pi}}_{i,j} - \widehat{\boldsymbol{\alpha}}_i - \widehat{\boldsymbol{\beta}}_j.$$

Let $\Delta\boldsymbol{\alpha} = \boldsymbol{\alpha}^* - \widehat{\boldsymbol{\alpha}}, \Delta\boldsymbol{\beta} = \boldsymbol{\beta}^* - \widehat{\boldsymbol{\beta}}$, and we have

$$\Delta\mathbf{C}_{i,j} = -\varepsilon\Delta\boldsymbol{\pi}_{i,j} - \Delta\boldsymbol{\alpha}_i - \Delta\boldsymbol{\beta}_j.$$

Viewing $\Delta\boldsymbol{\alpha}, \Delta\boldsymbol{\beta}$ as variables and taking the minimum value of the right-hand side according to Lemma 3 in [26], it follows

$$\|\Delta\mathbf{C}\|_F \geq \varepsilon^2 \left(\|\Delta \log \boldsymbol{\pi}\|_F\right) - \mathbf{f}^\top \mathbf{A}^\dagger \mathbf{f}.$$

Similarly, consider

$$\log \boldsymbol{\pi}^*_{i,j} = -\varepsilon^{-1}\left(\mathbf{C}^*_{i,j} + \boldsymbol{\alpha}_i + \boldsymbol{\beta}_j\right),$$
$$\log \widehat{\boldsymbol{\pi}}_{i,j} = -\varepsilon^{-1}\left(\widehat{\mathbf{C}}_{i,j} + \widehat{\boldsymbol{\alpha}}_i + \widehat{\boldsymbol{\beta}}_j\right),$$

and we have

$$\|\Delta \log \boldsymbol{\pi}\|_F \geq \varepsilon^{-2}\left(\|\Delta\mathbf{C}\|_F - \mathbf{g}^\top \mathbf{A}^\dagger \mathbf{g}\right).$$

□

Moreover, we derive a bound for the gap between the approximate GED and the exact GED.

Theorem B.3. *Given the ground-truth node-matching matrix $\boldsymbol{\pi}^*$, its corresponding cost matrix $\mathbf{C}^*$, and the learned coupling matrix and cost matrix $\widehat{\boldsymbol{\pi}}$ and $\widehat{\mathbf{C}}$, the gap between the approximate GED value $\widehat{GED}$ and the exact GED value GED^* is bounded by*

$$n\|\Delta\mathbf{C}\|_F + \|\mathbf{C}^*\|_F\|\Delta\boldsymbol{\pi}\|_F,$$

where $n = \max\{n_1, n_2\}$, $\Delta\mathbf{C} = \mathbf{C}^* - \widehat{\mathbf{C}}$, and $\Delta\boldsymbol{\pi} = \boldsymbol{\pi}^* - \widehat{\boldsymbol{\pi}}$.

Proof. Considering that $\mathbf{C}^* = \mathbf{M} + \frac{1}{2}\mathcal{L}(\mathbf{A}^1, \mathbf{A}^2) \otimes \boldsymbol{\pi}^*$ according to Eq. (16), we analyze

$$\begin{aligned} |\widehat{GED} - GED^*| &= \left|\left\langle \widehat{\mathbf{C}}, \widehat{\boldsymbol{\pi}} \right\rangle - \left\langle \mathbf{C}^*, \boldsymbol{\pi}^* \right\rangle\right| \\ &= \left|\left\langle \widehat{\mathbf{C}} - \mathbf{C}^*, \widehat{\boldsymbol{\pi}} \right\rangle + \left\langle \mathbf{C}^*, \widehat{\boldsymbol{\pi}} - \boldsymbol{\pi}^* \right\rangle\right| \\ &\leq \|\widehat{\mathbf{C}} - \mathbf{C}^*\|_F\|\widehat{\boldsymbol{\pi}}\|_F + \|\mathbf{C}^*\|_F\|\widehat{\boldsymbol{\pi}} - \boldsymbol{\pi}^*\|_F \\ &\leq n\|\Delta\mathbf{C}\|_F + \|\mathbf{C}^*\|_F\|\Delta\boldsymbol{\pi}\|_F. \end{aligned}$$

The first "≤" is derived from the Cauchy-Schwarz inequality and the second is based on the fact that $\boldsymbol{\pi}_{i,j} \leq 1$. □

B.3 Diagram of GEDGW

We present the diagram of our unsupervised method GEDGW in Figure 9 based on the graphs G^1 and G^2 in Figure 1. Note that G^1 has 3 nodes while G^2 has 4. We add a dummy node u_4 in G^1 so that the two graphs have the same number of nodes, and the elements in the matrices in Figure 9 corresponding to the dummy node are represented by dashed lines. GEDGW formulates the GED computation as an optimization problem in Eq. (17) related to node matching, since GED can be obtained according to node matching. It first divides editing operations into two categories: the edge edit operations and the node edit operations, and two terms $\frac{1}{2}\sum_{i,j,k,l}(\mathbf{A}^1_{i,j} - \mathbf{A}^2_{k,l})^2\boldsymbol{\pi}_{i,k}\boldsymbol{\pi}_{j,l}$ and $\sum_{i,k}\mathbf{M}_{i,k}\boldsymbol{\pi}_{i,k}$ in the objective function of the optimization problem measure the two types of edit operations, respectively. As shown in the left part of Figure 9, matrices $\mathbf{A}^1 \in \{0,1\}^{4\times4}$ and $\mathbf{A}^2 \in \{0,1\}^{4\times4}$ are the adjacency matrices of G^1 and G^2, respectively. As illustrated in the right part of Figure 9, $\mathbf{M} \in \{0,1\}^{4\times4}$ is the node label matching matrix between nodes of G^1 and G^2, where $\mathbf{M}_{i,k} = 1$ if nodes $u_i \in V^1$ and $v_k \in V^2$ have the same label; otherwise $\mathbf{M}_{i,k} = 0$.

More concretely, each element $\left((\mathbf{A}^1_{i,j} - \mathbf{A}^2_{k,l})^2\right)_{i,j,k,l}$ in the 4-th order tensor indicates the discrepancy between every two edges $(u_i, u_j) \in E^1$ and $(v_k, v_l) \in E^2$. Subsequently, $(\mathbf{A}^1_{i,j} - \mathbf{A}^2_{k,l})^2\boldsymbol{\pi}_{i,k}\boldsymbol{\pi}_{j,l}$ measures the cost of the edge edit operations including edge insertion/deletion, since it represents whether edge $(u_i, u_j) \in E^1$ and

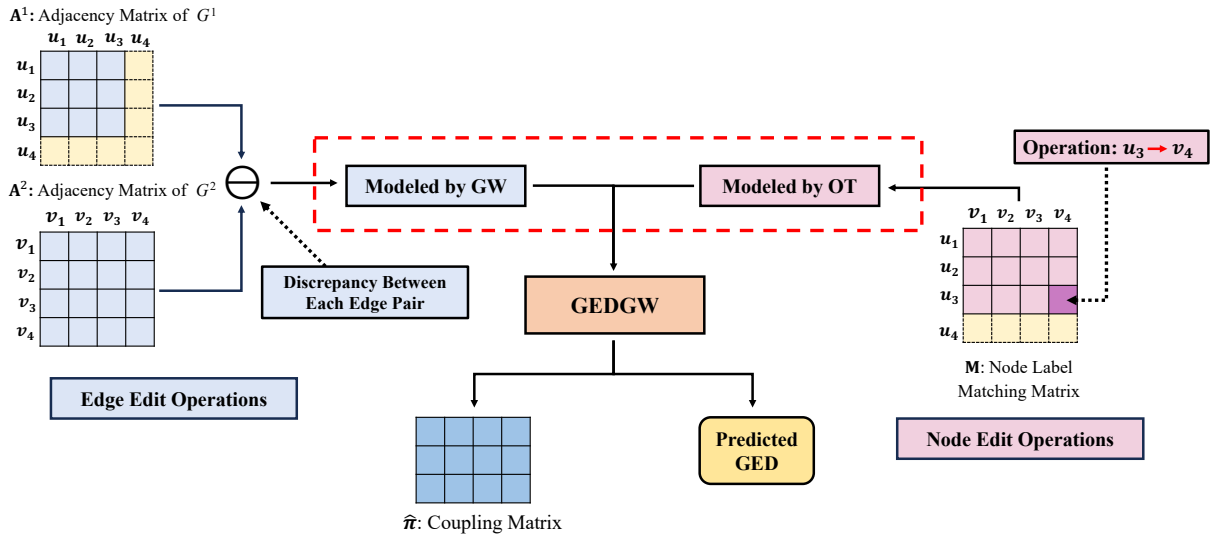


Figure 9: Diagram of the Proposed GEDGW

Algorithm 2: Conditional gradient algorithm for GEDGW

Input: graphs G^1, G^2

1 Compute $\mathbf{M}$ via the labels of nodes between G^1 and G^2
2 **for** $k = 1, 2, \ldots$ **do**
3 $\quad \mathbf{G}^{(k)} \leftarrow$ compute based on Eq. (20)
4 $\quad \tilde{\boldsymbol{\pi}}^{(k)} \leftarrow \underset{\boldsymbol{\pi} \in \Pi(\mathbf{1}_n, \mathbf{1}_n)}{\operatorname{argmin}} \langle \mathbf{G}^{(k)}, \boldsymbol{\pi} \rangle$
5 $\quad \gamma^{(k)} \leftarrow$ line search to find the optimal step size
6 $\quad \boldsymbol{\pi}^{(k)} \leftarrow (1 - \gamma^{(k)}) \cdot \boldsymbol{\pi}^{(k-1)} + \gamma^{(k)} \cdot \tilde{\boldsymbol{\pi}}^{(k)}$
7 $\widehat{\boldsymbol{\pi}} \leftarrow \boldsymbol{\pi}^{(k)}$
8 $\widehat{GED} \leftarrow \langle \widehat{\boldsymbol{\pi}}, \mathbf{M} \rangle + \frac{1}{2} \langle \widehat{\boldsymbol{\pi}}, \mathcal{L}(\mathbf{A}^1, \mathbf{A}^2) \otimes \widehat{\boldsymbol{\pi}} \rangle$
9 **return** $\widehat{GED}, \widehat{\boldsymbol{\pi}}$

edge $(v_k, v_l) \in E^2$ exist simultaneously when u_i matches v_k and u_j matches v_l. We model it as Gromov-Wasserstein Discrepancy (GW) (the left part of Figure 9).

Each element $\mathbf{M}_{i,k}\boldsymbol{\pi}_{i,k}$ measures the cost of the node edit operations including node relabeling and node insertion/deletion, since it represents matching a node in G^2 to a node in G^1 with a different label. We model it as Optimal Transport (OT) (the right part of Figure 9).

Then, we combine GW and OT to compute GED between G^1 and G^2 (marked with the red dashed frame) and output the approximate GED and the coupling matrix $\boldsymbol{\pi}$ for GEP generation as shown in the lower part of Figure 9.

B.4 Conditional Gradient Method

We solve Eq. (17) formulated in Section 5.1 to compute GED estimate $\widehat{GED}$ and the coupling matrix $\widehat{\pi}$ using the Conditional Gradient (CG) method. The main idea of CG method is to solve a linear approximate subproblem repeatedly and improve a solution within a feasible region. The key advantage is that it only requires solving a simpler linear subproblem at each iteration, which can be computationally efficient. The pseudo-code is presented in Algorithm 2.

At each iteration k, it first computes the gradient $\mathbf{G}^{(k)}$ with the current coupling matrix $\boldsymbol{\pi}^{(k-1)}$ (Line 3) by the following equation:

$$\mathbf{G}^{(k)} \leftarrow \mathbf{M} + \frac{1}{2}\mathcal{L}(\mathbf{A}^1, \mathbf{A}^2) \otimes \boldsymbol{\pi}^{(k-1)}. \tag{20}$$

The descent direction $\tilde{\boldsymbol{\pi}}^{(k)}$ is obtained by solving an OT problem with $\mathbf{G}^{(k)}$ as the cost matrix over the set $\Pi(\mathbf{1}_n, \mathbf{1}_n)$ (Line 4). Then the step size $\gamma^{(k)}$ in the line search is determined (Line 5) according to the constrained minimization of a second-order polynomial:

$$\underset{\gamma \in [0,1]}{\operatorname{argmin}} \left\langle \bar{\boldsymbol{\pi}}^{(k)}, \mathbf{M} \right\rangle + \frac{1}{2} \left\langle \mathcal{L}(\mathbf{A}^1, \mathbf{A}^2) \otimes \bar{\boldsymbol{\pi}}^{(k)}, \bar{\boldsymbol{\pi}}^{(k)} \right\rangle \tag{21}$$

$$\text{where } \bar{\boldsymbol{\pi}}^{(k)} = (1 - \gamma) \cdot \boldsymbol{\pi}^{(k-1)} + \gamma \cdot \tilde{\boldsymbol{\pi}}^{(k)}$$

More details of the line-search algorithm can be found in [10, 48]. The transport plan $\boldsymbol{\pi}^{(k)}$ is then updated for next iteration (Line 6).

Finally, it outputs GED estimate $\widehat{GED}$ and the coupling matrix $\widehat{\boldsymbol{\pi}}$, calculated in Lines 7-8. Moreover, $\widehat{\boldsymbol{\pi}}$ can be used for GED generation with the same k-best matching framework discussed in Section 4.5.

C k-BEST MATCHING

In this section, we provide the pseudocode of k-best matching framework that combines the label set based lower bound of GED and space splitting techniques. Algorithm 4 obtains the top-k best node matchings according to the length of their corresponding edit paths.

We begin by presenting a formal description of how to generate an edit path from a node matching between G^1 and G^2. The edit path generation procedure is shown in function EPGen(·) in Algorithm 3. With a given node matching $\mathbf{M}$ between G^1 and G^2, we first denote the node mapping as $f : V^1 \to V^2$ and the corresponding inverse mapping as $f^- : V^2 \to V^1$ (Line 2), where for $u \in V^1$ and $v \in V^2$, $f(u) = v$ and $f^-(v) = u$ if and only if

Algorithm 3: Edit Path Generation (EPGen)

Input: graphs $G^1 = (V^1, E^1, L^1)$, $G^2 = (V^2, E^2, L^2)$, node matching $\mathbf{M} \in \{0,1\}^{n_1 \times n_2}$

1 EPath = []
2 Generate the node mapping $f : V^1 \to V^2$ and inverse mapping $f^- : V^2 \to V^1$ from $\mathbf{M}$
// Node Relabeling
3 **foreach** node $u \in V^1$ **do**
4 **if** $L^1(u) \neq L^2(f(u))$ **then**
5 EPath.append(Relabel u with $L^2(f(u))$)
// Node Insertion
6 **foreach** node $v \in V^2 \backslash f(V^1)$ **do**
7 EPath.append(Insert a node with label $L^2(v)$ in G^1)
// Edge Deletion
8 **foreach** edge $(u, u') \in E^1$ **do**
9 **if** $(f(u), f(u')) \notin E^2$ **then**
10 EPath.append(Delete edge (u_1, u_2) from G^1)
// Edge Insertion
11 **foreach** edge $(v, v') \in E^2$ **do**
12 **if** $(f^-(v), f^-(v')) \notin E^1$ **then**
13 EPath.append(Insert edge $(f^-(v), f^-(v'))$ in G^1)
14 **return** EPath // edit path that transforms G^1 to G^2

$\mathbf{M}_{u,v} = 1$. The edit operations can be categorized into four types: node relabeling (Lines 3-5), node insertion (Lines 6-7), edge deletion (Lines 8-10), and edge insertion (Lines 11-13). For the two types of node edit operations, the algorithm checks whether node u in G^1 has a corresponding node $f(u)$ in G^2, and (if $f(u)$ exists) whether u and $f(u)$ have the same label. For each edge (u, u') in G^1, the algorithm checks whether the corresponding $(f(u), f(u'))$ in G^2 exist. If $(f(u), f(u'))$ does not exist, an edge deletion operation is needed. Similarly, for each edge (v, v') in G^2, it checks whether the corresponding $(f^-(v), f^-(v'))$ in G^1 exist. If $(f^-(v), f^-(v'))$ does not exist, an edge insertion operation is needed.

Then we introduce the label set based GED lower bound [8], which can be calculated in linear time and prune out unnecessary node matchings in k-best matching framework. It is formulated as:

$$\mathrm{GEDLB}\left(G^1, G^2\right) = \left|L\left(V^1\right) \oplus L\left(V^2\right)\right| + \left|\left|E^1\right| - \left|E^2\right|\right| \tag{22}$$

where $L\left(V^1\right)$ and $L\left(V^2\right)$ denote the multi-set of node labels of G^1 and G^2 respectively, and $\oplus$ denotes a multi-set function that $A \oplus B = A \cup B - A \cap B$.

Now, we explain the k-best matching framework in Algorithm 4. Line 1 construct a weighted complete bipartite graph between V^1 and V^2, where the weight of edge (u, v) ($u \in V^1$, $v \in V^2$) is $\boldsymbol{\pi}_{u,v}$. We also define the weight of a node matching M as the Frobenius product of $\boldsymbol{\pi}$ and M (i.e., $\langle \boldsymbol{\pi}, M \rangle$). Lines 2-7 initialize the first solution subspace S_1, where $M_1(S_1)$ and $M_2(S_1)$ denote the best and second-best node matchings in S_1 respectively, which can be found in $O\left(n^3\right)$ time by classical algorithms [11]. The function GEDLowerBound($\cdot$) in Lines 5 calculates the label-set-based GED lower bound via Eq. (22). In Lines 6-7, Update($\cdot$) means replacing

Algorithm 4: k-best Matching Framework

Input: graphs $G^1 = (V^1, E^1)$, $G^2 = (V^2, E^2)$, coupling matrix $\boldsymbol{\pi}$, k

1 Construct bipartite graph $G = (V^1, V^2, V^1 \times V^2, \boldsymbol{\pi})$
2 $BestPath \leftarrow None$; $S_1 \leftarrow \{M \mid M \text{ is a node matching}\}$
3 $M_1(S_1) \leftarrow$ BestMatch(S_1)
4 $M_2(S_1) \leftarrow$ SecondBestMatch(S_1)
5 $LB(S_1) \leftarrow$ GEDLowerBound(S_1)
6 Update($BestPath$, EPGen($M_1(S_1)$))
7 Update($BestPath$, EPGen($M_2(S_1)$))
8 **for** $t = 2$ *to* k **do**
9 $id \leftarrow None$, $max\_weight \leftarrow -\infty$
10 **for** $S_i \in \{S_1, \cdots, S_{t-1}\}$ **do**
11 **if** $LB(S_i) < len(BestPath)$ **then**
12 $weight \leftarrow \langle \boldsymbol{\pi}, M_2(S_i) \rangle$
13 **if** $weight > max\_weight$ **then**
14 $(id, max\_weight) \leftarrow (i, weight)$
15 $(S_{id}, S_t) \leftarrow$ SpaceSplit(G, S_{id})
16 $LB(S_{id}) \leftarrow$ GEDLowerBound(S_{id})
17 Update($BestPath$, EPGen($M_2(S_{id})$))
18 Update($BestPath$, EPGen($M_2(S_t)$))
19 **return** $BestPath$
20 **Function** SpaceSplit(G, S):
21 Choose an arbitrary edge $e \in M_1(S)$ but $e \notin M_2(S)$
22 $S' = \{M \in S \mid e \in M\}$, $S'' = \{M \in S \mid e \notin M\}$
23 $M_1(S') \leftarrow M_1(S)$, $M_2(S') \leftarrow$ SecondBestMatch(S')
24 $M_1(S'') \leftarrow M_2(S)$, $M_2(S'') \leftarrow$ SecondBestMatch(S'')
25 $LB(S'') \leftarrow LB(S)$
26 **return** S', S''

the current best solution BestPath by the edit path output from EPGen($\cdot$) if BestPath is None or that path is shorter.

Lines 8-26 show the iterative space-splitting method. Suppose that there are $(t-1)$ subspaces, and each subspace has its own best and second-best node matching $M_1(S_i)$ and $M_2(S_i)$, we choose the subspace where the second-best node matching has the maximum weight among all the subspaces for further splitting (Lines 9-14). If the GED lower bound of a subspace S is greater or equal to the length of the current best path, it is unpromising, so there is no need to further split S (Line 11). Then, we split the chosen subspace S_{id} and update the GED lower bound and best path of the new subspaces (Lines 15-18).

Lines 20-26 specify the SpaceSplit($\cdot$) function using Line 15, which splits S into two subspaces S' and S'', such that a node matching of S is in S' if it contains e, and otherwise it is in S'' (Lines 21-22). Note that $M_1(S)$ (resp. $M_2(S)$) becomes the best node matching in S' (resp. S'') after splitting (Lines 23-24). The entire node matching space is partitioned by repeatedly selecting a subspace to split in this manner. This process is repeated until k subspaces are reached. Finally, $2k$ node matchings (2 from each subspace) are collected as the candidate set to find the shortest edit path. More details can be found in Section 4 in [36].

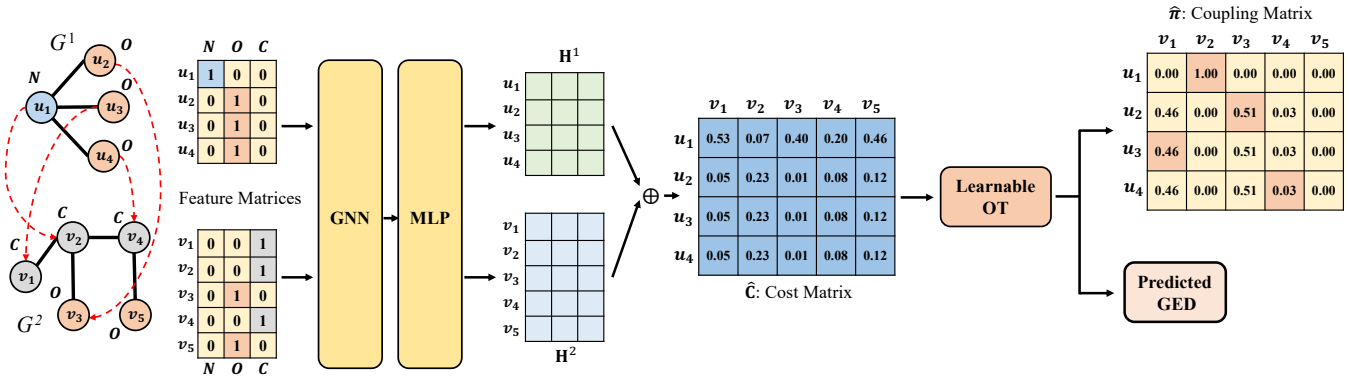


Figure 10: A Case Study for GEDIOT

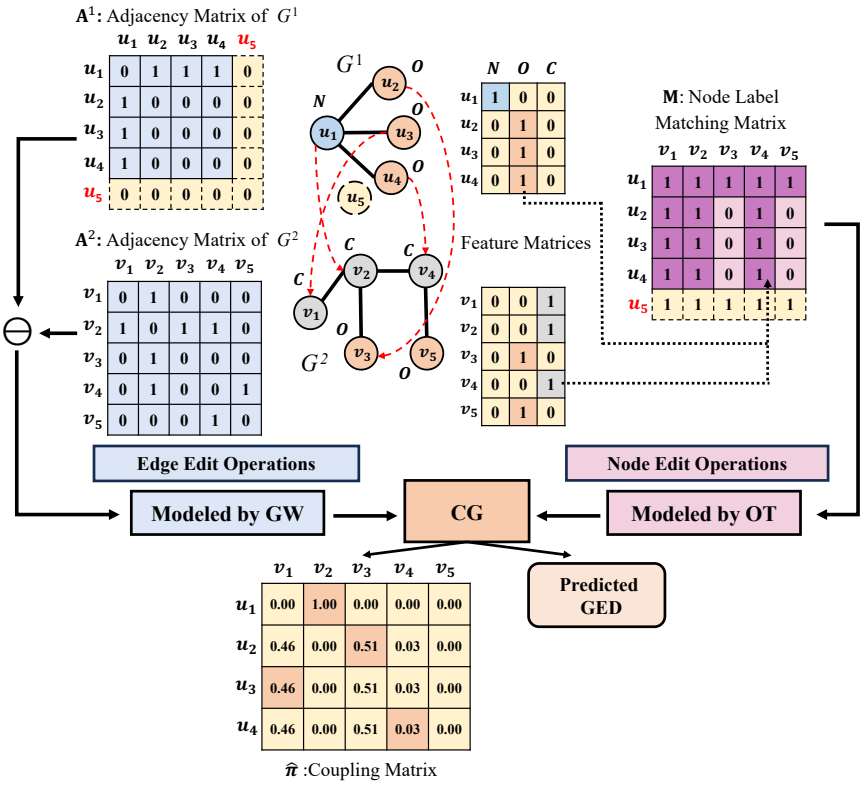


Figure 11: A Case Study for GEDGW

D CASE STUDY

We conduct a case study of GED computation between a 4-node G^1 and a 5-node G^2 from AIDS by our proposed GEDIOT in Figure 10. The graphs are converted from the chemical compounds where nodes and edges represent the atoms and covalent bonds, respectively. The color of a node indicates its label (i.e., the type of atoms). G^1 contains a Nitrogen (i.e., N) atom and three Oxygen (i.e., O) atoms, and G^2 contains three Carbon atoms (i.e., C) and two Oxygen atoms. The ground-truth node-matching is shown by the dashed red lines (e.g., u_1 in G^1 corresponds to v_2 in G^2). The i^{th} row of the feature matrix is the one-hot encoding of the label of node u_i (or v_i). We initialize the node embedding as the feature matrix and obtain the final embedding from GNN and MLP modules (i.e., node embedding component). Given two node embedding matrices obtained from G^1 and G^2, the pairwise scoring operation $\oplus$ returns a cost matrix (discrepancy matrix) $\mathbf{C}$ where element $\mathbf{C}_{i,j}$ is the pairwise score computed from the embeddings of node u_i in G^1 and node v_j in G^2, which is illustrated in Figure 2. We can see that in the cost matrix, the cost between node u_1 in G^1 and node v_2 in G^2 is much smaller than the cost between u_1 and other nodes in G^2,

which is consistent with the fact that u_1 and v_2 are similar (e.g., their degrees are both 3). Note that the numbers of nodes in the two graphs are different, and we extend the cost matrix with a dummy row filled with 0 and redefine mass distributions $\widetilde{\boldsymbol{\mu}}$ and $\widetilde{\boldsymbol{\nu}}$ according to Section 4.2. Then, the cost matrix is fed into the learnable OT component to seek a global decision that minimizes the total cost of transporting masses from nodes of G^1 to nodes of G^2. The OT component outputs a coupling matrix that fits the ground-truth node-matching matrix for GED computation and GEP generation. Each row in the coupling matrix is a probability vector for u_i, representing the probability of matching u_i to each v_j. Elements $\boldsymbol{\pi}_{ij}$ in the coupling matrix highlighted in the darker color in Figure 10 correspond to the non-zero elements $\boldsymbol{\pi}^*_{ij}$ (i.e., u_i in G^1 matches v_j in G^2) in the ground-truth node-matching.

In Figure 11, we also present a case study for GEDGW with the same graphs as Figure 10. We first construct a binary node label matching matrix $\mathbf{M}$, where each element $\mathbf{M}_{ij}$ is 0 if and only if u_i and v_j have the same label. Noticing that $|V^1| < |V^2|$, we add a dummy node u_5 in G^1 so that the two graphs have the same number of nodes. The node label matching matrix $\mathbf{M}$ and the two extended adjacency matrices $\mathbf{A}^1$ and $\mathbf{A}^2$ are used to model node and edge operations in the optimization problem of GEDGW, which is then solved with the Conditional Gradient (CG) method. Same as GEDIOT, GEDGW also outputs a predicted GED and a coupling matrix that fits the node-matching matrix.

E TIME COMPLEXITY ANALYSIS

In this section, we provide a comprehensive analysis of the time complexity of our proposed methods.

E.1 Time Complexity of GEDIOT

As the model training can be done offline, we consider the computation cost of the forward propagation for GEDIOT. For ease of description, we assume that the number of GNN layers is N, the dimension of hidden layers of GNN and MLP is d, and the output dimension of NTN is L. The dimension D of the input $\mathbf{h}$ of MLP is $(N+1)d$ since it is the concatenation of the output of each GNN layer and the initial node features. Let $n = n_2$, $m = \max(m_1, m_2)$ for the given graph pair (G^1, G^2) and M be the number of iterations of the Sinkhorn algorithm. Note that for two matrices $\mathbf{A} \in \mathbb{R}^{p\times q}$ and $\mathbf{B} \in \mathbb{R}^{q\times r}$, the time complexity of matrix multiplication $\mathbf{AB}$ is $O(pqr)$, which we will use without mentioning again in the following analysis. We introduce the computation cost of all modules and sum them up to get the total cost.

In the node embedding component, in each layer of GNN, the aggregation of node features from every neighbor of GNN takes $O(md)$ time, and the linear transformation of the features of each node consumes a total of $O(nd^2)$ time. Therefore, GNN takes $O(N(md+nd^2))$ time to generate the node embedding $\mathbf{h}$. To obtain the final node embedding $\mathbf{H}$, the three-layer MLP module requires a total of $O(n((N+1)d)^2)$ time for the transformation. The computation cost of the node embedding component is therefore bounded by $O(N(md+nd^2)+n((N+1)d)^2)$.

In the graph discrepancy component, the node attentive mechanism costs $O(nd+d^2)$ time to generate the graph-level embedding $\mathbf{H}_G$. Then, it takes $O(Ld^2)$ time to compute the interaction vector $\mathbf{s}(G^1, G^2)$. The fully connected neural networks consume $O(Ld^2)$ to obtain the predicted score. The computation cost of the neural tensor network is bounded by $O(Ld+d^2)$. The computation cost of the graph discrepancy component is therefore bounded by $O(nd+Ld^2)$.

As for the learnable OT component, it first computes the cost matrix with the two final node embeddings $\mathbf{H}^1$ and $\mathbf{H}^2$, which takes $O(nd^2+n^2d)$ time. Subsequently, the Sinkhorn layer runs Algorithm 1 for M iterations, with each iteration requiring $O(n^2)$ time. This results in $O(Mn^2)$ cost in total for the Sinkhorn layer. The computation cost of the learnable OT component is therefore bounded by $O(nd^2+n^2d+Mn^2)$.

Combining all the costs, the time complexity for the forward propagation of GEDIOT is

$$O\left(N(md+nd^2+nN^2d^2)+Ld^2+nd^2+n^2d+Mn^2\right).$$

Since $m = O(n^2)$ and N, L, d and M are fixed in GEDIOT, it can be simplified to $O(n^2)$, which is related to the size of the input graph.

For GEP generation, the two main steps of the k-best matching framework are finding the best node matching and edit path generation via this node matching, which are repeated k times to find the best edit path. The first task takes $O(n^3)$ time to find the maximum node matching [36]. Recall that we can generate an edit path by traversing all vertices and edges with a given node matching, which takes $O(m+n)$ time. In total, the time complexity of GEP generation is therefore $O(k(m+n+n^3)) = O(kn^3)$.

E.2 Time Complexity of GEDGW and GEDHOT

For GEDGW, we use the CG method [6, 52] (see Algortihm 2 in Section B.4 for details) to solve Eq. (17). The main time cost in each iteration of Algorithm 2 lies in the tensor product $\mathcal{L}(\mathbf{A}^1, \mathbf{A}^2) \otimes \boldsymbol{\pi}$ as shown in Eq. (20). Directly computing it takes $O(n^4)$ time, but according to Proposition 1 in [35], its computation can be accelerated to $O(n^3)$ time by decomposing $\mathcal{L}(\mathbf{A}^1, \mathbf{A}^2) \otimes \boldsymbol{\pi}$ into multiple matrix multiplications. Therefore, the total time complexity of Algorithm 2 is bounded by $O(Kn^3)$, where K is the number of iterations.

For the process of GEDHOT, the two methods GEDIOT and GEDGW are called separately. Recall that the time complexity of the forward propagation of GEDIOT and GEDGW in Algorithm 2 are $O(n^2)$ and $O(Kn^3)$, respectively, where K is the number of iterations of GW computation. Therefore, the time complexity of GEDHOT to approximate GED is bounded by $O(n^2+Kn^3) \approx O(Kn^3)$. Since the time complexity to generate GEP using the k-best matching framework is $O(kn^3)$, the total time of predicting both GED and GEP is bounded by $O((K+k)n^3)$.

F EXPERIMENTAL SETTING

F.1 Datasets

We use three real-world graph datasets: AIDS, Linux, and IMDB.

AIDS. The AIDS dataset consists of chemical compounds from the Developmental Therapeutics Program at NCI/NIH. The chemical compounds are converted into graphs where nodes and edges represent the atoms and covalent bonds, respectively. Each node is labeled with one chemical symbol, e.g. C, N, O, etc., while the edges are unlabeled.

Linux. The Linux dataset consists of program dependence graphs generated from the Linux kernel, where each graph represents a function. The nodes and edges represent the statements and the dependency between the two statements, which are both unlabeled.

IMDB. The IMDB dataset consists of ego-networks of movie actors and actresses. Each node denotes a movie actor or actress, and each edge between two nodes denotes the two people acting in the same movie. The nodes and edges are unlabeled.

Data Preprocessing. We use the A* algorithm [41] to generate the exact ground truth for the graph pairs from AIDS, Linux, and a part of IMDB where each graph has no more than 10 nodes. Since the GEP to transform G^1 to G^2 may not be unique, for training, we produce up to 10 ground-truth paths for each graph pair if they exist. Note that each ground-truth path GEP_i^* corresponds to a binary node-matching matrix $\boldsymbol{\pi}_i^*$. We use all these $\boldsymbol{\pi}_i^*$ as the ground-truth node matching (i.e., $\boldsymbol{\pi}^*$ in the matching loss $\mathcal{L}_m$ in Eq.(7)) during training to enrich the datasets and improve the model performance.

For the rest of IMDB where the number of nodes is larger than 10, we generate 100 synthetic graphs for each graph G with the ground-truth generation technique in [2, 36]. Concretely, each synthetic graph G' is randomly generated with Δ edit operations on nodes/edges, where Δ is a random number within $(0, 10]$ if the nodes of the original graph are larger than 20; otherwise it is within $(0, 5]$. Here, Δ is regarded as an approximation of the ground truth $GED^*(G, G')$, and the Δ edit operations are regarded as the GEP.

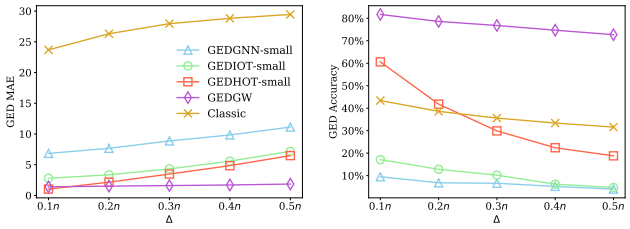
Training Set. Following the experimental settings of [36], we sample 60% graphs in each dataset to form the training set. Each sample in the training set is a graph pair. For AIDS and Linux, the two graphs in each graph pair in the training set are directly sampled from the graph dataset, since the GED exact ground truth of the graph pairs can be obtained with the A* algorithm. For IMDB, we denote those sampled training graphs with at most (resp. larger than) 10 nodes as small (resp. large) graphs. The training set contains two parts: 1) graph pairs formed by two small graphs; 2) graph pairs formed by a large graph and its corresponding synthetic graph.

Validation and Test Sets. We select 20% graphs in each dataset to form the test set. We evaluate our methods on the scenarios following the setting of [3, 36], which is to model the graph similarity search. The training set is regarded as the graph database and the graphs in the test set can be regarded as the queries. We sample 100 training graphs for each test graph to form the test set. The remaining 20% graphs of each dataset are sampled to form the validation set in the same way as the test set.

F.2 Detailed Setup of Our Methods

Our code is written in Python and all the models are implemented by PyTorch. We use PyTorch Geometric for GNN implementation.

Parameter Settings. For the node embedding component, the number of GIN layers is set to 3. The output dimension for each GIN layer is 128, 64, 32, respectively. The dimension of the final node embedding outputted by the MLP is set to $d = 32$. For the learnable OT component, we use a 32×32 learnable interaction matrix $\mathbf{W}$ in the cost matrix layer. Then, we perform the Sinkhorn algorithm with an initial $\varepsilon_0 = 0.05$ for 5 iterations in the learnable Sinkhorn layer. For the graph discrepancy component, the output dimension of NTN is set to $L = 16$. The output dimensions of the four succeeding dense layers are set to 16, 8, 4, 1. The hyper-parameter λ in the loss function is set to 0.8. During training, we set the batch size to 128 and use the Adam optimizer with initial learning rate and weight decay set to 0.001 and 5×10^{-4}, respectively. For GEP generation, we set k to 100 in the k-best matching framework.



(a) IMDB - MAE (b) IMDB - Accuracy

Figure 12: Further evaluation of generalizability for large unseen graphs on IMDB with Increasing GED

G ADDITIONAL EXPERIMENTS

G.1 Generalizability

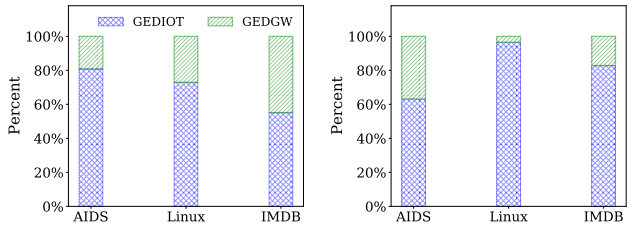
We further discuss how the generalizability of GEDGNN-small, GEDIOT-small, and GEDHOT-small is impacted when synthesizing large test graph pairs (more than 10 nodes) with larger GEDs (i.e., the discrepancy between two graphs becomes more pronounced). Concretely, for each original large graph with n nodes ($n > 10$) in the test set of IMDB, we regenerate 100 synthetic graphs with edit operations $\Delta = \lceil r \cdot n \rceil$, where r is in the range of $(0, 1)$ and Δ can be viewed as an approximation of the ground-truth GED as described in Section 6.1. We vary r from 10% to 50% and Figure 12 depicts the influence on MAE and accuracy.

We can see that both non-learning methods Classic and GEDGW are quite stable as Δ varies since they do not need ground-truths. The MAE of Classic is several times worse than that of the other four methods, but as Δ increases, Classic achieves a better accuracy than the learning-based methods. This implies that Classic can recover the exact GED in several instances but struggles in others. Our proposed GEDGW significantly outperforms all the others including the learning-based methods in terms of MAE and accuracy, showing the great robustness of GEDGW compared with other methods.

Among the three learning-based methods trained on the small training set (graphs with nodes no more than 10), GEDHOT-small achieves the best performance with the help of GEDGW, and GEDIOT-small is consistently better than GEDGNN-small. This indicates that our proposed neural network model exhibits superior generalizability compared to the existing learning-based methods.

G.2 More Evaluation on Proposed Methods

Adoption Ratio of GEDIOT and GEDHOT. The ensemble method GEDHOT adopts the smaller GED of GEDIOT and GEDGW, and the shorter GED path of GEDIOT and GEDGW. GEDHOT uses the values and paths from GEDIOT by default unless GEDGW outputs better results. We evaluate the ratio of the cases in which GEDGW



(a) Adoption Rate in GED Computation (b) Adoption Rate in GEP Generation

Figure 13: Adoption Rate of GEDIOT/GEDGW for GEDHOT

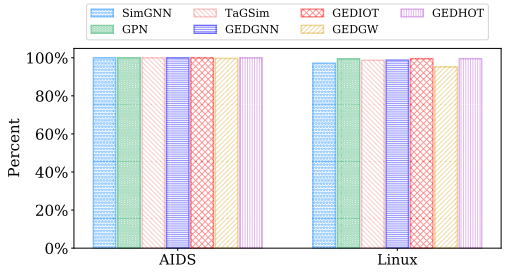


Figure 14: Triangle Property Preservation of the Predicted GEDs

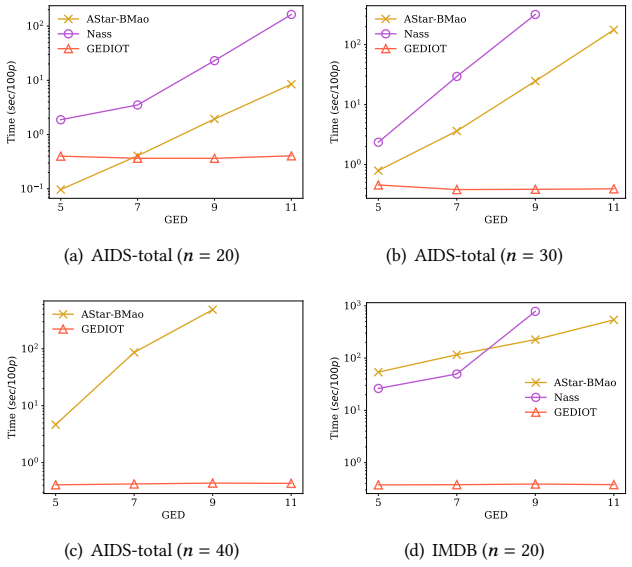
outperforms GEDIOT and vice versa. As shown in Figure 13, on AIDS, for GED computation, most graph pairs (80.8%) use the results from GEDIOT instead of GEDGW. For GEP generation, 63.1% of the graph pairs use the results from GEDIOT, and 36.9% of the graph pairs use the results from GEDGW. The results show the need to apply GEDGW (as a non-learning method) to offset the potential weakness of GEDIOT (and learning-based methods in general) for GED computation and GEP generation, particularly on pairs of larger graphs in IMDB that are difficult to train well.

Triangle Property Preservation of the Predicted GEDs. To evaluate whether learning-based methods preserve the triangle inequality in the GED predictions, We randomly sample triples of graphs of the form (G^1, G^2, G^3) and report the fraction of violations for various learning-based methods (including ours). Figure 14 shows that on AIDS and Linux, our methods preserve the GED triangle inequality for more than 95% cases. Particularly, on AIDS, GEDIOT and GEDHOT preserve the property for 99.9% cases.

G.3 Comparison with Exact Methods

We notice that the state-of-the-art methods Nass [22] and AStar-BMao [9] for graph similarity search (introduced in Section 2) can be applied for exact GED computation by setting the threshold in search task to infinity. As indicated in [36], exact methods suffer from huge computation costs when the graph size increases. We first compare our method GEDIOT with the exact ones on two large real-world datasets: AIDS-total[1] and IMDB. Differing from AIDS introduced in Table 2 and Section 6.1, here we use the large dataset AIDS-total that contains 42,689 graphs, with 25.60 nodes per graph

[1]https://cactus.nci.nih.gov/download/nci/AID2DA99.sdz



(a) AIDS-total ($n = 20$) (b) AIDS-total ($n = 30$)

(c) AIDS-total ($n = 40$) (d) IMDB ($n = 20$)

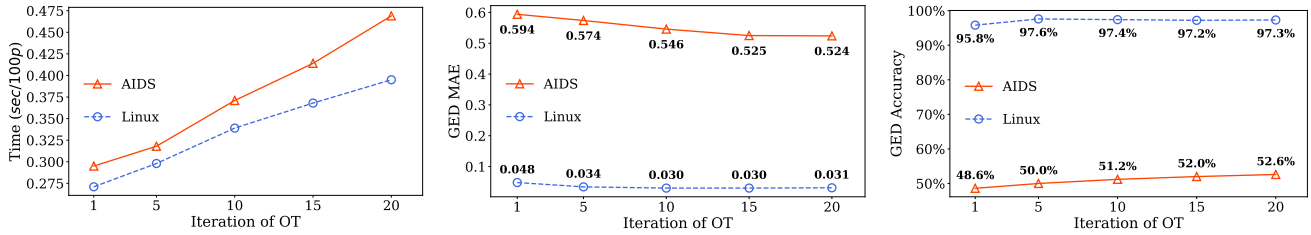
Figure 15: Efficiency Comparison with Exact Algorithms

on average. IMDB is the same as the dataset in Table 2, consisting of 1500 unlabeled graphs, with 13 nodes per graph on average.

We remove the edge labels in the large AIDS-total dataset following the convention of learning-based methods [2, 36], and compare Nass and AStar-BMao with our learning method GEDIOT on AIDS-total and IMDB. For each graph dataset, we select subsets of graphs from the dataset with n nodes, where all the graphs in each subset have n nodes. We use four groups of graphs with $n = 20, 30, 40$ from AIDS-total and only one group of graphs with $n = 20$ from IMDB since on larger graphs of IMDB, AStar-BMao cannot output the results within 24 hours and Nass returns a bus error, likely caused by too deep recursion. We sample 60% graphs to train GEDIOT and 40% for efficiency evaluation and use the ground-truth generation technique as described in Data Preprocessing in Section F.1 to generate graph pairs. Concretely, we fix $\Delta = 5, 7, 9, 11$ to generate four groups of graph pairs for each subset, where each group has 100 graph pairs.

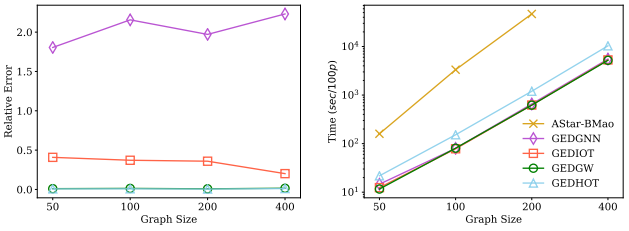
In Figure 15, we report the average running time of every 100 pairs for each group using the three methods. The computational time of the two exact methods Nass and AStar-BMao is quite sensitive w.r.t. the graph size and the GED value. We do not report the results of some groups of AStar-BMao and Nass since they fail to return the GED value due to bus error. Our method GEDIOT shows a consistent advantage compared to the two exact algorithms, particularly for larger graphs and GEDs. In particular, on AIDS-total ($n = 40$) and IMDB ($n = 20$), GEDIOT outperforms the state-of-the-art exact algorithm in time efficiency by orders of magnitude, as the time complexity of GEDIOT is only $O(n^2)$, whereas AStar-BMao and Nass are still exponential-time algorithms.

Moreover, notice that the scalability of the exact methods on IMDB is worse than that on AIDS. The reason could be that the graphs in IMDB are denser and have no labels leading to a huge search space when using exact algorithms.



(a) Number of Iterations - Time (b) Number of Iterations - MAE (c) Number of Iterations - Accuracy

Figure 18: Effect of Various Numbers of Iterations for the Sinkhorn Algorithm on GEDIOT



(a) Graph Size - Relative Error (b) Graph Size - Time (sec/100pair)

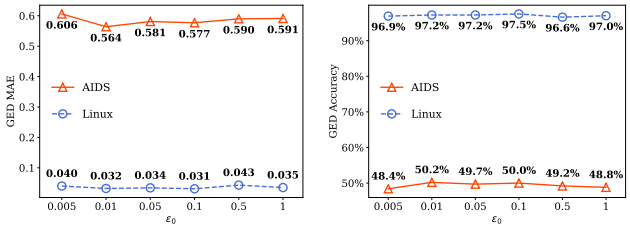
Figure 16: Accuracy and Efficiency on Power-law Graphs

G.4 Performance Evaluation on Large Power-Law Graphs

Following the experiments on large power-law graphs in Section 5.4 of GEDGNN [36], we also generate four groups of large synthetic power-law graphs with various graph sizes n. The graph sizes n of the four groups are set as 50, 100, 200, and 400, respectively. For each n, we generate 500 pairs for training and testing, respectively. The results are shown in Figure 16.

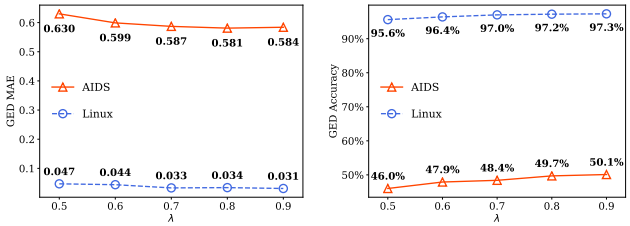
In Figure 16(a), we report the GED relative errors (i.e., $(\widehat{GED} - GED^*)/GED^*$) of the approximate methods with k-best matching framework: GEDGNN, GEDIOT, GEDGW, and GEDHOT. Note that the relative errors of all the methods are quite stable as the graph size n varies. Specifically, the relative error of our GEDGW and GEDHOT is nearly 0. In stark contrast, that of GEDGNN hovers around a relatively high value of almost 2. This pronounced discrepancy showcases the superiority of our proposed methods in larger power-law graphs.

Figure 16(b) depicts the average running time of 100 graph pairs for the exact algorithm AStar-BMao and the above approximate methods. We do not report the result of Nass as it cannot output the results on the four groups due to bus error. It shows that the average running time of approximate methods is consistently orders of magnitude faster than AStar-BMao. The result of AStar-BMao on 400-node graphs is not reported since it cannot generate results within 24 hours. The time taken by GEDIOT, GEDGW and GEDGNN is comparable, whereas the time consumed by the ensemble method GEDHOT is about the summation of the time consumed by GEDIOT and GEDGW.



(a) ε_0 - MAE (b) ε_0 - Accuracy

Figure 17: Varying ε_0 in the Sinkhorn Algorithm



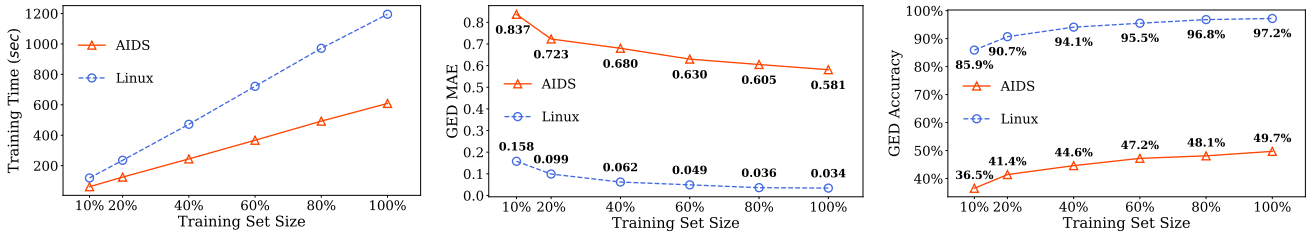
(a) λ - MAE (b) λ - Accuracy

Figure 19: Varying λ in the Loss Function

G.5 Ablation Study

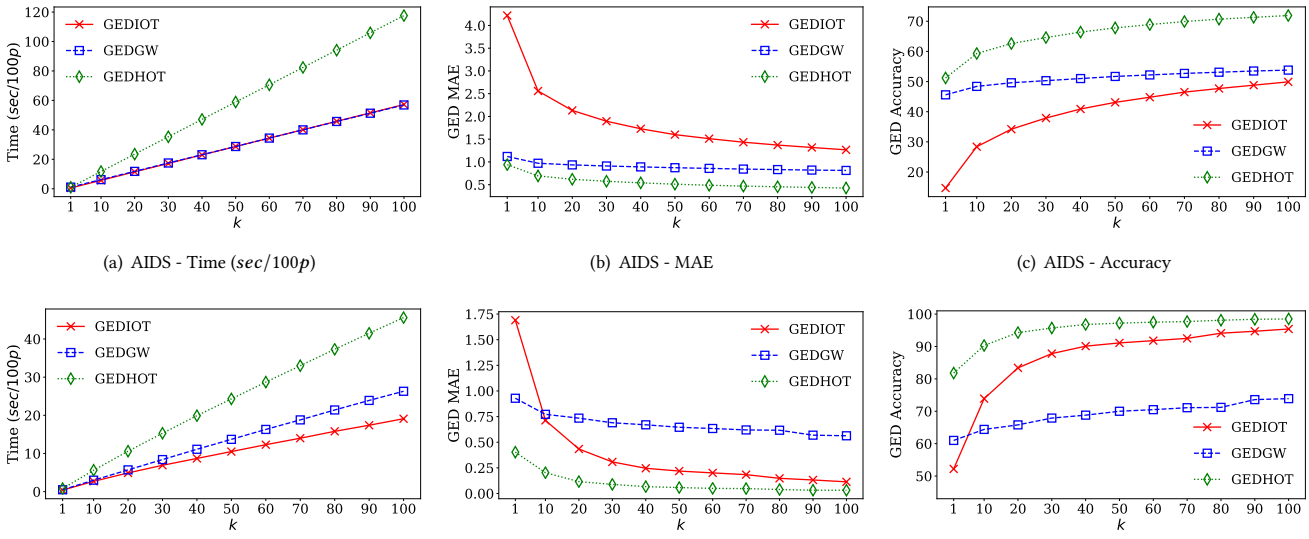
Varying Parameters in the Sinkhorn Algorithm. We also study how the performance of GEDIOT is impacted as the initial regularization coefficient, denoted by ε_0, and the number of iterations vary in the learnable Sinkhorn layer. The results are presented in Figure 17 and Figure 18. In Figure 17, we set ε_0 to 0.005, 0.01, 0.05, 0.1, 0.5, and 1 on AIDS and Linux. Both MAE and accuracy are stable with various ε_0, which shows the robustness of the learnable regularization method to ε_0. In Figure 18, we set the number of iterations to 1, 5, 10, 15, and 20 on AIDS and Linux. We can see that the MAE decreases and the accuracy increases as the number of iterations increases, but after 15 (resp. 10) iterations on AIDS (resp. Linux), the MAE and accuracy becomes fairly stable as the Sinkhorn algorithm converges. Note that the computational time also increases when conducting more iterations. Considering the time-accuracy tradeoff, we set the number of iterations to 5 by default.

Varying λ in the Loss Function. As presented in Figure 19, we also discuss the effect of varying λ (from 0 to 1) that balances the two terms $\mathcal{L}_m$ and $\mathcal{L}_o$ of the loss function in Eq. (15). The results



(a) Training Size - Time (b) Training Size - MAE (c) Training Size - Accuracy

Figure 20: Effect of Various Training Set Sizes on GEDIOT



(a) AIDS - Time ($sec/100p$) (b) AIDS - MAE (c) AIDS - Accuracy

(d) Linux - Time ($sec/100p$) (e) Linux - MAE (f) Linux - Accuracy

Figure 21: Varying k in k-Best Matching for GEP Generation

show that the performance improves with the increase of λ in $[0, 1]$ and becomes stable when λ is around 0.8. We set $\lambda = 0.8$ by default.

Varying the Size of Training Set. In this experiment, we evaluate the effect of varying the training set size. Concretely, we randomly sample 10%-100% of the original training set of AIDS and Linux to retrain GEDIOT. Figure 20 describes its influence on training time, MAE, and accuracy of GEDIOT. It can be observed that as the training set size increases, the MAE decreases and the accuracy increases, while the training time increases linearly. Furthermore, the observed trends of MAE and accuracy with increasing training set size appear to be flattening, which shows that training set size is sufficient.

k-Best Matching. We further verify the effect of k in k-best matching for GEP generation. As depicted in Figure 21, the MAE constantly decreases and the accuracy increases as the parameter k increases. Nevertheless, computational time also increases with the increase of k since the search space becomes larger.

H MORE DISCUSSION ON OUR METHODS

H.1 GED Computation on Edge-labeled Graphs

We here discuss how to handle the GED computation of edge-labeled graphs with GEDHOT. For GEDIOT, GINE [20] is a modified version of GIN that encodes the edge features, so we can replace GIN with GINE. For GEDGW, we can modify the 4-th order tensor $\mathcal{L}(\mathbf{A}^1, \mathbf{A}^2)$ (recall its definition from Table 1 and above Eq. (5)), which is regarding the cost of edge edit operations. Let $\ell(u_i, u_j)$ be the label of edge (u_i, u_j) and $\ell(u_i, u_j) = null$ if edge (u_i, u_j) does not exist. Given u_i, u_j in G^1 and v_k, v_l in G^2, we set $\mathcal{L}(\mathbf{A}^1_{i,j}, \mathbf{A}^2_{k,l})_{i,j,k,l} = 1$ if $\ell(u_i, u_j) \neq \ell(v_k, v_l)$, and 0 otherwise. This modified formulation can handle edge-labeled graphs.

H.2 Sizes of Parameters in GEDIOT

Like any machine learning model, we learn the model parameters during training, and these parameters are then directly used during test to provide predictions. Notably, our parameters are independent

of the graph sizes n_1 and n_2, so we do not need to do any hard-coding of n_1 and n_2. Specifically,

- The first network component is graph neural network (GIN in particular), where parameters are the MLP weight matrices that only depend on the input dimension d of node embeddings (see Eq. (8) and Eq. (9) in Section 4.1).
- The second network component is the cost matrix layer, which only has a parameter $\mathbf{W} \in \mathbb{R}^{d\times d}$ (see Eq. (10) in Section 4.2).
- The third network component is the learnable Sinkhorn layer, where the only learnable parameter is the regularization parameter ε, which is a scalar.
- The last network component is the graph discrepancy component described in Section 4.3, where there is a weight matrix $\mathbf{W}_1 \in \mathbb{R}^{d\times d}$ for graph pooling in Eq. (13), and parameters $\mathbf{W}_2^{[1:L]} \in \mathbb{R}^{L\times d\times d}$, $\mathbf{W}_3 \in \mathbb{R}^{L\times 2d}$ and $\mathbf{b} \in \mathbb{R}^{L}$ for NTN in Eq. (14). Here L is also a hyperparameter that is independent of graph size.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009