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

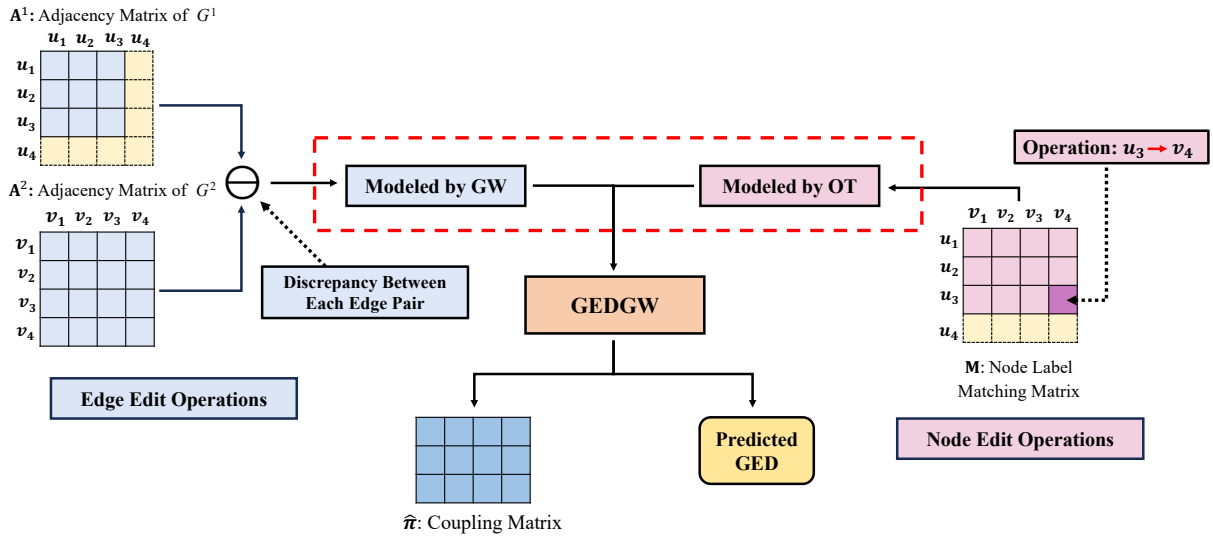


Figure 9: Diagram of the Proposed GEDGW

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].

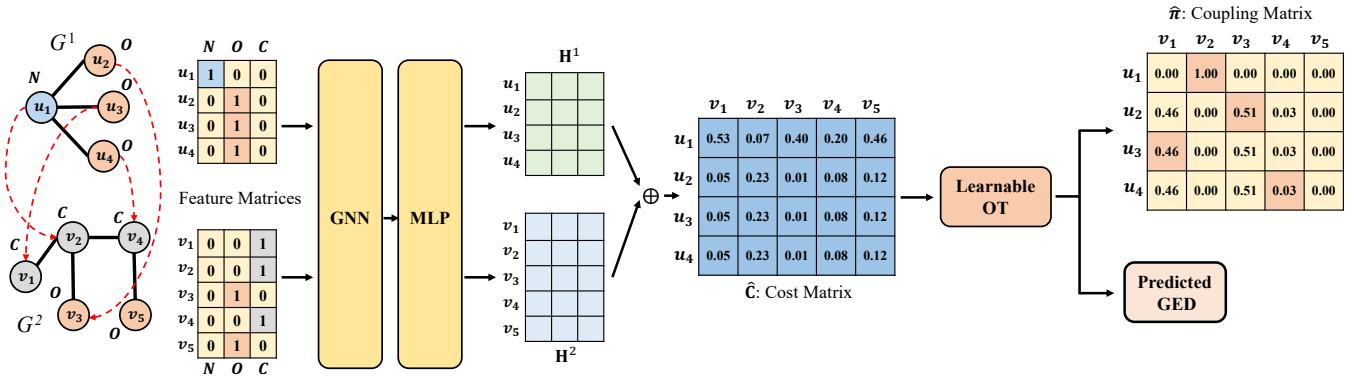


Figure 10: A Case Study for GEDIOT

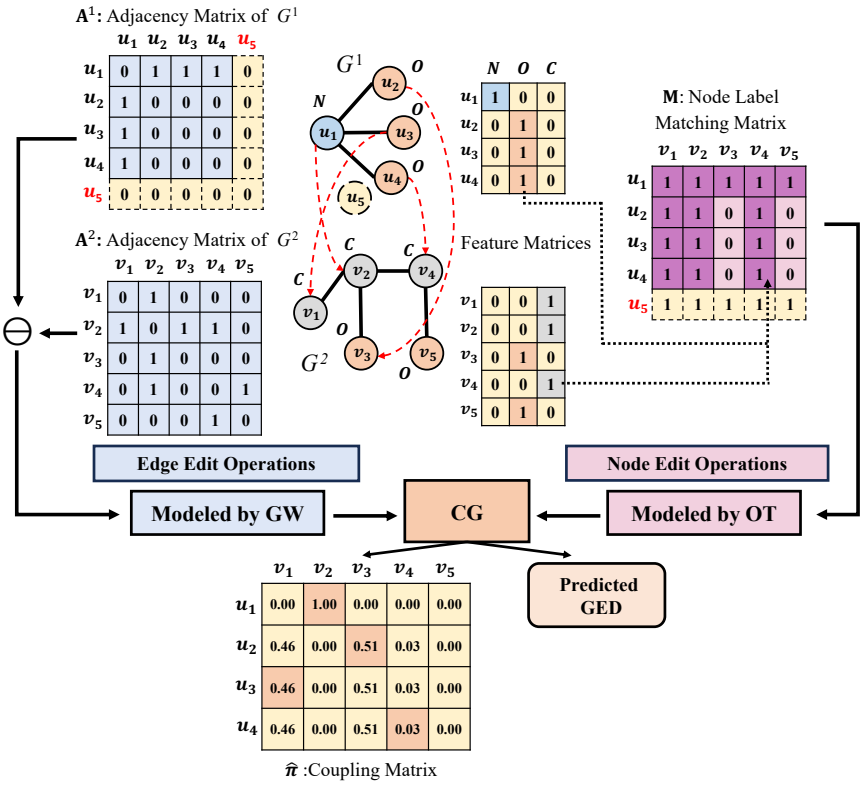


Figure 11: A Case Study for GEDGW

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.

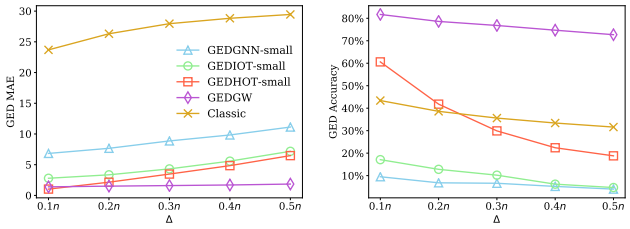
Training Set. Following the experimental settings of [36], we sample 60% graphs in each dataset to form the training set. Each sample in the training set is a graph pair. For AIDS and Linux, the two graphs in each graph pair in the training set are directly sampled from the graph dataset, since the GED 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

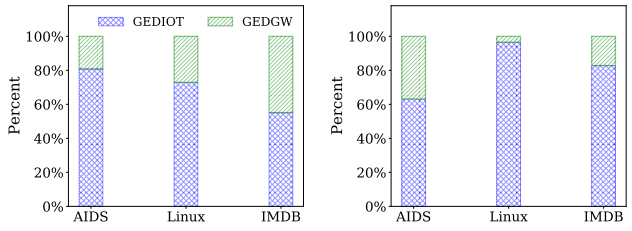
We further discuss how the generalizability of GEDGNN-small, GEDIOT-small, and GEDHOT-small is impacted when synthesizing large test graph pairs (more than 10 nodes) with larger GEDs (i.e., the discrepancy between two graphs becomes more pronounced). Concretely, for each original large graph with n nodes ($n > 10$) 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

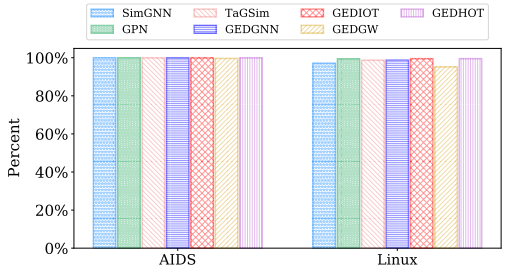


Figure 14: Triangle Property Preservation of the Predicted GEDs

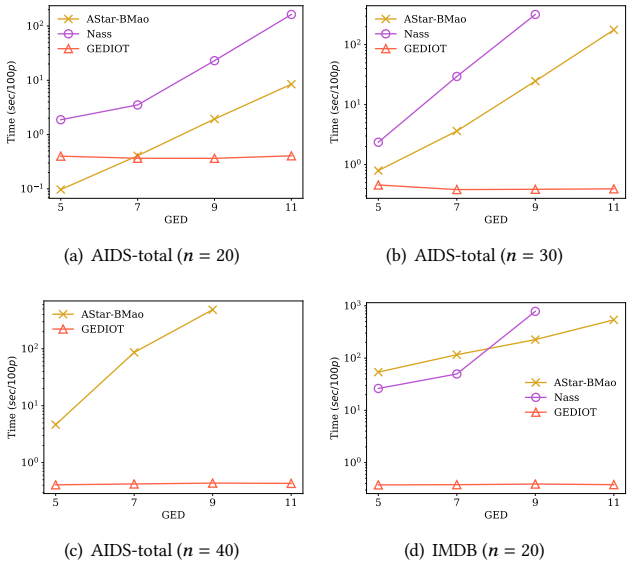
outperforms GEDIOT and vice versa. As shown in Figure 13, on AIDS, for GED computation, most graph pairs (80.8%) use the results from GEDIOT instead of GEDGW. For GEP generation, 63.1% of the graph pairs use the results from GEDIOT, and 36.9% of the graph pairs use the results from GEDGW. The results 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 ($n = 20$)

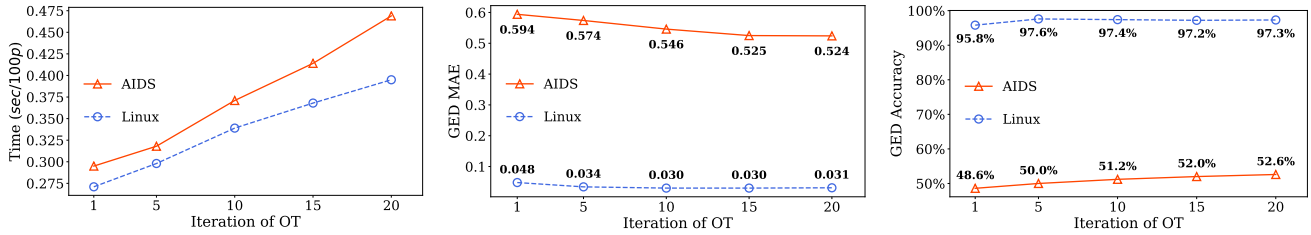
Figure 15: Efficiency Comparison with Exact Algorithms

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.

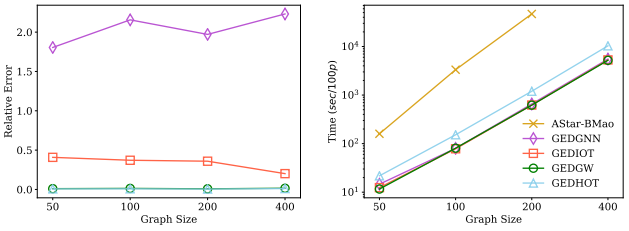
In Figure 15, we report the average running time of every 100 pairs for each group using the three methods. The computational time of the two exact methods Nass and AStar-BMao is quite sensitive w.r.t. the graph size and the GED value. We do not report the results of some groups of AStar-BMao and Nass 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)

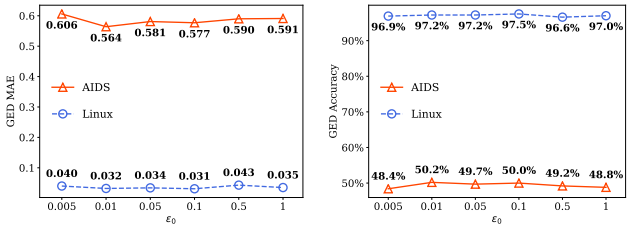
Figure 16: Accuracy and Efficiency on Power-law Graphs

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.

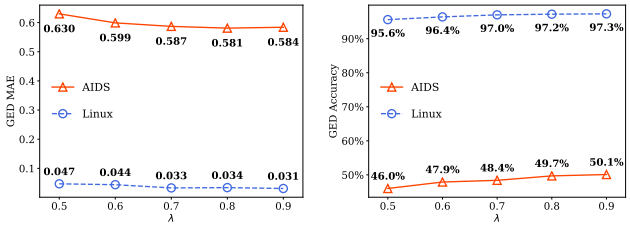
In Figure 16(a), we report the GED relative errors (i.e., $(\widehat{GED} - GED^*)/GED^*$) of the approximate methods with k-best matching framework: GEDGNN, GEDIOT, GEDGW, and GEDHOT. Note that the relative errors of all the methods are quite stable as the graph size n varies. Specifically, the relative 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



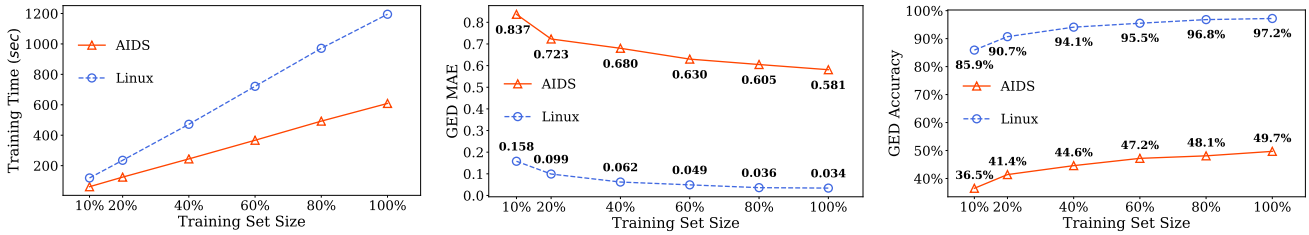
(a) λ - MAE (b) λ - Accuracy

Figure 19: Varying λ in the Loss Function

G.5 Ablation Study

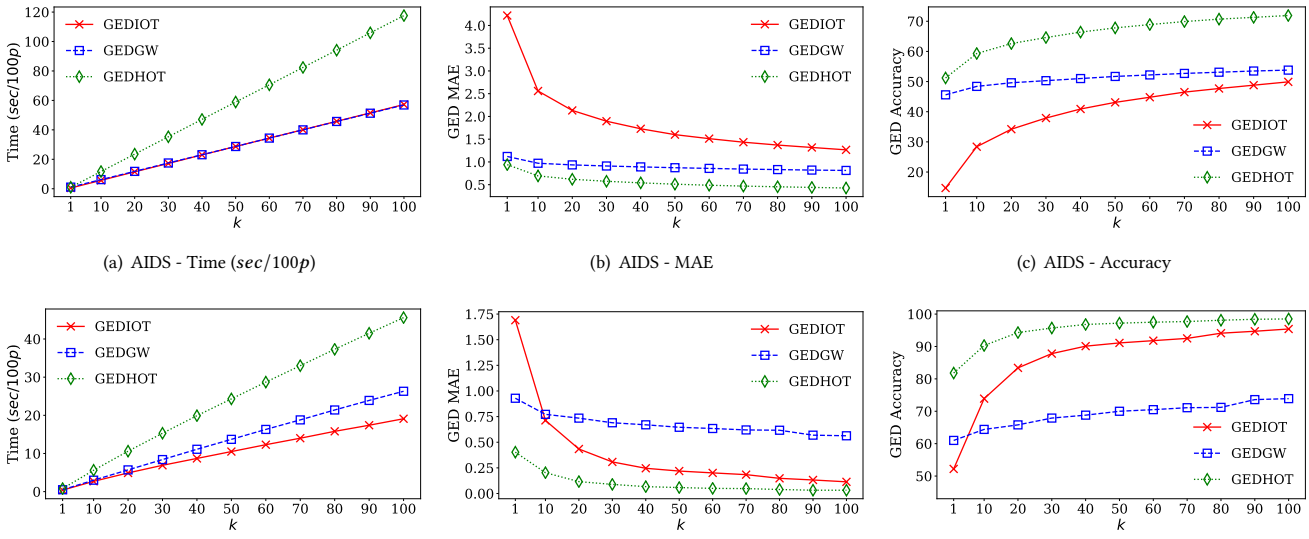
Varying Parameters in the Sinkhorn Algorithm. We also study how the performance of GEDIOT is impacted as the initial regularization coefficient, denoted by ε_0, and the number of iterations vary in the learnable Sinkhorn layer. The results are presented in Figure 17 and Figure 18. In Figure 17, we set 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