Graph Edit Distance with General Costs Using Neural Set Divergence

Eeshaan Jain* **Indradyumna Roy***
Saswat Meher **Soumen Chakrabarti** **Abir De**
IIT Bombay
eeshaanjain.iitb@gmail.com
{indraroy15, saswat, soumen, abir}@cse.iitb.ac.in

Abstract

Graph Edit Distance (GED) measures the (dis-)similarity between two given graphs, in terms of the minimum-cost edit sequence that transforms one graph to the other. However, the exact computation of GED is NP-Hard, which has recently motivated the design of neural methods for GED estimation. However, they do not explicitly account for edit operations with different costs. In response, we propose GraphEdX, a neural GED estimator that can work with general costs specified for the four edit operations, *viz.*, edge deletion, edge addition, node deletion and node addition. We first present GED as a quadratic assignment problem (QAP) that incorporates these four costs. Then, we represent each graph as a set of node and edge embeddings and use them to design a family of neural set divergence surrogates. We replace the QAP terms corresponding to each operation with their surrogates. Computing such neural set divergence require aligning nodes and edges of the two graphs. We learn these alignments using a Gumbel-Sinkhorn permutation generator, additionally ensuring that the node and edge alignments are consistent with each other. Moreover, these alignments are cognizant of both the presence and absence of edges between node-pairs. Experiments on several datasets, under a variety of edit cost settings, show that GraphEdX consistently outperforms state-of-the-art methods and heuristics in terms of prediction error.

1 Introduction

The Graph Edit Distance (GED) between a source graph, G, and a target graph, G', quantifies the minimum cost required to transform G into a graph isomorphic to G'. This transformation involves a sequence of edit operations, which can include node and edge insertions, deletions and substitutions. Each type of edit operation may incur a different and distinctive cost, allowing the GED framework to incorporate domain-specific knowledge. Its flexibility has led to the widespread use of GED for comparing graphs across diverse applications including graph retrieval [5, 6], pattern recognition [46], image and video indexing [50, 48] and chemoinformatics [21]. Because costs for addition and deletion may differ, GED is not necessarily symmetric, *i.e.*, $\mathrm{GED}(G, G') \neq \mathrm{GED}(G', G)$. This flexibility allows GED to model a variety of graph comparison scenarios, such as finding the Maximum Common Subgraph and checking for Subgraph Isomorphism [13]. In general, it is hard to even approximate GED [32]. Recent work [5, 6, 19, 55, 39] has leveraged graph neural networks (GNNs) to build neural models for GED computation, but many of these approaches cannot account for edit operations with different costs. Moreover, several approaches [40, 31, 55, 6] cast GED as the Euclidean distance between graph embeddings, leading to models that are overly attuned to cost-invariant edit sequences.

*Equal contribution.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

1.1 Present work

We propose a novel neural model for computing GED, designed to explicitly incorporate the various costs of edit operations. Our contributions are detailed as follows.

Neural set divergence surrogates for GED We formulate GED under general (non-uniform) cost as a quadratic assignment problem (QAP) with four asymmetric distance terms representing edge deletion, edge addition, node deletion and node addition. The edge-edit operations involve quadratic dependencies on a node alignment plan — a proposed mapping of nodes from the source graph to the target graph. To avoid the the complexity of QAP [44], we design a family of differentiable set divergence surrogates, which can replace the QAP objective with a more benign one. In this approach, each graph is represented as a set of embeddings of nodes and node-pairs (edges or non-edges). We replace the original QAP distance terms with their corresponding set divergences, and obtain the node alignment from a differentiable alignment generator modeled using a Gumbel-Sinkhorn network. This network produces a soft node permutation matrix based on contextual node embeddings from the graph pairs, enabling the computation of the overall set divergence in a differentiable manner, which facilitates end-to-end training. Our proposed model relies on late interaction, where the interactions between the graph pairs occur only at the final layer, rather than during the embedding computation in the GNN. This supports the indexing of embedding vectors, thereby facilitating efficient retrieval through LSH [25, 24, 12], inverted index [20], graph based ANN [34, 37] *etc*.

Learning all node-pair representations The optimal sequence of edits in GED is heavily influenced by the global structure of the graphs. A perturbation in one part of the graph can have cascading effects, necessitating edits in distant areas. To capture this sensitivity to structural changes, we associate both edges as well as non-edges with suitable expressive embeddings that capture the essence of subgraphs surrounding them. Note that the embeddings for non-edges are never explicitly computed during GNN message-passing operations. They are computed only once, after the GNN has completed its usual message-passing through *existing* edges, thereby minimizing additional computational overhead.

Node-edge consistent alignment To ensure edge-consistency in the learned node alignment map, we explicitly compute the node-pair alignment map from the node alignment map and then utilize this derived map to compute collective edge deletion and addition costs. More precisely, if $(u, v) \in G$ and $(u', v') \in G'$ are matched, then the nodes u and v are constrained to match with u' and v' (or, v' and u') respectively. We call our neural framework as GRAPHEDX.

Our experiments across several real datasets show that (1) GRAPHEDX outperforms several state-of-the-art methods including those that use early interaction; (2) the performance of current state-of-the-art methods improves significantly when their proposed distance measures are adjusted to reflect GED-specific distances, as in our approach.

2 Problem setup

Notation The source graph is denoted by $G = (V, E)$ and the target graph by $G' = (V', E')$. Both graphs are undirected and are padded with isolated nodes to equalize the number of nodes to N. The adjacency matrices for G and G' after padding are $A, A' \in \{0,1\}^{N\times N}$. (Note that we will use $M^\top$, not M', for the transpose of matrix M.) The sets of padded nodes in G and G' are denoted by PaddedNodes_G and $\text{PaddedNodes}_{G'}$ respectively. We construct $\eta \in \{0,1\}^N$, where $\eta[u] = 0$ if $u \in \text{PaddedNodes}_G$ and 1 otherwise (same for G'). The embedding of a node $u \in V$ computed at propagation layer k by the GNN, is represented as $x_k(u)$. Edit operations, denoted by edit, belong to one of four types, *viz.*, (i) node deletion, (ii) node addition, (iii) edge deletion, (iv) edge addition. Each operation edit is assigned a cost $\text{cost}(\text{edit})$. The node and node-pair alignment maps are described using (hard) permutation matrices $P \in \{0,1\}^{N\times N}$ and $S \in \{0,1\}^{\binom{N}{2}\times\binom{N}{2}}$ respectively. Given that the graphs are undirected, node-pair alignment need only be specified across at most $\binom{N}{2}$ pairs. When a hard permutation matrix is relaxed to a doubly-stochastic matrix, we call it a soft permutation matrix. We use P and S to refer to both hard and soft permutations, depending on the context. We denote $\mathbb{P}_N$ as the set of all hard permutation matrices of dimension N; $[N]$ as $\{1, \dots, N\}$ and $\|A\|_{1,1}$ to describe $\sum_{u,v} |A[u, v]|$. For two binary variables $c_1, c_2 \in \{0,1\}$, we denote $J(c_1, c_2)$ as (c_1 XOR c_2), *i.e.*, $J(c_1, c_2) = c_1c_2 + (1 - c_1)(1 - c_2)$.

Graph edit distance with general cost We define an *edit path* as a sequence of edit operations $\boldsymbol{o} = \{\text{edit}_1, \text{edit}_2, \ldots\}$; and $\mathcal{O}(G, G')$ as the set of all possible edit paths that transform the source graph G into a graph isomorphic to the target graph G'. Given $\mathcal{O}(G, G')$ and the cost associated with each operation edit, the GED between G and G' is the minimum collective cost across all edit paths in $\mathcal{O}(G, G')$. Formally, we write [14, 7]:

$$\text{GED}(G, G') = \min_{\boldsymbol{o}=\{\text{edit}_1,\text{edit}_2,\ldots\}\in\mathcal{O}(G,G')} \textstyle\sum_{i\in[|\boldsymbol{o}|]} \text{cost}(\text{edit}_i). \quad (1)$$

In this work, we assume a fixed cost for each of the four types of edit operations. Specifically, we use $a^\ominus$, $a^\oplus$, $b^\ominus$ and $b^\oplus$ to represent the costs for edge deletion, edge addition, node deletion, and node addition, respectively. These costs are not necessarily equal, in contrast to the assumptions made in previous works [5, 31, 55, 39]. Additional discussion on GED with node substitution in presence of labels can be found in Appendix D.

Problem statement Our objective is to design a neural architecture for predicting GED under a general cost framework, where the edit costs $a^\ominus$, $a^\oplus$, $b^\ominus$ and $b^\oplus$ are not necessarily the same. During the learning stage, these four costs are specified, and remain fixed across all training instances $\mathcal{D} = \{(G_i, G'_i, \text{GED}(G_i, G'_i))\}_{i\in[n]}$. Note that the edit paths are not supervised. Later, given a test instance G, G', assuming the same four costs, the trained system has to predict $\text{GED}(G, G')$.

3 Proposed approach

In this section, we first present an alternative formulation of GED as described in Eq. (1), where the edit paths are induced by node alignment maps. Then, we adapt this formulation to develop GraphEdX, a neural set distance surrogate, amenable to end-to-end training. Finally, we present the network architecture of GraphEdX.

3.1 GED computation using node alignment map

Given the padded graph pair G and G', deleting a node $u \in V$ can be viewed as aligning node u with some padded node $u' \in \text{PaddedNodes}_{G'}$. Similarly, adding a new node u' to G can be seen as aligning some padded node $u \in \text{PaddedNodes}_G$ with node $u' \in V'$. Likewise, adding an edge to G corresponds to aligning a non-edge $(u, v) \notin E$ with an edge $(u', v') \in G'$. Conversely, deleting an edge in G corresponds to aligning an edge $(u, v) \in G$ with a non-edge $(u', v') \notin G'$.

Therefore, $\text{GED}(G, G')$ can be defined in terms of a node alignment map. Let Π_N represent the set of all node alignment maps $\pi : [N] \to [N]$ from V to V'. Recall that $\eta_G[u] = 0$ if $u \in \text{PaddedNodes}_G$ and 1 otherwise.

$$\min_{\pi\in\Pi_N} \frac{1}{2}\sum_{u,v}\Big(a^\ominus \cdot \mathbb{I}\left[(u,v)\in E \wedge (\pi(u),\pi(v))\notin E'\right] + a^\oplus \cdot \mathbb{I}\left[(u,v)\notin E \wedge (\pi(u),\pi(v))\in E'\right]\Big)$$
$$+ \sum_u \Big(b^\ominus \cdot \eta_G[u]\,(1-\eta_{G'}[\pi(u)]) + b^\oplus \cdot (1-\eta_G[u])\,\eta_{G'}[\pi(u)]\Big). \quad (2)$$

In the above expression, the first sum iterates over all pairs of $(u, v) \in [N] \times [N]$ and the second sum iterates over $u \in [N]$. Because both graphs are undirected, the fraction $1/2$ accounts for double counting of the edges. The first and second terms quantify the cost of deleting and adding the edge (u, v) from and to G, respectively. The third and the fourth terms evaluate the cost of deleting and adding node u from and to G, respectively.

GED as a quadratic assignment problem In its current form, Eq. (2) cannot be immediately adapted to a differentiable surrogate. To circumvent this problem, we provide an equivalent matricized form of Eq. (2), using a hard node permutation matrix P instead of the alignment map π. We compute the asymmetric distances between A and $PA'P^\top$ and combine them with weights $a^\ominus$ and $a^\oplus$. Notably, $\text{ReLU}\left(A - PA'P^\top\right)[u, v]$ is non-zero if the edge $(u, v) \in E$ is mapped to a non-edge $(u', v') \in E'$ with $P[u, u'] = P[v, v'] = 1$, indicating deletion of the edge (u, v) from G. Similarly, $\text{ReLU}\left(PA'P^\top - A\right)[u, v]$ becomes non-zero if an edge (u, v) is added to G. Therefore, for the edit operations involving edges, we have:

$$\mathbb{I}\left[(u,v)\in E \wedge (\pi(u),\pi(v))\notin E'\right] = \text{ReLU}\left(A - PA'P^\top\right)[u,v], \quad (3)$$
$$\mathbb{I}\left[(u,v)\notin E \wedge (\pi(u),\pi(v))\in E'\right] = \text{ReLU}\left(PA'P^\top - A\right)[u,v]. \quad (4)$$

Similarly, we note that $\text{ReLU}\left(\eta_G[u] - \eta_{G'}[\pi(u)]\right) > 0$ if $u \notin \text{PaddedNodes}_G$ and $\pi(u) \in \text{PaddedNodes}_{G'}$, which allows us to compute the cost of deleting the node u from G. Similarly, we

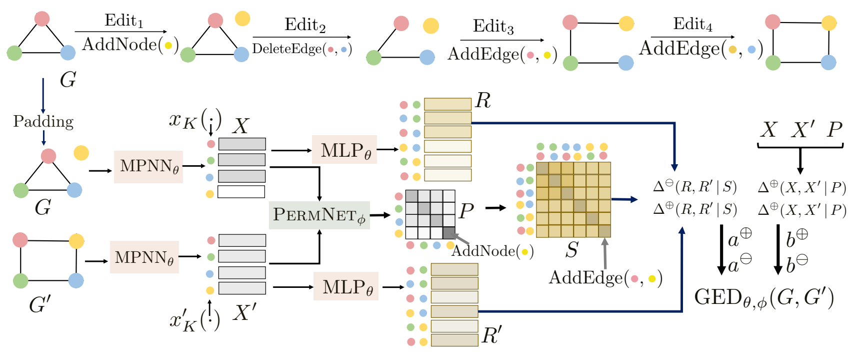


Figure 1: **Top:** Example graphs G and G' are shown with color-coded nodes to indicate alignment corresponding to the optimal edit path transforming G to G'. **Bottom:** GRAPHEDX's GED prediction pipeline. G and G' are independently encoded using MPNN_θ, and then padded with zero vectors to equalize sizes, resulting in contextual node representations $X, X' \in \mathbb{R}^{N\times d}$. For each node-pair, the corresponding embeddings and edge presence information are gathered and fed into MLP$_\theta$ to obtain $R, R' \in \mathbb{R}^{N(N-1)/2\times D}$. Simultaneously, X, X' are fed into PERMNET$_\phi$ to obtain the soft node alignment P (Eq.(18)) which constructs the node-pair alignment matrix $S \in \mathbb{R}^{N(N-1)/2\times N(N-1)/2}$ as $S[(u,v),(u',v')] = P[u,u']P[v,v'] + P[u,v']P[v,u']$. Finally, X, X', P are used to approximate node insertion and deletion costs, while R, R', S are used to approximate edge insertion and deletion costs. The four costs are summed to give the final prediction $\text{GED}_{\theta,\phi}(G, G')$ (Eq.(9)).

use $\text{ReLU}\left(\eta_{G'}[\pi(u)] - \eta_G[u]\right)$ to account for the addition of the node u to G. Formally, we write:

$$\eta_G[u]\left(1 - \eta_{G'}[\pi(u)]\right) = \text{ReLU}\left(\eta_G[u] - \eta_{G'}[\pi(u)]\right), \tag{5}$$

$$\left(1 - \eta_G[u]\right)\eta_{G'}[\pi(u)] = \text{ReLU}\left(\eta_{G'}[\pi(u)] - \eta_G[u]\right). \tag{6}$$

Using Eqs. (3)–(6), we rewrite Eq. (2) as:

$$\text{GED}(G, G') = \min_{P\in\mathbb{P}_N} \frac{a^{\ominus}}{2}\left\|\text{ReLU}\left(A - PA'P^\top\right)\right\|_{1,1} + \frac{a^{\oplus}}{2}\left\|\text{ReLU}\left(PA'P^\top - A\right)\right\|_{1,1} + b^{\ominus}\left\|\text{ReLU}\left(\eta_G - P\eta_{G'}\right)\right\|_1 + b^{\oplus}\left\|\text{ReLU}\left(P\eta_{G'} - \eta_G\right)\right\|_1. \tag{7}$$

The first and the second term denote the collective costs of deletion and addition of edges, respectively. The third and the fourth terms present a matricized representation of Eqs. (5)- (6). The above problem can be viewed as a quadratic assignment problem (QAP) on graphs, given that the hard node permutation matrix P has a quadratic involvement in the first two terms. Note that, in general, $\text{GED}(G, G') \neq \text{GED}(G', G)$. However, the optimal edit paths for these two GED values, encoded by the respective node permutation matrices, are inverses of each other, as formally stated in the following proposition (proven in Appendix D).

Proposition 1 *Given a fixed set of values of $b^{\ominus}, b^{\oplus}, a^{\ominus}, a^{\oplus}$, let P be an optimal node permutation matrix corresponding to* $\text{GED}(G, G')$*, computed using Eq.* (7)*. Then, $P' = P^\top$ is an optimal node permutation corresponding to* $\text{GED}(G', G)$*.*

Connection to different notions of graph matching The above expression of GED can be used to represent various notions of graph matching and similarity measures by modifying the edit costs. These include graph isomorphism, subgraph isomorphism, and maximum common edge subgraph detection. For example, by setting all costs to one, $\text{GED}(G, G') = \min_P \frac{1}{2}||A - PA'P^\top||_1 + ||\eta_G - P\eta_{G'}||_1$, which equals zero only when G and G' are isomorphic. Further discussion on this topic is provided in Appendix D.

3.2 GRAPHEDX model

Minimizing the objective in Eq. (7) is a challenging problem. In similar problems, recent methods have approximated the hard node permutation matrix P with a soft permutation matrix obtained using Sinkhorn iterations on a neural cost matrix. However, the binary nature of the adjacency matrix and the pad indicator q still impede the flow of gradients during training. To tackle this problem, we make relaxations in two key places within each term in Eq. (7), leading to our proposed GRAPHEDX model.

(1) We replace the binary values in $q_G, q_{G'}, A$ and A' with real values from node and node-pair embeddings: $X \in \mathbb{R}^{N\times d}$ and $R \in \mathbb{R}^{\binom{N}{2}\times D}$. These embeddings are computed using a GNN guided neural module $\textsc{Embed}_\theta$ with parameter θ. Since the graphs are undirected, R gathers the embeddings of the unique node-pairs, resulting in $\binom{N}{2}$ rows instead of N^2.

(2) We substitute the hard node permutation matrix P with a soft alignment matrix, generated using a differentiable alignment planner $\textsc{PermNet}_\phi$ with parameter ϕ. Here, P is a doubly stochastic matrix, with $P[u, u']$ indicating the "score" or "probability" of aligning $u \mapsto u'$. Additionally, we also compute the corresponding node-pair alignment matrix S.

Using these relaxations, we approximate the four edit costs in Eq. (7) with four continuous set distance surrogate functions.

$$\left\|\text{ReLU}\left(A - PA'P^\top\right)\right\|_{1,1} \to \Delta^\ominus(R, R' \,|\, S), \quad \left\|\text{ReLU}\left(PA'P^\top - A\right)\right\|_{1,1} \to \Delta^\oplus(R, R' \,|\, S),$$
$$\|\text{ReLU}\left(\eta_G - P\eta_{G'}\right)\|_1 \to \Delta^\ominus(X, X' \,|\, P), \quad \|\text{ReLU}\left(P\eta_{G'} - \eta_G\right)\|_1 \to \Delta^\oplus(X, X' \,|\, P). \tag{8}$$

This gives us an approximated GED parameterized by θ and ϕ.

$$\text{GED}_{\theta,\phi}(G, G') = a^\ominus \Delta^\ominus(R, R' \,|\, S) + a^\oplus \Delta^\oplus(R, R' \,|\, S) + b^\ominus \Delta^\ominus(X, X' \,|\, P) + b^\oplus \Delta^\oplus(X, X' \,|\, P). \tag{9}$$

Note that since R and R' contain the embeddings of each node-pair only once, there is no need to multiply $1/2$ in the first two terms, unlike Eq. (7). Next, we propose three types of neural surrogates to approximate each of the four operations.

(1) AlignDiff Given the node-pair embeddings R and R' for the graph pairs G and G', we apply the soft node-pair alignment S to R'. We then define the edge edits in terms of asymmetric differences between R and SR', which serves as a replacement for the corresponding terms in Eq. (7). We write $\Delta^\ominus(R, R' \,|\, S)$ and $\Delta^\oplus(R, R' \,|\, S)$ as:

$$\Delta^\ominus(R, R' \,|\, S) = \|\text{ReLU}\left(R - SR'\right)\|_{1,1}, \quad \Delta^\oplus(R, R' \,|\, S) = \|\text{ReLU}\left(SR' - R\right)\|_{1,1}. \tag{10}$$

Similarly, for the node edits, we can compute $\Delta^\ominus(X, X' \,|\, P)$ and $\Delta^\oplus(X, X' \,|\, P)$ as:

$$\Delta^\ominus(X, X' \,|\, P) = \|\text{ReLU}\left(X - PX'\right)\|_{1,1}, \quad \Delta^\oplus(X, X' \,|\, P) = \|\text{ReLU}\left(PX' - X\right)\|_{1,1}. \tag{11}$$

(2) DiffAlign In Eq. (10), we first aligned R' using S and then computed the difference from R. Instead, here we first computed the pairwise differences between R' and R for all pairs of node-pairs (e, e'), and then combine these differences with the corresponding alignment scores $S[e, e']$. We compute the edge edit surrogates $\Delta^\ominus(R, R' \,|\, S)$ and $\Delta^\oplus(R, R' \,|\, S)$ as:

$$\Delta^\ominus(R, R' \,|\, S) = \sum_{e,e'} \|\text{ReLU}\left(R[e,:] - R'[e',:]\right)\|_1 \, S[e, e'], \tag{12}$$
$$\Delta^\oplus(R, R' \,|\, S) = \sum_{e,e'} \|\text{ReLU}\left(R'[e',:] - R[e,:]\right)\|_1 \, S[e, e']. \tag{13}$$

Here, e and e' represent node-pairs, which are not necessarily edges. When the node-pair alignment matrix S is a hard permutation, $\Delta^\oplus$ and $\Delta^\ominus$ remain the same across AlignDiff and DiffAlign (as shown in Appendix D). Similar to Eqs. (12)—(13), we can compute $\Delta^\ominus(X, X' \,|\, P) = \sum_{u,u'} \|\text{ReLU}\left(X[u,:] - X'[u',:]\right)\|_1 \, P[u, u']$ and $\Delta^\oplus(X, X' \,|\, P) = \sum_{u,u'} \|\text{ReLU}\left(X'[u',:] - X[u,:]\right)\|_1 \, P[u, u']$.

(3) XOR-DiffAlign As indicated by the combinatorial formulation of GED in Eq. (7), the edit cost of a particular node-pair is non-zero only when an edge is mapped to a non-edge or vice-versa. However, the surrogates for the edge edits in AlignDiff or DiffAlign fail to capture this condition because they can assign non-zero costs to the pairs $(e = (u, v), e' = (u', v'))$ even when both e and e' are either edges or non-edges. To address this, we explicitly discard such pairs from the surrogates defined in Eqs. (12)–(13). This is ensured by applying a XOR operator $J(\cdot, \cdot)$ between the corresponding entries in the adjacency matrices, *i.e.*, $A[u, v]$ and $A'[u', v']$, and then multiplying this result with the underlying term. Hence, we write:

$$\Delta^\ominus(R, R' \,|\, S) = \sum_{e=(u,v)} \sum_{e'=(u',v')} J\big(A[u, v], A'[u', v']\big) \, \|\text{ReLU}\left(R[e,:] - R'[e',:]\right)\|_1 \, S[e, e'], \tag{14}$$
$$\Delta^\oplus(R, R' \,|\, S) = \sum_{e=(u,v)} \sum_{e'=(u',v')} J\big(A[u, v], A'[u', v']\big) \, \|\text{ReLU}\left(R'[e',:] - R[e,:]\right)\|_1 \, S[e, e']. \tag{15}$$

Similarly, the cost contribution for node operations arises from mapping a padded node to a non-padded node or vice versa. We account for this by multiplying $J(\eta_G[u], \eta_{G'}[u'])$ with each term of $\Delta^{\ominus}(X, X' \,|\, P)$ and $\Delta^{\oplus}(X, X' \,|\, P)$ computed using DiffAlign. Hence, we compute $\Delta^{\ominus}(X, X' \,|\, P) = \sum_{u,u'} J(\eta_G[u], \eta_{G'}[u']) \left\| \mathrm{ReLU}\left(X[u,:] - X'[u',:]\right)\right\|_1 P[u,u']$ and $\Delta^{\oplus}(X, X' \,|\, P) = \sum_{u,u'} J(\eta_G[u], \eta_{G'}[u']) \left\| \mathrm{ReLU}\left(X'[u',:] - X[u,:]\right)\right\|_1 P[u,u']$.

Comparison between AlignDiff, DiffAlign and XOR-DiffAlign AlignDiff and DiffAlign become equivalent when S is a hard permutation. However, when S is doubly stochastic, the above three surrogates, AlignDiff, DiffAlign and XOR-DiffAlign, are not equivalent. As we move from AlignDiff to DiffAlign to XOR-DiffAlign, we increasingly align the design to the inherent inductive biases of GED, thereby achieving a better representation of its cost structure.

Suppose we are computing the GED between two isomorphic graphs, G and G', with equal costs for all edit operations. In this scenario, we ideally expect a neural network to consistently output a zero cost. Now consider a proposed soft alignment S which is close to the optimal alignment. Under the AlignDiff design, the aggregated value $\sum_{e'} S[e,e']R[e',:]$ — where e and e' represent two edges matched in the optimal alignment — can accumulate over the large number of $N(N-1)/2$ node-pairs. This aggregation leads to high values of $||R[e,:] - SR'[e',:]||_1$, implying that AlignDiff captures an aggregate measure of the cost incurred by spurious alignments, but cannot disentangle the effect of individual misalignments. This makes it difficult for AlignDiff to learn the optimal alignment.

In contrast, the DiffAlign approach, which relies on pairwise differences between embeddings to explicitly guide S towards the optimal alignment, significantly ameliorates this issue. For example, in the aforementioned setting of GED with equal costs, the cost associated with each pairing (e, e') is explicitly encoded using $||R[e,:] - R'[e',:]||_1$, and is explicitly set to zero for pairs that are correctly aligned. Moreover, this representation allows DiffAlign to isolate the cost incurred by each misalignment, making it easier to train the model to reduce the cost of these spurious matches to zero.

However, DiffAlign does not explicitly set edge-to-edge and non-edge-to-non-edge mapping costs to zero, potentially leading to inaccurate GED estimates. XOR-DiffAlign addresses these concerns by applying a XOR of the adjacency matrices to the cost matrix, ensuring that non-zero cost is computed only when mapping an edge to a non-edge or vice versa. This resolves the issues in both AlignDiff and DiffAlign by focusing on mismatches between edges and non-edges, while disregarding redundant alignments that do not contribute to the GED.

Amenability to indexing and approximate nearest neighbor (ANN) search. All of the aforementioned distance surrogates are based on a late interaction paradigm, where the embeddings of G and G' are computed independently of each other before computing the distances Δ. This is particularly useful in the context of graph retrieval, as it allows for the corpus graph embeddings to be indexed a-priori, thereby enabling efficient retrieval of relevant graphs for new queries.

When the edit costs are equal, our predicted GED (9) becomes symmetric with respect to G and G'. In such cases, DiffAlign and AlignDiff yield a structure similar to the Wasserstein distance induced by L_1 norm. This allows us to leverage ANN techniques like Quadtree or Flowtree [4]. However, while the presence of the XOR operator J within each term in Eq. (14) – (15) of XOR-DiffAlign enhances the interaction between G and G', this same feature prevents XOR-DiffAlign from being cast to an ANN-amenable setup, unlike DiffAlign and AlignDiff.

3.3 Network architecture of $\textsc{Embed}_\theta$ and $\textsc{PermNet}_\phi$

In this section, we present the network architectures of the two components of GraphEdX, *viz.*, $\textsc{Embed}_\theta$ and $\textsc{PermNet}_\phi$, as introduced in items (1) and (2) in Section 3.2. Notably, in our proposed graph representation, non-edges and edges alike are embedded as non-zero vectors. In other words, all node-pairs are endowed with non-trivial embeddings. We then explain the design approach for edge-consistent node alignment.

Neural architecture of $\textsc{Embed}_\theta$ $\textsc{Embed}_\theta$ consists of a message passing neural network MPNN_θ and a decoupled neural module MLP_θ. Given the graphs G, G', MPNN_θ with K propagation layers is used to iteratively compute the node embeddings $\left\{x_K(u) \in \mathbb{R}^d \,|\, u \in V\right\}$ and $\left\{x'_K(u) \in \mathbb{R}^d \,|\, u \in V'\right\}$, then collect them into X and X' after padding, *i.e.*,

$$X := \{x_K(u) \,|\, u \in [N]\} = \mathrm{MPNN}_\theta(G), \qquad X' := \{x'_K(u') \,|\, u' \in [N]\} = \mathrm{MPNN}_\theta(G'). \quad (16)$$

The optimal alignment S is highly sensitive to the global structure of the graph pairs, *i.e.*, $S[e, e']$ can significantly change when we perturb G or G' in regimes distant from e or e'. Conventional representations mitigate this sensitivity while training models, by setting non-edges to zero, rendering them invariant to structural changes. To address this limitation, we utilize more expressive graph representations, where non-edges are also embedded using trainable non-zero vectors. This approach allows information to be captured from the structure around the nodes through both edges and non-edges, thereby enhancing the representational capacity of the embedding network. For each node-pair $e = (u, v) \in G$ (and equivalently (v, u)), and $e' = (u', v') \in G'$, the embeddings of the corresponding nodes and their connectivity status are concatenated, and then passed through an MLP to obtain the embedding vectors $r(e), r'(e') \in \mathbb{R}^D$. For $e = (u, v) \in G$, we compute $r(e)$ as follows:

$$r(e) = \text{MLP}_\theta(x_K(u) \,||\, x_K(v) \,||\, A[u, v]) + \text{MLP}_\theta(x_K(v) \,||\, x_K(u) \,||\, A[v, u]). \tag{17}$$

We can compute $r'(e)$ in similar manner. The property $r((u, v)) = r((v, u))$ reflects the undirected property of graph. Finally, the vectors $r(e)$ and $r'(e')$ are stacked into matrices R and R', both with dimensions $\mathbb{R}^{\binom{N}{2} \times D}$. We would like to highlight that $r((u, v))$ or $r'((u', v'))$ are computed only once for all node-pairs, after the MPNN completes its final Kth layer of execution. The message passing in the MPNN occurs only over edges. Therefore, this approach does not significantly increase the time complexity.

Neural architecture of PERMNET$_\phi$ The network PERMNET$_\phi$ provides P as a soft node alignment matrix by taking the node embeddings as input, *i.e.*, $P = \text{PERMNET}_\phi(X, X')$. PERMNET$_\phi$ is implemented in two steps. In the first step, we apply a neural network c_ϕ on both $\boldsymbol{x}_K$ and $\boldsymbol{x}'_K$, and then compute the normed difference between their outputs to construct the matrix C, where $C[u, u'] = \|c_\phi(x_K(u)) - c_\phi(x'_K(u'))\|_1$. Next, we apply iterative Sinkhorn normalizations [16, 35] on $\exp(-C/\tau)$, to obtain a soft node alignment P. Therefore,

$$P = \text{Sinkhorn}\left(\left[\exp\left(-\|c_\phi(x_K(u)) - c_\phi(x'_K(u'))\|_1 / \tau\right)\right]_{(u,u') \in [N] \times [N]}\right). \tag{18}$$

Here, τ is a temperature hyperparameter. In a general cost setting, GED is typically asymmetric, so it may be desirable for $C[u, u']$ to be asymmetric with respect to x and x'. However, as noted in Proposition 1, when we compute $\text{GED}(G', G)$, the alignment matrix $P' = \text{PERMNET}_\phi(X', X)$ should satisfy the condition that $P' = P^\top$, where P is computed from Eq. (18). The current form of C supports this condition, whereas an asymmetric form might not, as shown in Appendix D.

We construct $S \in \mathbb{R}^{\binom{N}{2}} \times \mathbb{R}^{\binom{N}{2}}$ as follows. Each pair of nodes (u, v) in G and (u', v') in G' can be mapped in two ways, regardless of whether they are edges or non-edges: (1) node $u \mapsto u'$ and $v \mapsto v'$ which is denoted by $P[u, u']P[v, v']$; (2) node $u \mapsto v'$ and $v \mapsto u'$, which is denoted by $P[u, v']P[v, u']$ Combining these two scenarios, we compute the node-pair alignment matrix S as: $S[(u, v), (u', v')] = P[u, u']P[v, v'] + P[u, v']P[v, u']$. This explicit formulation of S from P ensures mutually consistent permutation across nodes and node-pairs.

4 Experiments

We conduct extensive experiments on GRAPHEDX to showcase the effectiveness of our method across several real-world datasets, under both equal and unequal cost settings for GED. Additiional experimental results can be found in Appendix F.

4.1 Setup

Datasets We experiment with seven real-world datasets: Mutagenicity (Mutag) [18], Ogbg-Code2 (Code2) [23], Ogbg-Molhiv (Molhiv) [23], Ogbg-Molpcba (Molpcba) [23], AIDS [36], Linux [5] and Yeast [36]. For each dataset's training, test and validation sets $\mathcal{D}_{\text{split}}$, we generate $\binom{|\mathcal{D}_{\text{split}}|}{2} + |\mathcal{D}_{\text{split}}|$ graph pairs, considering combinations between every two graphs, including self-pairing. We calculate the exact ground truth GED using the F2 solver [29], implemented within GEDLIB [10]. For GED with equal cost setting, we set the cost values to $b^\ominus = b^\oplus = a^\ominus = a^\oplus = 1$. For GED with unequal cost setting, we use $b^\ominus = 3, b^\oplus = 1, a^\ominus = 2, a^\oplus = 1$. Further details on dataset generation and statistics are presented in Appendix E. In the main paper, we present results for the first five datasets under both equal and unequal cost settings for GED. Additional experiments for Linux and Yeast, as well as GED with node label substitutions, are presented in Appendix F.

Baselines We compare our approach with nine state-of-the-art methods. These include two variants of GMN [31]: (1) GMN-Match and (2) GMN-Embed; (3) ISONET [43], (4) GREED [40], (5)

	GED with equal cost					GED with unequal cost				
	Mutag	Code2	Molhiv	Molpcba	AIDS	Mutag	Code2	Molhiv	Molpcba	AIDS
GMN-Match [31]	0.797	1.677	1.318	1.073	0.821	69.210	13.472	76.923	23.985	31.522
GMN-Embed [31]	1.032	1.358	1.859	1.951	1.044	72.495	13.425	78.254	28.437	33.221
ISONET [43]	1.187	0.879	1.354	1.106	1.640	3.369	3.025	3.451	2.781	5.513
GREED [40]	1.398	1.869	1.708	1.550	1.004	68.732	11.095	78.300	26.057	34.354
ERIC [55]	0.719	1.363	1.165	0.862	0.731	1.981	12.767	3.377	2.057	1.581
SimGNN [5]	1.471	2.667	1.609	1.456	1.455	4.747	5.212	4.145	3.465	4.316
H2MN [53]	1.278	7.240	1.521	1.402	1.114	3.413	9.435	3.782	3.396	3.105
GraphSim [6]	2.005	3.139	2.577	1.656	1.936	5.370	7.405	6.643	3.928	5.266
EGSC [39]	0.765	4.165	1.138	0.938	0.627	1.758	3.957	2.371	2.133	1.693
GRAPHEDX	0.492	0.429	0.781	0.764	0.565	1.134	1.478	1.804	1.677	1.252

Table 2: Prediction error measured in terms of MSE of GRAPHEDX and all the state-of-the-art baselines across five datasets on 20% test set, for GED with equal costs and unequal costs. For GED with equal (unequal) costs we have $b^{\ominus} = b^{\oplus} = a^{\ominus} = a^{\oplus} = 1$ ($b^{\ominus} = 3, b^{\oplus} = 1, a^{\ominus} = 2, a^{\oplus} = 1$.) We select $\Delta^{\ominus}(R, R' \mid S), \Delta^{\oplus}(R, R' \mid S)$ and $\Delta^{\ominus}(X, X' \mid P), \Delta^{\oplus}(X, X' \mid P)$ from the cartesian space of Edge-$\{$AlignDiff, DiffAlign, XOR-DiffAlign$\}$ $\times$ Node-$\{$AlignDiff, DiffAlign, XOR-DiffAlign$\}$ through cross validation. Green (yellow) numbers report the best (second best) performers.

ERIC [55], (6) SimGNN [5], (7) H2MN [53], (8) GraphSim [6] and (9) EGSC [39]. To compute the GED, GMN-Match, GMN-Embed, and GREED use the Euclidean distance between the vector representation of two graphs. ISONET uses an asymmetric distance specifically tailored to subgraph isomorphism. H2MN is an early interaction network that utilizes higher-order node similarity through hypergraphs. ERIC, SimGNN, and EGSC leverage neural networks to calculate the distance between two graphs. Furthermore, the last three methods predict a score based on the normalized GED in the form of $\exp\left(-2\text{GED}(G, G')/(|V| + |V'|)\right)$. Notably, none of these baseline approaches have been designed to incorporate unequal edit costs into their models. To address this limitation, when working with GED under unequal cost setting, we include the edit costs as initial features in the graphs for all baseline models.

Evaluation Given a dataset $\mathcal{D} = \{(G_i, G'_i, \text{GED}(G_i, G'_i))\}_{i\in[n]}$, we divide it into training, validation and test folds with a split ratio of 60:20:20. We train the models using the Mean Squared Error (MSE) between the predicted GED and the ground truth GED as the loss. For model evaluation, we calculate the Mean Squared Error (MSE) between the actual and predicted GED on the test set. For ERIC, SimGNN and EGSC, we rescale the predicted score to obtain the true (unscaled) GED as $\text{GED}(G, G') = -(|V| + |V|') \log(s)/2$. In Appendix F, we also report Kendall's Tau (KTau) to evaluate the rank correlation across different experiments.

Selection of $\Delta^{\bullet}(X, X' \mid P)$ and $\Delta^{\bullet}(R, R' \mid S)$ We have three neural distance surrogates to choose from — AlignDiff, DiffAlign and XOR-DiffAlign — for both edge and node edits, resulting in nine possible combinations. We experiment with each of these nine combinations and select the one with the lowest validation error. However, as we will see later, the best performing surrogates always incorporate XOR-DiffAlign for edge edits. Consequently, one can limit the cross validation to only three surrogates for node edits, while using XOR-DiffAlign as the fixed surrogate for edge edits.

4.2 Results

Comparison with baselines We start by comparing the performance of GRAPHEDX against all state-of-the-art baselines for GED with both equal and unequal costs. Table 2 summarizes the results. We make the following observations. (1) GRAPHEDX outperforms all the baselines by a significant margin. For GED with equal costs, this margin often goes as high as 15%. This advantage becomes even more pronounced for GED with unequal costs, where our method outperforms the baselines by a margin as high as 30%, as seen in Code2. (2) There is no clear second-best method. Among the baselines, EGSC and ERIC each outperforms the others in two out of five datasets for both equal and unequal cost settings. Also, EGSC demonstrates competitive performance in AIDS.

Impact of cost-guided GED Among the baselines, GMN-Match, GMN-Embed and GREED compute GED using the euclidean distance between the graph embeddings, *i.e.*, $\text{GED}(G, G') = \|x_G - x_{G'}\|_2$, whereas we compute it by summing the set distance surrogates between the node and edge embedding sets. To understand the impact of our cost guided distance, we adapt it to the graph-level embeddings used by the above three baselines as follows: $\text{GED}(G, G') = \frac{b^{\ominus}+a^{\ominus}}{2} \left\|\text{ReLU}\left(x_G - x_{G'}\right)\right\|_1 + \frac{b^{\oplus}+a^{\oplus}}{2} \left\|\text{ReLU}\left(x_{G'} - x_G\right)\right\|_1$. Table 3 summarizes the results in terms of MSE, which shows that (1) cost guided distance reduces the MSE by a significant margin

	Equal cost			Unequal cost		
	Mutag	Code2	Molhiv	Mutag	Code2	Molhiv
GMN-Match	0.797	1.677	1.318	69.210	13.472	76.923
GMN-Match *	**0.654**	**0.960**	**1.008**	**1.592**	**2.906**	**2.162**
GMN-Embed	1.032	1.358	1.859	72.495	13.425	78.254
GMN-Embed *	**1.011**	**1.179**	**1.409**	**2.368**	**3.272**	**3.413**
GREED	**1.398**	1.869	1.708	68.732	11.095	78.300
GREED *	2.133	**1.850**	**1.644**	**2.456**	**5.429**	**3.827**
GRAPHEDX	0.492	0.429	0.781	1.134	1.478	1.804

Table 3: Impact of cost guided distance in terms of MSE; * represents the variant of the baseline with cost-guided distance. Green (**bold**) shows the best among all methods (only baselines).

	Equal cost			Unequal cost		
	Mutag	Code2	Molhiv	Mutag	Code2	Molhiv
Edge-only (edge $\rightarrow$ edge)	0.566	0.683	0.858	1.274	1.817	1.847
Edge-only (pair $\rightarrow$ pair)	0.596	0.760	0.862	1.276	1.879	1.865
GRAPHEDX	0.492	0.429	0.781	1.134	1.478	1.804

Table 4: Benefits of all node-pair representation MSE using only edges vs. all node-pair representations. Green (yellow) indicate the best (second best) performers.

in most cases; (2) even in the setting of GED with equal costs, our set divergence formulation is a better surrogate compared to the baselines (3) the margin of improvement is more prominent with GED involving unequal costs, where the modeling of specific cost values is crucial (4) GRAPHEDX outperforms the baselines even after changing their default distance to our cost guided distance.

Benefits of all node-pairs representation In this section, we compare the performance of using graph representation with two variants of our method. (i) Edge-only (edge $\rightarrow$ edge): Here, $R, R' \in \mathbb{R}^{\max(|E|,|E'|)\times D}$ are computed using only the embeddings of node-pairs that are edges, and excluding non-edges. This means that S becomes an edge-to-edge alignment matrix instead of a full node-pair alignment matrix. (ii) Edge-only (pair $\rightarrow$ pair): In this variant, S remains a node-pair alignment matrix, but the embeddings of the non-edges in $R, R' \in \mathbb{R}^{N(N-1)/2\times D}$ are explicitly set to zero. In Table 4, we report the results in terms of MSE, which show that (1) both these sparse representations perform significantly worse compared to our method using non-trivial representations for both edges and non-edges, and (2) Edge-only (edge $\rightarrow$ edge) performs better than Edge-only (pair $\rightarrow$ pair). This underscores the importance of explicitly modeling trainable non-edge embeddings to capture the sensitivity of GED to global graph structure.

Comparison across nine distances Here, we compare among the nine different combinations of our neural distance surrogates. Table 5 shows that the best combination mostly share the XOR-DiffAlign on the edge edit. This is because, XOR-DiffAlign offers more inductive bias, by zeroing the edit cost of aligning an edge to edge and a non-edge to non-edge, as we discussed in Section 3.2. There is no winner between AlignDiff and DiffAlign.

		Equal cost		Unequal cost	
Edge edit	Node edit	Mutag	Code2	Mutag	Code2
DiffAlign	DiffAlign	0.579	0.740	1.205	2.451
DiffAlign	AlignDiff	0.557	0.742	1.211	2.116
DiffAlign	XOR	0.538	0.719	1.146	1.896
AlignDiff	DiffAlign	0.537	0.513	1.185	1.689
AlignDiff	AlignDiff	0.578	0.929	1.338	1.488
AlignDiff	XOR	0.533	0.826	1.196	1.741
XOR	AlignDiff	0.492	0.429	1.134	1.478
XOR	DiffAlign	0.510	0.634	1.148	1.489
XOR	XOR	0.530	1.588	1.195	2.507

Table 5: Comparison among the nine neural distance combinations. Green (yellow) indicate the best (second best) performers in terms of MSE.

	Mutag	Code2	Molhiv	Molpcba	AIDS
GMN-Match	1.057	5.224	1.388	1.432	0.868
GMN-Embed	2.159	4.070	3.523	4.657	1.818
ISONET	0.876	1.129	1.617	1.332	1.142
GREED	2.876	4.983	2.923	3.902	2.175
ERIC	0.886	6.323	1.537	1.278	1.602
SimGNN	1.160	5.909	1.888	2.172	1.418
H2MN	1.277	6.783	1.891	1.666	1.290
GraphSim	1.043	4.708	1.817	1.748	1.561
EGSC	0.776	8.742	1.273	1.426	1.270
GRAPHEDX	0.441	0.820	0.792	0.846	0.538

Table 6: MSE for different methods with unit node substitution cost in equal cost setting. Green (yellow) show (second) best method.

Performance for GED under node substitution cost The scoring function in Eq. 9 can also be extended to incorporate node label substitution cost, which has been described in Appendix D. Here, we compare the performance of our model with the baselines in terms of MSE where we include node substitution cost $b^{\sim}$, with cost setting as $b^{\ominus} = b^{\oplus} = b^{\sim} = a^{\ominus} = a^{\oplus} = 1$. In Table 6, we report the results across 5 datasets equipped with node labels, passed as one-hot encoded node features. We observe that (1) our model outperforms all other baselines across all datasets by significant margin; (2) there is no clear second winner but ERIC, EGSC and ISONET performs better than the others.

5 Conclusion

Our work introduces a novel neural model for computing GED that explicitly incorporates general costs of edit operations. By leveraging graph representations that recognize both edges and non-edges, together with the design of suitable set distance surrogates, we achieve a more robust neural surrogate for GED. Our experiments demonstrate that this approach outperforms state-of-the-art methods, especially in settings with general edit costs, providing a flexible and effective solution for a range of applications. A potential limitation is that real-world applications often involve richly attributed graphs, where relevance based on GED might require separate formulations for modeling the structure of edit operations and the similarity of all node-pair features. Future work could focus on developing specialized formulations that integrate domain-specific knowledge, that improve effectiveness of GED-based graph comparison across various domains.

References

[1] J. Adler and S. Lunz. Banach wasserstein gan. *Advances in neural information processing systems*, 31, 2018.

[2] B. Amos, L. Xu, and J. Z. Kolter. Input convex neural networks. In *International Conference on Machine Learning*, pages 146–155. PMLR, 2017.

[3] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[4] A. Backurs, Y. Dong, P. Indyk, I. Razenshteyn, and T. Wagner. Scalable nearest neighbor search for optimal transport. In *International Conference on machine learning*, pages 497–506. PMLR, 2020.

[5] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, and W. Wang. Simgnn: A neural network approach to fast graph similarity computation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 384–392, 2019.

[6] Y. Bai, H. Ding, K. Gu, Y. Sun, and W. Wang. Learning-based efficient graph similarity computation via multi-scale convolutional set matching. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3219–3226, 2020.

[7] D. B. Blumenthal. New techniques for graph edit distance computation. *arXiv preprint arXiv:1908.00265*, 2019.

[8] D. B. Blumenthal and J. Gamper. Improved lower bounds for graph edit distance. *IEEE Transactions on Knowledge and Data Engineering*, 30:503–516, 2018. URL https://api.semanticscholar.org/CorpusID:3438059.

[9] D. B. Blumenthal, E. Daller, S. Bougleux, L. Brun, and J. Gamper. Quasimetric graph edit distance as a compact quadratic assignment problem. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 934–939, 2018. doi: 10.1109/ICPR.2018.8546055.

[10] D. B. Blumenthal, S. Bougleux, J. Gamper, and L. Brun. Gedlib: A c++ library for graph edit distance computation. In D. Conte, J.-Y. Ramel, and P. Foggia, editors, *Graph-Based Representations in Pattern Recognition*, pages 14–24, Cham, 2019. Springer International Publishing. ISBN 978-3-030-20081-7.

[11] S. Bougleux, L. Brun, V. Carletti, P. Foggia, B. Gaüzère, and M. Vento. Graph edit distance as a quadratic assignment problem. *Pattern Recognition Letters*, 87:38–46, 2017. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2016.10.001. URL https://www.sciencedirect.com/science/article/pii/S0167865516302665. Advances in Graph-based Pattern Recognition.

[12] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 327–336, 1998.

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

[14] H. Bunke and G. Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1(4):245–253, 1983.

[15] L. Chang, X. Feng, X. Lin, L. Qin, and W. Zhang. Efficient graph edit distance computation and verification via anchor-aware lower bound estimation. *arXiv preprint arXiv:1709.06810*, 2017.

[16] M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26:2292–2300, 2013.

[17] M. Daniels, T. Maunu, and P. Hand. Score-based generative neural networks for large-scale optimal transport. *Advances in neural information processing systems*, 34:12955–12965, 2021.

[18] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, 1991. doi: 10.1021/jm00106a046. URL https://doi.org/10.1021/jm00106a046.

[19] K. D. Doan, S. Manchanda, S. Mahapatra, and C. K. Reddy. Interpretable graph similarity computation via differentiable optimal alignment of node embeddings. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 665–674, 2021.

[20] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou. The faiss library. *arXiv preprint arXiv:2401.08281*, 2024.

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

[22] A. Genevay, M. Cuturi, G. Peyré, and F. Bach. Stochastic optimization for large-scale optimal transport. *Advances in neural information processing systems*, 29, 2016.

[23] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

[24] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

[25] P. Indyk, R. Motwani, P. Raghavan, and S. Vempala. Locality-preserving hashing in multidimensional spaces. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 618–625, 1997.

[26] S. Ivanov, S. Sviridov, and E. Burnaev. Understanding isomorphism bias in graph data sets. arXiv 1910.12091, 2019. URL https://arxiv.org/abs/1910.12091.

[27] D. Justice and A. Hero. A binary linear programming formulation of the graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1200–1214, 2006.

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

[29] J. Lerouge, Z. Abu-Aisheh, R. Raveaux, P. Héroux, and S. Adam. New binary linear programming formulation to compute the graph edit distance. *Pattern Recognition*, 72:254–265, 2017. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2017.07.029. URL https://www.sciencedirect.com/science/article/pii/S003132031730300X.

[30] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[31] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli. Graph matching networks for learning the similarity of graph structured objects. In *International conference on machine learning*, pages 3835–3845. PMLR, 2019.

[32] C.-L. Lin. Hardness of approximating graph transformation problem. In D.-Z. Du and X.-S. Zhang, editors, *Algorithms and Computation*, pages 74–82, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg. ISBN 978-3-540-48653-4.

[33] Z. Lou, J. You, C. Wen, A. Canedo, J. Leskovec, et al. Neural subgraph matching. *arXiv preprint arXiv:2007.03092*, 2020.

[34] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018.

[35] G. Mena, D. Belanger, S. Linderman, and J. Snoek. Learning latent permutations with gumbel-sinkhorn networks. *arXiv preprint arXiv:1802.08665*, 2018. URL https://arxiv.org/pdf/1802.08665.pdf.

[36] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. Tudataset: A collection of benchmark datasets for learning with graphs, 2020.

[37] B. Naidan, L. Boytsov, Y. Malkov, and D. Novak. Non-metric space library manual. *arXiv preprint arXiv:1508.05470*, 2015.

[38] E. Ozdemir and C. Gunduz-Demir. A hybrid classification model for digital pathology using structural and statistical pattern recognition. *IEEE Transactions on Medical Imaging*, 32(2): 474–483, 2013. doi: 10.1109/TMI.2012.2230186.

[39] C. Qin, H. Zhao, L. Wang, H. Wang, Y. Zhang, and Y. Fu. Slow learning and fast inference: Efficient graph similarity computation via knowledge distillation. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[40] R. Ranjan, S. Grover, S. Medya, V. Chakaravarthy, Y. Sabharwal, and S. Ranu. Greed: A neural framework for learning graph distance functions. *Advances in Neural Information Processing Systems*, 35:22518–22530, 2022.

[41] K. Riesen and H. Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, 27(7):950–959, 2009. ISSN 0262-8856. doi: https://doi.org/10.1016/j.imavis.2008.04.004. URL https://www.sciencedirect.com/science/article/pii/S026288560800084X. 7th IAPR-TC15 Workshop on Graph-based Representations (GbR 2007).

[42] K. Riesen, A. Fischer, and H. Bunke. Combining bipartite graph matching and beam search for graph edit distance approximation. In *Artificial Neural Networks in Pattern Recognition: 6th IAPR TC 3 International Workshop, ANNPR 2014, Montreal, QC, Canada, October 6-8, 2014. Proceedings 6*, pages 117–128. Springer, 2014.

[43] I. Roy, V. S. Velugoti, S. Chakrabarti, and A. De. Interpretable neural subgraph matching for graph retrieval. *AAAI*, 2022.

[44] S. Sahni and T. Gonzalez. P-complete approximation problems. *J. ACM*, 23(3):555–565, jul 1976. ISSN 0004-5411. doi: 10.1145/321958.321975. URL https://doi.org/10.1145/321958.321975.

[45] A. Sanfeliu and K.-S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE transactions on systems, man, and cybernetics*, pages 353–362, 1983.

[46] A. Sanfeliu, R. Alquézar, J. Andrade, J. Climent, F. Serratosa, and J. Vergés. Graph-based representations and techniques for image processing and image analysis. *Pattern recognition*, 35(3):639–650, 2002.

[47] V. Seguy, B. B. Damodaran, R. Flamary, N. Courty, A. Rolet, and M. Blondel. Large-scale optimal transport and mapping estimation. *arXiv preprint arXiv:1711.02283*, 2017.

[48] K. Shearer, H. Bunke, and S. Venkatesh. Video indexing and similarity retrieval by largest common subgraph detection using decision trees. *Pattern Recognition*, 34(5):1075–1091, 2001.

[49] R. Sinkhorn and P. Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

[50] S. Tirthapura, D. Sharvit, P. Klein, and B. B. Kimia. Indexing based on edit-distance matching of shape graphs. In *Multimedia storage and archiving systems III*, volume 3527, pages 25–36. SPIE, 1998.

[51] Y. Xie, M. Chen, H. Jiang, T. Zhao, and H. Zha. On scalable and efficient computation of large scale optimal transport. In *International Conference on Machine Learning*, pages 6882–6892. PMLR, 2019.

[52] Z. Zeng, A. K. Tung, J. Wang, J. Feng, and L. Zhou. Comparing stars: On approximating graph edit distance. *Proceedings of the VLDB Endowment*, 2(1):25–36, 2009.

[53] Z. Zhang, J. Bu, M. Ester, Z. Li, C. Yao, Z. Yu, and C. Wang. H2mn: Graph similarity learning with hierarchical hypergraph matching networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 2274–2284, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383325. doi: 10.1145/3447548.3467328. URL https://doi.org/10.1145/3447548.3467328.

[54] W. Zheng, L. Zou, X. Lian, D. Wang, and D. Zhao. Efficient graph similarity search over large graph databases. *IEEE Transactions on Knowledge and Data Engineering*, 27(4):964–978, 2014.

[55] W. Zhuo and G. Tan. Efficient graph similarity computation with alignment regularization. *Advances in Neural Information Processing Systems*, 35:30181–30193, 2022.

Graph Edit Distance with General Costs Using Neural Set Divergence (Appendix)

A Limitations

Our neural model for GED affords significant improvements in accuracy and flexibility for modeling edit costs. However, there are some limitations to consider.

(1) While computing graph representations over $\binom{N}{2} \times \binom{N}{2}$ node-pairs does not require additional parameters due to parameter-sharing, it does demand significant memory resources. This could pose challenges, especially with larger-sized graphs.

(2) The assumption of fixed edit costs across all graph pairs within a dataset might not reflect real-world scenarios where costs vary based on domain-specific factors and subjective human relevance judgements. This calls for more specialized approaches to accurately model the impact of each edit operation, which may differ across node pairs.

(3) the current model may not adequately address richly attributed graphs with complex node and edge features. Incorporating such attributes alongside graph structure based GED computation may require further exploration.

B Broader impact

Graphs serve as powerful representations across diverse domains, capturing complex relationships and structural notions inherent in various systems. From biological networks to social networks, transportation networks, and supply chains, graphs provide a versatile framework for modeling interactions between interconnected entities. In domains where structure-similarity based applications are prevalent, GED emerges as a valuable and versatile tool.

For example, in bio-informatics, molecular structures can naturally be represented as graphs. GED computation expedites tasks such as drug discovery, protein-protein interaction modeling, and molecular similarity analysis by identifying structurally similar molecular compounds. Similarly, in social network analysis, GED can measure similarities between user interactions, aiding in friend recommendation systems or community detection tasks. In transportation networks, GED-based tools assess similarity between road networks for route planning or traffic optimizations. Further applications include learning to edit scene graphs, analyzing gene regulatory pathways, fraud detection, and more

Moreover, our proposed variations of GED, particularly those amenable to hashing, find utility in retrieval based setups. In various information retrieval systems, hashed graph representations can be used to efficiently index and retrieve relevant items using our GED based scores. Such applications include image retrieval from image databases where images are represented as scene graphs, retrieval of relevant molecules from molecular databases, *etc*.

Furthermore, our ability to effectively model different edit costs in GED opens up new possibilities in various applications. In recommendation systems, it can model user preferences of varying importance, tailoring recommendations based on user-specific requirements or constraints. Similarly, in image or video processing, different types of distortions may have varying impacts on perceptual quality, and GED with adaptive costs can better assess similarity. In NLP tasks such as text similarity understanding and document clustering, assigning variable costs to textual edits corresponding to word insertion, deletions or substitutions, provides a more powerful framework for measuring textual similarity, improving performance in downstream tasks such as plagiarism detection, summarization, *etc*.

Lastly, and most importantly, the design of our model encourages interpretable alignment-driven justifications, thereby promoting transparency and reliability while minimizing potential risks and negative impacts, in high stake applications like drug discovery.

C Discussion on related work

Heuristics for Graph Edit Distance GED was first introduced in [45]. Bunke and Allermann [14] used it as a tool for non exact graph matching. Later on, [13] connected GED with maximum common subgraph estimation. Blumenthal [7] provide an excellent survey. As they suggest, combinatorial heuristics to solve GED predominantly follows three approaches: (1) Linear sum assignment problem with error-correction, which include [27, 41, 52, 54] (2) Linear programming, which predominantly uses standard tools like Gurobi, (3) Local search [42]. However, they can be extremely time consuming, especially for a large number of graph pairs. Among them Zheng et al. [54] operate in our problem setting, where the cost of edits are different across the edit operations, but for the same edit operation, the cost is same across node or node pairs.

Optimal transport In our work, we utilize Graph Neural Networks (GNNs) to represent each graph as a set of node embeddings. This transforms the inherent Quadratic Assignment Problem (QAP) of graph matching into a Linear Sum Assignment Problem (LSAP) on the sets of node embeddings. Essentially, this requires solving an optimal transport problem in the node embedding space. The use of neural surrogates for optimal transport was first proposed by Cuturi [16], who introduced entropy regularization to make the optimal transport objective strictly convex and utilized Sinkhorn iterations [49] to obtain the transport plan. Subsequently, Mena et al. [35] proposed the neural Gumbel Sinkhorn network as a continuous and differentiable surrogate of a permutation matrix, which we incorporate into our model.

In various generative modeling applications, optimal transport costs are used as loss functions, such as in Wasserstein GANs [1, 3]. Computing the optimal transport plan is a significant challenge, with approaches leveraging the primal formulation [51, 33], the dual formulation with entropy regularization [17, 47, 22], or Input Convex Neural Networks (ICNNs) [2].

Neural graph similarity computation Most earlier works on neural graph similarity computation have focused on training with GED values as ground truth [5, 6, 19, 40, 55, 39, 53, 31], while some have used MCS as the similarity measure [6, 5]. Current neural models for GED approximation primarily follow two approaches. The first approach uses a trainable nonlinear function applied to graph embeddings to compute GED [5, 39, 6, 55, 53, 19]. The second approach calculates GED based on the Euclidean distance in the embedding space [31, 40].

Among these models, GOTSIM [19] focuses solely on node insertion and deletion, and computes node alignment using a combinatorial routine that is decoupled from end-to-end training. However, their network struggles with training efficiency due to the operations on discrete values, which are not amenable to backpropagation. With the exception of GREED [40] and Graph Embedding Network (GEN) [31], most methods use early interaction or nonlinear scoring functions, limiting their adaptability to efficient indexing and retrieval pipelines

D Discussion on our proposed formulation of GED

D.1 Modification of scoring function from label substitution

To incorporate the effect of node substitution into account when formulating the GED, we first observe that the effect of node substitution cost $b^{\sim}$ only comes into account when a non-padded node maps to a non-padded node. In all other cases, when a node is deleted or inserted, we do not additionally incur any substitution costs. Note that, we consider the case when node substitution cannot be replaced by node addition and deletion, *i.e.*, $b^{\sim} \leq b^{\ominus} + b^{\oplus}$. Such a constraint on costs has uses in multiple applications [9, 38]. Let $\mathcal{L}$ denote the set of node labels, and $\ell(u)$, $\ell'(u') \in \mathcal{L}$ denote the node label corresponding to nodes u and u' in G and G' respectively. We construct the node label matrix L for G as follows: $L \in \{0,1\}^{N\times|\mathcal{L}|}$, such that $L[i,:] = \texttt{one\_hot}(\ell(i))$, *i.e.*, L is the one-hot indicator matrix for the node labels, which each row corresponding to the one-hot vector of the label. Similarly, we can construct L' for G'. Then, the distance between labels of two nodes $u \in V$ and $u' \in V'$ can be given as $\|L[u,:] - L'[u',:]\|_1$. To ensure that only valid node to node mappings contribute to the cost, we multiply the above with $\Lambda(u,u') = \text{AND}(\eta_G[u], \eta_{G'}[u'])$. This allows us to write the expression for GED with node label substitution cost as

$$\begin{aligned}\text{GED}(G,G') = \min_{P\in\mathbb{P}_N} & \frac{a^{\ominus}}{2}\left\|\text{ReLU}\left(A - PA'P^{\top}\right)\right\|_{1,1} + \frac{a^{\oplus}}{2}\left\|\text{ReLU}\left(PA'P^{\top} - A\right)\right\|_{1,1} \\ & + b^{\ominus}\left\|\text{ReLU}\left(Pq_{G'} - q_G\right)\right\|_1 + b^{\oplus}\left\|\text{ReLU}\left(q_G - Pq_{G'}\right)\right\|_1 \\ & + b^{\sim}\underbrace{\sum_{u,u'}\Lambda(u,u')\left\|L[u,:] - L[u',:]\right\|_1 P[u,u']}_{\Delta^{\sim}(L,L'|P)}\end{aligned}$$

We can design a neural surrogate for above in the same way as done in Section 3.2, and write

$$\begin{aligned}\text{GED}_{\theta,\phi}(G,G') = & \, a^{\ominus}\Delta^{\ominus}(R,R'\,|\,S) + a^{\oplus}\Delta^{\oplus}(R,R'\,|\,S) \\ & + b^{\ominus}\Delta^{\ominus}(X,X'\,|\,P) + b^{\oplus}\Delta^{\oplus}(X,X'\,|\,P) \\ & + b^{\sim}\Delta^{\sim}(L,L'|P)\end{aligned} \tag{19}$$

In this case, to account for node substitutions in the proposed permutation, we use $L[u,:]$ and $L'[u',:]$ as the features for node u in G and node u' in G', respectively. We present the comparison of our method including subsitution cost with state-of-the-art baselines in Appendix F.

D.2 Proof of Proposition 1

Proposition Given a fixed set of values of $b^{\ominus}, b^{\oplus}, a^{\ominus}, a^{\oplus}$, let P be an optimal node permutation matrix corresponding to $\text{GED}(G,G')$, computed using Eq. (7). Then, $P' = P^{\top}$ is an optimal node permutation corresponding to $\text{GED}(G',G)$.

Proof: Noticing that $\text{ReLU}\,(c-d) = \max(c,d) - d$, we can write

$$\begin{aligned}\left\|\text{ReLU}\left(A - PA'P^{\top}\right)\right\|_{1,1} &= \left\|\max(A, PA'P^{\top}) - PA'P^{\top}\right\|_{1,1} \\ &= \left\|\max(A, PA'P^{\top})\right\|_{1,1} - 2|E'|\end{aligned}$$

The last equality follows since $\max(A, PA'P^{\top}) \geq PA'P^{\top}$ element-wise, and $\left\|PA'P^{\top}\right\|_{1,1} = \|A'\|_{1,1} = 2|E'|$. Similarly, we can rewrite $\left\|\text{ReLU}\left(PA'P^{\top} - A\right)\right\|_{1,1}$, $\|\text{ReLU}\left(\eta_G - P\eta_{G'}\right)\|_1$, and $\|\text{ReLU}\left(P\eta_{G'} - q_G\right)\|_1$, and finally rewrite Eq. (7) as

$$\begin{aligned}\text{GED}(G,G') = \min_{P\in\mathbb{P}_N} & \frac{a^{\oplus}+a^{\ominus}}{2}\left\|\max(A, PA'P^{\top})\right\|_{1,1} - a^{\ominus}|E'| - a^{\oplus}|E| \\ & + \frac{b^{\oplus}+b^{\ominus}}{2}\|\max(\eta_G, P\eta_{G'})\|_1 - b^{\ominus}|V'| - b^{\oplus}|V|\end{aligned} \tag{20}$$

$$\begin{aligned}\text{GED}(G',G) = \min_{P\in\mathbb{P}_N} & \frac{a^{\oplus}+a^{\ominus}}{2}\left\|\max(A', PAP^{\top})\right\|_{1,1} - a^{\ominus}|E| - a^{\oplus}|E'| \\ & + \frac{b^{\oplus}+b^{\ominus}}{2}\|\max(\eta_{G'}, P\eta_G)\|_1 - b^{\ominus}|V| - b^{\oplus}|V'|\end{aligned} \tag{21}$$

We can rewrite the max term as follows:

$$\begin{aligned}
\left\|\max(A, PA'P^\top)\right\|_{1,1} &= \sum_{u,v} \max(A, PA'P^\top)[u,v] \\
&= \sum_{u,v} \max(PP^\top APP^\top, PA'P^\top)[u,v] \\
&= \sum_{u,v} P\max(P^\top AP, A')P^\top[u,v] \\
&= \sum_{u,v} \max(P^\top AP, A')[u,v] \\
&= \left\|\max(P^\top AP, A')\right\|_{1,1} = \left\|\max(A', P^\top AP)\right\|_{1,1}
\end{aligned}$$

Similarly we can re write $\|\max(\eta_G, P\eta_{G'})\|_1$ as $\left\|\max(\eta_{G'}, P^\top \eta_G)\right\|_1$. Given a fixed set of cost function $b^\ominus, b^\oplus, a^\ominus, a^\oplus$, the terms containing $|E'|, |E|, |V'|, |V|$ are constant and do not affect choosing an optimal P. Let $C = -a^\ominus|E'| - a^\oplus|E| - b^\ominus|V| - b^\oplus|V'|$, Using the above equations, we can write:

$$\begin{aligned}
&\frac{a^\oplus + a^\ominus}{2}\left\|\max(A, PA'P^\top)\right\|_{1,1} + \frac{b^\oplus + b^\ominus}{2}\|\max(\eta_G, P\eta_{G'})\|_1 \\
&= \frac{a^\oplus + a^\ominus}{2}\left\|\max(A', P^\top AP)\right\|_{1,1} + \frac{b^\oplus + b^\ominus}{2}\left\|\max(\eta_{G'}, P^\top \eta_G)\right\|_1
\end{aligned}$$

Let the first term be $\rho(G, G' \mid P)$. Then second term can be expressed as $\rho(G', G \mid P^\top)$ and $\rho(G, G' \mid P) = \rho(G', G \mid P^\top)$ for all $P \in \mathbb{P}_N$. If P is the optimal solution of $\min_{P\in\mathbb{P}_N} \rho(G, G' \mid P)$ then, $\rho(G', G \mid P^\top) = \rho(G, G' \mid P) \le \rho(G, G' \mid \widetilde{P}^\top) = \rho(G', G \mid \widetilde{P})$ for any permutation $\widetilde{P}$. Hence, $P' = P^\top \in \mathbb{P}_N$ is one optimal solution.

D.3 Connections with other notions of graph matching

Graph isomorphism: When we set all costs to zero, we can write that $\mathrm{GED}(G, G') = \min_P 0.5\left\|A - PA'P^\top\right\|_{1,1} + \|\eta_G - P\eta_{G'}\|_1$. In such a scenario, $\mathrm{GED}(G, G')$ is symmetric, *i.e.*, $\mathrm{GED}(G', G) = \mathrm{GED}(G, G')$ and it becomes zero only when G and G' are isomorphic.

Subgraph isomorphism: Assume $b^\ominus = b^\oplus = 0$. Then, if we set the cost of edge addition to be arbitrarily small as compared to the cost of edge deletion, *i.e.*, $a^\oplus \ll a^\ominus$. This yields $\mathrm{GED}(G, G') = \min_P(b^\ominus \sum_{u,v} \mathrm{ReLU}\left(A - PA'P^\top\right)[u,v])$, which can be reduced to zero for some permutation P, $G \subseteq G'$.

Maximum common edge subgraph: From Appendix D.2, we can write that $\mathrm{GED}(G, G') = \min_P 0.5(a^\oplus + a^\ominus)\left\|\max(A, PA'P^\top)\right\|_{1,1} + (b^\oplus + b^\ominus)\|\eta_G, P\eta_{G'}\|_1 - a^\ominus|E'| - a^\oplus|E| - b^\ominus|V'| - b^\oplus|V|$. When $a^\ominus = a^\oplus = 1$ and $b^\oplus = b^\ominus = 0$, then $\mathrm{GED}(G, G') = \left\|\max(A, PA'P^\top)\right\|_{1,1} = |E| + |E'| - \left\|\min(A, PA'P^\top)\right\|_{1,1}$. Here, $\min(A, PA'P^\top)$ characterizes maximum common edge subgraph and $\left\|\min(A, PA'P^\top)\right\|_{1,1}$ provides the number of edges of it.

D.4 Relation between AlignDiff and DiffAlign

Lemma 2 *Let $Z, Z' \in \mathbb{R}^{N\times M}$, and $S \in \mathbb{R}_{\ge 0}^{N\times N}$ be double stochastic. Then,*

$$\|\mathrm{ReLU}\left(Z - SZ'\right)\|_{1,1} \le \sum_{i,j} \|\mathrm{ReLU}\left(Z[i,:] - Z'[j,:]\right)\|_1 S[i,j]$$

Proof: We can write,

$$\begin{aligned}
\|\mathrm{ReLU}\left(Z - SZ'\right)\|_{1,1} &= \sum_{i,j} \left|\mathrm{ReLU}\left(Z[i,j] - \textstyle\sum_k S[i,k]Z'[k,j]\right)\right| \\
&\overset{(*)}{=} \sum_{i,j} \mathrm{ReLU}\left(\textstyle\sum_k S[i,k]Z[i,j] - S[i,k]Z'[k,j]\right) \\
&\overset{(**)}{\leq} \sum_{i,j}\sum_k S[i,k]\mathrm{ReLU}\left(Z[i,j] - Z'[k,j]\right) \\
&= \sum_{i,k} \|\mathrm{ReLU}\left(Z[i,:] - Z'[k,:]\right)\|_1 S[i,k] \qquad \square
\end{aligned}$$

where $(*)$ follows since $\sum_k S[i,k] = 1\,\forall i \in [N]$, and $(**)$ follows due to convexity of $\mathrm{ReLU}\,()$. Now, notice that when $S \in \mathbb{P}_N$, then $S[i,:]$ is 1 at one element while 0 at the rest. In that case, we have

$$\begin{aligned}
\sum_{i,j} \mathrm{ReLU}\left(\textstyle\sum_k S[i,k]Z[i,j] - S[i,k]Z'[k,j]\right) &= \sum_{i,j} \mathrm{ReLU}\left(Z[i,j] - Z'[k_i^*,j]\right) \\
&= \sum_{i,j}\sum_k S[i,k]\mathrm{ReLU}\left(Z[i,j] - Z'[k,j]\right)
\end{aligned}$$

where k_i^* is the index where $S[i,:]$ is 1. Hence, we have an equality when S is a hard permutation. Replacing (Z, Z') with (R, R') and (X, X'), we get that AlignDiff and DiffAlign are equivalent when S is a hard permutation matrix, and moreover DiffAlign is an upper bound on AlignDiff when S is a soft permutation matrix.

D.5 Proof that our design ensures $P' = P^\top$

Here we show why it is necessary to have a symmetric form for $C[u,u']$ in $\textsc{PermNet}_\phi$.
For GED(G, G'),

$$C[u,v] = \|c_\phi\left(x_K(u)\right) - c_\phi\left(x'_K(v)\right)\|_1$$

For GED(G', G),

$$C'[v,u] = \|c_\phi\left(x'_K(v)\right) - c_\phi\left(x_K(u)\right)\|_1$$

Because the Sinkhorn cost $C[u,v]$ is symmetric, using the above equations we can infer,

$$C[u,v] = C'[v,u]$$
$$C' = C^\top$$

This further leads to $P' = P^\top$.
If we use an asymmetric Sinkhorn cost (e.g. $C[u,v] = \|\mathrm{ReLU}\left(c_\phi\left(x_K(u)\right) - c_\phi\left(x'_K(v)\right)\right)\|_1$), we cannot ensure $C[u,v] = C'[v,u]$, which fails to satisfy $P = P^\top$.

D.6 Alternative surrogate for GED

From Appendix D.2, we have

$$\begin{aligned}
\mathrm{GED}(G, G') = \min_{P \in \mathbb{P}_N} \frac{a^\oplus + a^\ominus}{2} \left\|\max(A, PA'P^\top)\right\|_{1,1} - a^\ominus|E'| - a^\oplus|E| \\
+ \frac{b^\oplus + b^\ominus}{2} \|\max(\eta_G, P\eta_{G'})\|_1 - b^\ominus|V'| - b^\oplus|V|
\end{aligned}$$

Following the relaxations done in Section 3.2, we propose an alternative neural surrogate by replacing $\left\|\max(A, PA'P^\top)\right\|_{1,1}$ by $\|\max(R, SR')\|_{1,1}$ and $\|\max(\eta_G, P\eta_{G'})\|_1$ by $\|\max(X, PX')\|_{1,1}$, which gives us the approximated GED parameterized by θ and ϕ as

$$\begin{aligned}
\mathrm{GED}_{\theta,\phi}(G, G') = \frac{a^\oplus + a^\ominus}{2} \|\max(R, SR')\|_{1,1} - a^\ominus|E'| - a^\oplus|E| \\
+ \frac{b^\oplus + b^\ominus}{2} \|\max(X, PX')\|_{1,1} - b^\ominus|V'| - b^\oplus|V|
\end{aligned} \tag{22}$$

We call this neural surrogate as MAX. We note that element-wise maximum over A and $PA'P^\top$, only allows non-edge to non-edge mapping attribute a value of zero. However, the neural surrogate described in Equation 22 fails to capture this, due to the presence of the soft alignment matrix S. To address this, we explicitly discard such pairs from MAX by applying an OR operator over the edge presence between concerned node pairs, derived from the adjacency matrices A and A' and populated in $\text{OR}(A, A') \in \mathbb{R}^{\binom{N}{2} \times \binom{N}{2}}$ given by $\text{OR}(A[u,v], A'[u',v'])$. Similarly, the indication of node presence can be given be given as $\text{OR}(\eta_G, \eta_{G'})[u,u'] = \text{OR}(\eta_G[u], \eta_{G'}[u'])$. Hence, we write

$$\begin{aligned}\text{GED}_{\theta,\phi}(G,G') = \frac{a^{\oplus} + a^{\ominus}}{2} \left\| \text{OR}(A, A') \odot \max(R, SR') \right\|_{1,1} - a^{\ominus}|E'| - a^{\oplus}|E| \\ + \frac{b^{\oplus} + b^{\ominus}}{2} \left\| \text{OR}(\eta_G, \eta_{G'}) \odot \max(X, PX') \right\|_{1,1} - b^{\ominus}|V'| - b^{\oplus}|V|\end{aligned} \tag{23}$$

We call this formulation as MAX-OR. We provide the comparison between MAX, MAX-OR, and our models in Appendix F.

E Details about experimental setup

E.1 Generation of datasets

We have evaluated the performance of our methods and baselines on seven real-world datasets: Mutagenicity (Mutag), Ogbg-Code2 (Code2), Ogbg-Molhiv (Molhiv), Ogbg-Molpcba (Molpcba), AIDS, Linux and Yeast. We split each dataset into training, validation, and test splits in ratio of 60:20:20. For each split $\mathcal{D}$, we construct $(|\mathcal{D}|(|\mathcal{D}|+1))/2$ source and target graph instance pairs as follows: $\mathcal{S} = \{(G_i, G_j) : G_i, G_j \in \mathcal{D} \land i \leq j\}$. We perform experiment in *four* GED regimes:

1. GED under equal cost functions, where $b^{\ominus} = b^{\oplus} = a^{\ominus} = a^{\oplus} = 1$ and substitution costs are 0
2. GED under unequal cost functions, where $b^{\ominus} = 3, b^{\oplus} = 1, a^{\ominus} = 2, a^{\oplus} = 1$ and substitution costs are 0
3. edge GED under unequal cost functions, where $b^{\ominus} = b^{\oplus} = 0$, $a^{\ominus} = 2, a^{\oplus} = 1$, and substitution costs are 0
4. GED with node substitution under equal cost functions, where $b^{\ominus} = b^{\oplus} = a^{\ominus} = a^{\oplus} = 1$, as well as the node substitution cost $b^{\sim} = 1$.

We emphasize that we generated clean datasets by filtering out isomorphic graphs from the original datasets before performing the training, validation, and test splits. This step is crucial to prevent isomorphism bias in the models, which can occur due to leakage between the training and testing splits, as highlighted by [26].

For each graph, we have limited the maximum number of nodes to twenty, except for Linux, where the limit is ten. Information about the datasets is summarized in Table 7. Mutag contains nitroaromatic compounds, with each node having labels representing atom types. Molhiv and Molpcba contain molecules with node features representing atomic number, chirality, and other atomic properties. Code2 contains abstract syntax trees generated from Python codes. AIDS contains graphs of chemical compounds, with node types representing different atoms. For Molhiv, Molpcba and Linux, we have randomly sampled 1,000 graphs from each original dataset.

	#Graphs	# Train Pairs	# Val Pairs	# Test Pairs	Avg. $|V|$	Avg. $|E|$	Avg. GED equal cost	Avg. GED unequal cost
Mutag	729	95703	10585	10878	16.01	31.51	11.15	18.57
Code2	128	2926	325	378	18.77	35.53	10.02	16.43
Molhiv	1000	180300	20100	20100	15.01	31.3	11.77	19.86
Molpcba	1000	180300	20100	20100	17.52	37.35	9.58	15.73
AIDS	911	149331	16653	16836	10.97	21.94	7.38	12.07
Yeast	1000	180300	20100	20100	16.59	34.07	10.65	17.74
Linux	89	1431	153	190	8.71	16.7	4.91	7.94

Table 7: Salient characteristics of data sets.

E.2 Details about state-of-the-art baselines

We compared our model against nine state-of-the-art neural baselines and three combinatorial GED baselines. Below, we provide details of the methodology and hyperparameter settings used for each baseline. We ensured that the number of model parameters were in a comparable range. Specifically, we set the number of GNN layers to 5, each with a node embedding dimension of 10, to ensure consistency and comparability with our model. The following hyperparameters are used for training: Adam optimiser with a learning rate of 0.001 and weight decay of 0.0005, batch size of 256, early stopping with patience of 100 epochs, and Sinkhorn temperature set to 0.01.

Neural Baselines:

- **GMN-Match and GMN-Embed** Graph Matching Networks (GMN) use Euclidean distance to assess the similarity between graph-level embeddings of each graph. GMN is available in two variants: GMN-Embed, a late interaction model, and GMN-Match, an early interaction model. For this study, we used the official implementation of GMN to compute Graph Edit Distance (GED).[2]
- **ISONET** ISONET utilizes the Gumbel-Sinkhorn operator to learn asymmetric edge alignments between two graphs for subgraph matching. In our study, we extend ISONET's approach to predict

[2] https://github.com/Lin-Yijie/Graph-Matching-Networks/tree/main

the Graph Edit Distance (GED) score. We utilized the official PyTorch implementation provided by the authors for our experiments.[3]

- **GREED** GREED utilizes a siamese network architecture to compute graph-level embeddings in parallel for two graphs. It calculates the Graph Edit Distance (GED) score by computing the norm of the difference between these embeddings. The official implementation provided by the authors was used for our experiments.[4]
- **ERIC** ERIC utilizes a regularizer to learn node alignment, eliminating the need for an explicit node alignment module. The similarity score is computed using a Neural Tensor Network (NTN) and a Multi-Layer Perceptron (MLP) applied to the final graph-level embeddings of both graphs. These embeddings are derived by concatenating graph-level embeddings from each layer of a Graph Isomorphism Network (GIN). The model is trained using a combined loss from the regularizer and the predicted similarity score. For our experiments, we used the official PyTorch implementation to compute the Graph Edit Distance (GED). The GED scores were inverse normalized from the model output to predict the absolute GED.[5]
- **SimGNN** SimGNN leverages both graph-level and node-level embeddings at each layer of the GNN. The graph-level embeddings are processed through a Neural Tensor Network to obtain a pair-level embedding. Concurrently, the node-level embeddings are used to compute a pairwise similarity matrix between nodes, which is then converted into a histogram feature vector. A similarity score is calculated by passing the concatenation of these embeddings through a Multi-Layer Perceptron (MLP). We used the official PyTorch implementation of SimGNN and inverse normalization of the predicted Graph Edit Distance (GED) score to obtain the absolute GED value.[6]
- **H2MN** H2MN presents an early interaction model for graph similarity tasks. Instead of learning pairwise node relations, this method attempts to find higher-order node similarity using hypergraphs. At each time step of the hypergraph convolution, a subgraph matching module is employed to learn cross-graph similarity. After the convolution layers, a readout function is utilized to obtain graph-level embeddings. These embeddings are then concatenated and passed through a Multi-Layer Perceptron (MLP) to compute the similarity score. We used the official PyTorch implementation of H2MN.[7]
- **GraphSim** GraphSim uses GNN, where at each layer, a node-to-node similarity matrix is computed using the node embeddings. These similarity matrices are then processed using Convolutional Neural Networks (CNNs) and Multi-Layer Perceptrons (MLPs) to calculate a similarity score. We utilized the official PyTorch implementation.[8]
- **EGSC** We used the Teacher model proposed by Efficient Graph Similarity Computation (EGSC), which leverages an Embedding Fusion Network (EFN) at each layer of the Graph Isomorphism Network (GIN). The EFN generates a single embedding from a pair of graph embeddings. The embeddings of the graph pair from each layer are concatenated and subsequently passed through an additional EFN layer and a Multi-Layer Perceptron (MLP) to obtain the similarity score. To predict the absolute Graph Edit Distance (GED), we inversely normalized the GED score obtained from the output of EGSC. We utilized the official PyTorch implementation provided by the authors for our experiments. [9]

Combinatorial Baselines: We use the GEDLIB[10] library for implementation of all combinatorial baselines.

- **Bipartite** [41] Bipartite is an approximate algorithm that considers nodes and surrounding edges of nodes into account try to make a bipartite matching between two graphs. They use linear assignment algorithms to match nodes and their surroundings in two graphs.
- **Branch [8], Branch Tight [8]** improve upon [41] by decomposing graphs into branches. Branch Tight algorithm is another version of Branch that calculates a tighter lower bound but has a higher time complexity than Branch.

[3]https://github.com/Indradyumna/ISONET
[4]https://github.com/idea-iitd/greed
[5]https://github.com/JhuoW/ERIC
[6]https://github.com/benedekrozemberczki/SimGNN
[7]https://github.com/cszhangzhen/H2MN
[8]https://github.com/yunshengb/GraphSim
[9]https://github.com/canqin001/Efficient_Graph_Similarity_Computation
[10]https://github.com/dbblumenthal/gedlib

- **Anchor Aware GED** Chang et al. [15] provides an approximation algorithm that calculates a tighter lower bound using the anchor aware technique.
- **IPFP** [11] is an approximation algorithm which handles node and edge mapping simultaneously unlike previously discussed methods. This solves a quadratic assignment problem on edges and nodes.
- **F2** [29] uses a binary linear programming approach to find a higher lower bound on GED calculation. This method was used with a very high time limit to generate Ground truth for our experiments.

E.3 Details about GraphEdX

At the high level, GraphEdX consists of two components Embed_θ and PermNet_ϕ.

Neural Parameterization of Embed_θ: Embed_θ consists of two modules: a GNN denoted as MPNN_θ and a MLP_θ. The MPNN_θ consists of $K = 5$ propagation layers used to compute node embeddings of dimension $d = 10$. At each layer k, we compute the updated the node embedding as follows:

$$x_{k+1}(u) = \text{Update}_\theta\left(x_k(u), \sum_{v\in\text{nbr}(u)} \text{LRL}_\theta(x_k(u), x_k(v))\right) \tag{24}$$

where LRL_θ is a Linear-ReLU-Linear network, with $d = 10$ features, and the Update_θ network consists of a Gated Recurrent Unit [30]. In case of GED setting under equal cost and GED setting under unequal cost, we set the initial node features $x_0(u) = 1$, following [30]. However, in case of computation of GED with node substitution costs, we explicitly provide the one-hot labels as node features. Given the node embeddings and edge-presence indicator obtained from the adjacency matrices, after 5 layer propogations, we compute the edge embeddings $r(e)$ using MLP_θ, which is decoupled from MPNN_θ. MLP_θ consists of a Linear-ReLU-Linear network that maps the $2d + 1 = 21$ dimensional input consisting of forward $(x_K(u) \,||\, x_K(v) \,||\, A[u, v])$ and backward $(x_K(v) \,||\, x_K(u) \,||\, A[v, u])$ signals to $D = 20$ dimensions.

Neural Parameterization of PermNet_ϕ: Given the node embeddings $x_K(\cdot)$ and $x'_K(\cdot)$, we first pass them through a neural network c_ϕ which consists of a Linear-ReLU-Linear network transforming the features from $d = 10$ to N dimensions, which is the number of nodes after padding. Except for Linux where $N = 10$, all other datasets have $N = 20$. We obtain the matrix C such that $C[u, u'] = \|c_\phi(x_K(u)) - c_\phi(x'_K(u'))\|_1$. Using temperature $\tau = 0.01$, we perform Sinkhorn iterations on $\exp(-C/\tau)$ as follows for $T = 20$ iterations to get P:

$$P_k = \text{NormCol}\left(\text{NormRow}\left(P_{k-1}\right)\right)$$

where $P_0 = \exp(-C/\tau)$. Here $\text{NormRow}(M)[i, j] = M[i, j]/\sum_\ell M[\ell, j]$ denotes the row normalization function and $\text{NormCol}(M)[i, j] = M[i, j]/\sum_\ell M[i, \ell]$ denotes the column normalization function. We note that the soft alignment P obtained does not depend on the GED cost values, as discussed in Appendix D. The soft alignment P for nodes is used to construct soft alignment S for as follows: $S[(u, v), (u', v')] = P[u, u']P[v, v'] + P[u, v']P[v, u']$.

E.4 Evaluation metrics

Given the dataset $\mathcal{S}$ consisting of input pairs of graphs (G, G') along with the ground truth $\text{GED}(G, G')$ and model prediction $\widehat{\text{GED}}(G, G')$, we evaluate the performance of the model using the Root Mean Square Error (RMSE) and Kendall-Tau (KTau) [28] between the predicted GED scores and actual GED values.

- **MSE:** It evaluates how far the predicted GED values are from the ground truth. A better performing model is indicated by a lower MSE value.

$$\text{MSE} = \frac{1}{|\mathcal{S}|}\sum_{(G,G')\in\mathcal{S}} \left(\text{GED}(G, G') - \widehat{\text{GED}}(G, G')\right)^2 \tag{25}$$

- **KTau:** Selection of relevant corpus graphs via graph similarity scoring is crucial to graph retrieval setups. In this context, we would like the number of concordant pairs N_+ (where the ranking of ground truth GED and model prediction agree) to be high, and the discordant pairs N_-(where the two disagree) to be low. Formally, we write

$$\text{KTau} = \frac{N_+ - N_-}{\binom{|\mathcal{S}|}{2}} \tag{26}$$

For the methods which compute a similarity score between the pair of graphs through the notion of normalized GED, we map the similarity score s back to the GED as $\widehat{\text{GED}}(G, G') = -\frac{|V|+|V|'}{2} \log(s+\epsilon)$ where $\epsilon = 10^{-7}$ is added for stability of the logarithm.

E.5 Hardware and license

We implement our models using Python 3.11.2 and PyTorch 2.0.0. The training of our models and the baselines was performed across servers containing Intel Xeon Silver 4216 2.10GHz CPUs, and Nvidia RTX A6000 GPUs. Running times of all methods are compared on the same GPU.

F Additional experiments

In this section, we present results from various additional experiments performed to measure the performance of our model under different cost settings.

F.1 Comparison of GraphEdX with baselines on equal and unequal cost setting

Tables 8 and 9 report performance in terms of MSE under equal and unequal cost settings, respectively. Table 10 reports performance in terms of KTau under both equal and unequal cost settings. The results are similar to those in Table 2, where our model is the clear winner across all datasets, outperforming the second-best performer by a significant margin. There is no consistent second-best model, but ERIC, EGSC, and ISONET perform comparably and better than the others.

	Mutag	Code2	Molhiv	Molpcba	AIDS	Linux	Yeast
GMN-Match	0.797 ± 0.013	1.677 ± 0.187	1.318 ± 0.020	1.073 ± 0.011	0.821 ± 0.010	0.687 ± 0.088	1.175 ± 0.013
GMN-Embed	1.032 ± 0.016	1.358 ± 0.104	1.859 ± 0.020	1.951 ± 0.020	1.044 ± 0.013	0.736 ± 0.102	1.767 ± 0.021
ISONET	1.187 ± 0.021	0.879 ± 0.061	1.354 ± 0.015	1.106 ± 0.011	1.640 ± 0.020	1.185 ± 0.115	1.578 ± 0.019
GREED	1.398 ± 0.033	1.869 ± 0.140	1.708 ± 0.019	1.550 ± 0.017	1.004 ± 0.012	1.331 ± 0.169	1.423 ± 0.015
ERIC	0.719 ± 0.011	1.363 ± 0.110	1.165 ± 0.018	0.862 ± 0.009	0.731 ± 0.008	1.664 ± 0.260	0.969 ± 0.010
SimGNN	1.471 ± 0.024	2.667 ± 0.215	1.609 ± 0.020	1.456 ± 0.020	1.455 ± 0.020	7.232 ± 0.762	1.999 ± 0.043
H2MN	1.278 ± 0.021	7.240 ± 0.527	1.521 ± 0.020	1.402 ± 0.020	1.114 ± 0.015	2.238 ± 0.247	1.353 ± 0.018
GraphSim	2.005 ± 0.031	3.139 ± 0.206	2.577 ± 0.064	1.656 ± 0.023	1.936 ± 0.026	2.900 ± 0.318	2.232 ± 0.030
EGSC	0.765 ± 0.011	4.165 ± 0.285	1.138 ± 0.016	0.938 ± 0.010	0.627 ± 0.007	2.411 ± 0.325	0.950 ± 0.010
GraphEdX	0.492 ± 0.007	0.429 ± 0.036	0.781 ± 0.008	0.764 ± 0.007	0.565 ± 0.006	0.354 ± 0.043	0.717 ± 0.007

Table 8: Comparison with baselines in terms of MSE including standard error for equal cost setting ($b^{\ominus} = b^{\oplus} = a^{\ominus} = a^{\oplus} = 1$). Green (yellow) numbers report the best (second best) performers.

	Mutag	Code2	Molhiv	Molpcba	AIDS	Linux	Yeast
GMN-Match	69.210 ± 0.883	13.472 ± 0.970	76.923 ± 0.862	23.985 ± 0.224	31.522 ± 0.513	21.519 ± 2.256	63.179 ± 1.127
GMN-Embed	72.495 ± 0.915	13.425 ± 1.035	78.254 ± 0.865	28.437 ± 0.268	33.221 ± 0.523	20.591 ± 2.136	60.949 ± 0.663
ISONET	3.369 ± 0.062	3.025 ± 0.206	3.451 ± 0.039	2.781 ± 0.029	5.513 ± 0.092	3.031 ± 0.299	4.555 ± 0.061
GREED	68.732 ± 0.867	11.095 ± 0.773	78.300 ± 0.795	26.057 ± 0.238	34.354 ± 0.557	20.667 ± 2.140	60.652 ± 0.704
ERIC	1.981 ± 0.032	12.767 ± 1.177	3.377 ± 0.070	2.057 ± 0.020	1.581 ± 0.017	7.809 ± 0.911	2.341 ± 0.030
SimGNN	4.747 ± 0.079	5.212 ± 0.360	4.145 ± 0.051	3.465 ± 0.047	4.316 ± 0.071	5.369 ± 0.546	4.496 ± 0.060
H2MN	3.413 ± 0.053	9.435 ± 0.728	3.782 ± 0.046	3.396 ± 0.046	3.105 ± 0.043	5.848 ± 0.611	3.678 ± 0.046
GraphSim	5.370 ± 0.092	7.405 ± 0.577	6.643 ± 0.181	3.928 ± 0.053	5.266 ± 0.081	6.815 ± 0.628	6.907 ± 0.137
EGSC	1.758 ± 0.026	3.957 ± 0.365	2.371 ± 0.025	2.133 ± 0.022	1.693 ± 0.023	5.503 ± 0.496	2.157 ± 0.027
GraphEdX	1.134 ± 0.016	1.478 ± 0.118	1.804 ± 0.019	1.677 ± 0.016	1.252 ± 0.014	0.914 ± 0.110	1.603 ± 0.016

Table 9: Comparison with baselines in terms of MSE including standard error for unequal cost setting ($b^{\ominus} = 3, b^{\oplus} = 1, a^{\ominus} = 2, a^{\oplus} = 1$). Green (yellow) numbers report the best (second best) performers.

	GED with equal cost							GED with unequal cost						
	Mutag	Code2	Molhiv	Molpcba	AIDS	Linux	Yeast	Mutag	Code2	Molhiv	Molpcba	AIDS	Linux	Yeast
GMN-Match	0.901	0.876	0.887	0.797	0.824	0.826	0.852	0.606	0.781	0.619	0.596	0.611	0.438	0.610
GMN-Embed	0.887	0.892	0.856	0.723	0.796	0.815	0.815	0.603	0.790	0.607	0.534	0.601	0.531	0.573
ISONET	0.885	0.918	0.878	0.793	0.756	0.786	0.827	0.887	0.908	0.875	0.817	0.755	0.776	0.834
GREED	0.873	0.878	0.859	0.757	0.807	0.756	0.832	0.614	0.812	0.598	0.547	0.596	0.522	0.582
ERIC	0.909	0.892	0.897	0.820	0.837	0.736	0.868	0.620	0.804	0.895	0.841	0.855	0.633	0.886
SimGNN	0.871	0.856	0.877	0.776	0.775	0.377	0.834	0.862	0.874	0.872	0.804	0.768	0.731	0.843
H2MN	0.878	0.711	0.879	0.781	0.794	0.664	0.848	0.873	0.813	0.875	0.804	0.792	0.681	0.851
GraphSim	0.847	0.839	0.856	0.756	0.730	0.601	0.810	0.851	0.844	0.851	0.784	0.744	0.656	0.824
EGSC	0.906	0.815	0.896	0.809	0.850	0.664	0.868	0.912	0.894	0.900	0.836	0.858	0.696	0.884
GraphEdX	0.926	0.937	0.910	0.831	0.857	0.882	0.886	0.929	0.932	0.912	0.858	0.871	0.875	0.898

Table 10: Comparison with baselines in terms of KTau for both equal and unequal cost settings, where for equal cost settings costs are $b^{\ominus} = b^{\oplus} = a^{\ominus} = a^{\oplus} = 1$ and for unequal cost settings costs are $b^{\ominus} = 3, b^{\oplus} = 1, a^{\ominus} = 2, a^{\oplus} = 1$. Green (yellow) numbers report the best (second best) performers.

F.2 Comparison of GraphEdX with baselines with node substitution cost

In Tables 11 and 12, we compare the performance of GraphEdX with baselines under a node substitution cost $b^{\sim}$. The cost setting is $b^{\ominus} = b^{\oplus} = b^{\sim} = a^{\ominus} = a^{\oplus} = 1$. This experiment includes only five datasets where node labels are present. We observe that GraphEdX outperforms all other baselines. There is no clear second-best model, but ERIC, EGSC, and ISONET perform better than the others.

	Mutag	Code2	Molhiv	Molpcba	AIDS
GMN-Match	1.057 ± 0.011	5.224 ± 0.404	1.388 ± 0.018	1.432 ± 0.017	0.868 ± 0.007
GMN-Embed	2.159 ± 0.026	4.070 ± 0.318	3.523 ± 0.040	4.657 ± 0.054	1.818 ± 0.014
ISONET	0.876 ± 0.008	1.129 ± 0.084	1.617 ± 0.020	1.332 ± 0.014	1.142 ± 0.010
GREED	2.876 ± 0.032	4.983 ± 0.531	2.923 ± 0.033	3.902 ± 0.044	2.175 ± 0.016
ERIC	0.886 ± 0.009	6.323 ± 0.683	1.537 ± 0.018	1.278 ± 0.014	1.602 ± 0.036
SimGNN	1.160 ± 0.013	5.909 ± 0.490	1.888 ± 0.031	2.172 ± 0.050	1.418 ± 0.020
H2MN	1.277 ± 0.014	6.783 ± 0.587	1.891 ± 0.024	1.666 ± 0.021	1.290 ± 0.011
GraphSim	1.043 ± 0.010	4.708 ± 0.425	1.817 ± 0.021	1.748 ± 0.021	1.561 ± 0.021
EGSC	0.776 ± 0.008	8.742 ± 0.831	1.273 ± 0.016	1.426 ± 0.018	1.270 ± 0.028
GraphEdX	0.441 ± 0.004	0.820 ± 0.092	0.792 ± 0.009	0.846 ± 0.009	0.538 ± 0.003

Table 11: Comparison with baselines in terms of MSE including standard error, in presence of the node substitution cost, which set to one in equal cost setting: $b^{\ominus} = b^{\oplus} = b^{\sim} = a^{\ominus} = a^{\oplus} = 1$. Green (yellow) numbers report the best (second best) performers.

	Mutag	Code2	Molhiv	Molpcba	AIDS
GMN-Match	0.895	0.811	0.881	0.809	0.839
GMN-Embed	0.847	0.845	0.796	0.684	0.767
ISONET	0.906	0.925	0.868	0.815	0.812
GREED	0.827	0.829	0.822	0.710	0.746
ERIC	0.905	0.847	0.872	0.818	0.815
SimGNN	0.891	0.836	0.864	0.797	0.810
H2MN	0.886	0.818	0.858	0.789	0.802
GraphSim	0.896	0.846	0.860	0.782	0.795
EGSC	0.912	0.802	0.885	0.821	0.832
GraphEdX	0.936	0.945	0.913	0.856	0.874

Table 12: Comparison with baselines in terms of KTau, in presence of the node substitution cost, which set to one in equal cost setting: $b^{\ominus} = b^{\oplus} = b^{\sim} = a^{\ominus} = a^{\oplus} = 1$. Green (yellow) numbers report the best (second best) performers.

F.3 Performance evaluation for edge-only vs. all-node-pair representations

Tables 13 and 14 contain extended results from Table 4 across seven datasets. The results are similar to those discussed in the main paper: (1) The all-node-pair representation performs better than the variants of edge-only representations. (2) within the edge-only representation, Edge-only (edge $\rightarrow$ edge) performs better than Edge-only (pair $\rightarrow$ pair) in most of the cases.

	Mutag	Code2	Molhiv	Molpcba	AIDS	Linux	Yeast
Edge-only (edge $\rightarrow$ edge)	0.566 ± 0.008	0.683 ± 0.051	0.858 ± 0.009	0.791 ± 0.008	0.598 ± 0.006	0.454 ± 0.063	0.749 ± 0.007
Edge-only (pair $\rightarrow$ pair)	0.596 ± 0.008	0.760 ± 0.058	0.862 ± 0.009	0.811 ± 0.008	0.606 ± 0.006	0.474 ± 0.056	0.761 ± 0.008
GraphEdX	0.492 ± 0.007	0.429 ± 0.036	0.781 ± 0.008	0.764 ± 0.007	0.565 ± 0.006	0.354 ± 0.043	0.717 ± 0.007

Table 13: Comparison of using all-node-pairs against edge-only representations using MSE for equal cost setting. Green (yellow) numbers report the best (second best) performers.

	Mutag	Code2	Molhiv	Molpcba	AIDS	Linux	Yeast
Edge-only (edge $\rightarrow$ edge)	1.274 ± 0.017	1.817 ± 0.141	1.847 ± 0.019	1.793 ± 0.017	1.318 ± 0.014	0.907 ± 0.129	1.649 ± 0.016
Edge-only (pair $\rightarrow$ pair)	1.276 ± 0.017	1.879 ± 0.136	1.865 ± 0.020	1.779 ± 0.017	1.422 ± 0.015	0.992 ± 0.114	1.694 ± 0.017
GraphEdX	1.134 ± 0.016	1.478 ± 0.118	1.804 ± 0.019	1.677 ± 0.016	1.252 ± 0.014	0.914 ± 0.110	1.603 ± 0.016

Table 14: Comparison of using all-node-pairs against edge-only representations using MSE for unequal cost setting. Green (yellow) numbers report the best (second best) performers.

F.4 Effect of using cost-guided scoring function on baselines

In Tables 15 and 16, we report the impact of replacing the baselines' scoring function with our proposed cost-guided scoring function on three baselines across seven datasets for equal and unequal cost settings, respectively. We notice that similar to the results reported in Section 4.2, the cost-guided scoring function helps the baselines perform significantly better in both the cost settings.

	Mutag	Code2	Molhiv	Molpcba	AIDS	Linux	Yeast
GMN-Match	0.797 ± 0.013	1.677 ± 0.187	1.318 ± 0.020	1.073 ± 0.011	0.821 ± 0.010	0.687 ± 0.088	1.175 ± 0.013
GMN-Match *	**0.654 ± 0.011**	**0.960 ± 0.092**	**1.008 ± 0.011**	**0.858 ± 0.009**	**0.601 ± 0.007**	**0.590 ± 0.084**	**0.849 ± 0.009**
GMN-Embed	1.032 ± 0.016	1.358 ± 0.104	1.859 ± 0.020	1.951 ± 0.020	1.044 ± 0.013	0.736 ± 0.102	1.767 ± 0.021
GMN-Embed *	**1.011 ± 0.017**	**1.179 ± 0.098**	**1.409 ± 0.015**	**1.881 ± 0.019**	**0.849 ± 0.010**	**0.577 ± 0.094**	**1.600 ± 0.017**
GREED	**1.398 ± 0.033**	1.869 ± 0.140	1.708 ± 0.019	**1.550 ± 0.017**	**1.004 ± 0.012**	1.331 ± 0.169	**1.423 ± 0.015**
GREED *	2.133 ± 0.037	**1.850 ± 0.156**	**1.644 ± 0.019**	1.623 ± 0.017	1.143 ± 0.015	**1.297 ± 0.151**	1.440 ± 0.016
GraphEdX	0.492 ± 0.007	0.429 ± 0.036	0.781 ± 0.008	0.764 ± 0.007	0.565 ± 0.006	0.354 ± 0.043	0.717 ± 0.007

Table 15: Impact of cost-guided distance on MSE in equal cost setting ($b^{\ominus} = b^{\oplus} = a^{\ominus} = a^{\oplus} = 1$). * represents the variant of the baseline with cost-guided distance. Green shows the best performing model. **Bold** font indicates the best variant of the baseline.

	Mutag	Code2	Molhiv	Molpcba	AIDS	Linux	Yeast
GMN-Match	69.210 ± 0.883	13.472 ± 0.970	76.923 ± 0.862	23.985 ± 0.224	31.522 ± 0.513	21.519 ± 2.256	63.179 ± 1.127
GMN-Match *	**1.592 ± 0.027**	**2.906 ± 0.285**	**2.162 ± 0.024**	**1.986 ± 0.021**	**1.434 ± 0.017**	**1.596 ± 0.211**	**2.036 ± 0.022**
GMN-Embed	72.495 ± 0.915	13.425 ± 1.035	78.254 ± 0.865	28.437 ± 0.268	33.221 ± 0.523	20.591 ± 2.136	60.949 ± 0.663
GMN-Embed *	**2.368 ± 0.039**	**3.272 ± 0.289**	**3.413 ± 0.037**	**4.286 ± 0.043**	**2.046 ± 0.025**	**1.495 ± 0.200**	**3.850 ± 0.042**
GREED	68.732 ± 0.867	11.095 ± 0.773	78.300 ± 0.795	26.057 ± 0.238	34.354 ± 0.557	20.667 ± 2.140	60.652 ± 0.704
GREED *	**2.456 ± 0.040**	**5.429 ± 0.517**	**3.827 ± 0.043**	**3.807 ± 0.040**	**2.282 ± 0.028**	**2.894 ± 0.394**	**3.506 ± 0.038**
GraphEdX	1.134 ± 0.016	1.478 ± 0.118	1.804 ± 0.019	1.677 ± 0.016	1.252 ± 0.014	0.914 ± 0.110	1.603 ± 0.016

Table 16: Impact of cost-guided distance on MSE in unequal cost setting ($b^{\ominus} = 3, b^{\oplus} = 1, a^{\ominus} = 2, a^{\oplus} = 1$). * represents the variant of the baseline with cost-guided distance. Green shows the best performing model. **Bold** font indicates the best variant of the baseline.

F.5 Results on performance of the alternate surrogates for GED

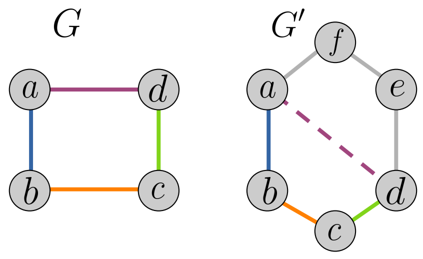
In Table 17, we present the performance of the alternate surrogates scoring function for GED discussed in D under unequal cost settings ($b^{\ominus} = 3, b^{\oplus} = 1, a^{\ominus} = 2, a^{\oplus} = 1$). From the results, we can infer that the alternate surrogates have comparable performance to GraphEdX however GraphEdX outperforms it by a small margin on six out of the seven datasets.

	Mutag	Code2	Molhiv	Molpcba	AIDS	Linux	Yeast
MAX-OR	1.194 ± 0.016	1.112 ± 0.084	1.987 ± 0.022	1.806 ± 0.017	1.347 ± 0.014	1.009 ± 0.132	1.686 ± 0.018
MAX	1.351 ± 0.018	1.772 ± 0.122	1.972 ± 0.021	1.764 ± 0.017	1.346 ± 0.015	1.435 ± 0.169	1.748 ± 0.018
GraphEdX	1.134 ± 0.016	1.478 ± 0.118	1.804 ± 0.019	1.677 ± 0.016	1.252 ± 0.014	0.914 ± 0.110	1.603 ± 0.016

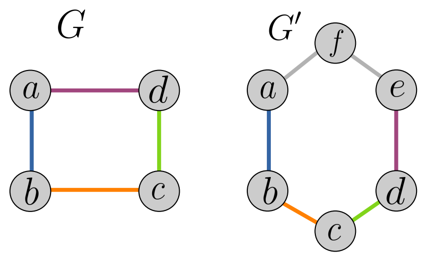
Table 17: Comparison of MSE between our variant MAX-OR and MAX. Green (yellow) numbers report the best (second best) performers.

F.6 Importance of node-edge consistency

GRAPHEDX enforces consistency between node and edge alignments by design. However, one might choose to enforce node-edge consistency through alignment regularization between independently learnt soft node and edge alignment. However, as shown in Figure 18, we notice that such non-constrained learning might lead to under-prediction or incorrect alignments. We demonstrate the importance of constraining the node-pair alignment S with the node alignment P by showing the mapping of nodes and edges between two graphs. The required edit operations for subfigure a) with the constrained S are two node additions $\{e, f\}$, one edge deletion (d, a), and three edge additions $\{(a, f), (e, d), (e, f)\}$. Assuming that each edit costs one, the true GED is 6. However, in subplot b), S is not constrained, and the edit operations with the lowest cost are two node additions $\{e, f\}$ and two edge additions $\{(a, f), (e, f)\}$. This erroneously results in a GED of 4.



(a) Constrained S



(b) Unconstrained S

Figure 18: Node and edge alignment with constrained and unconstrained alignment S. A dashed edge represents the deleted edge. Grey edges represent added edges.

Further, in Table 19, we compare the performance of enforcing node-edge consistency through design (GRAPHEDX), and through alignment regularization (REG). Following the discussion in Section 3.2, such a model also exhibits a variant with XOR, called REG-xor. We notice that GRAPHEDX even outperforms such the described model in 4 out of 6 cases. We also notice that REG-xor outperforms GRAPHEDX in the other two cases. However, the above example shows a tendency to learn wrong alignments which in turn gives wrong optimal edit paths.

	GED with equal cost			GED with unequal cost		
	Mutag	Code2	Molhiv	Mutag	Code2	Molhiv
REG	0.536	0.576	0.848	1.162	1.488	1.877
REG-xor	0.513	0.587	0.826	1.309	1.440	1.711
GRAPHEDX	0.492	0.429	0.781	1.134	1.478	1.804

Table 19: Comparison of alignment regularizer usage versus no alignment regularizer usage on equal cost GED, Measured by MSE. Green (yellow) numbers report the best (second best) performers.

F.7 Comparison of nine possible combinations our proposed set distances

In Tables 20 and 21, we compare the performance of nine possible combinations our proposed set distances for equal and unequal cost settings respectively. Results follow the observations in Table 5, where the variant with XOR-DiffAlign outperforms those without it.

Edge edit	Node edit	Mutag	Code2	Molhiv	Molpcba	AIDS	Linux	Yeast
DiffAlign	DiffAlign	0.579 ± 0.0078	0.740 ± 0.0585	0.820 ± 0.0086	0.778 ± 0.0075	0.603 ± 0.0063	0.494 ± 0.0528	0.728 ± 0.0071
DiffAlign	AlignDiff	0.557 ± 0.0073	0.742 ± 0.0612	0.806 ± 0.0088	0.779 ± 0.0076	0.597 ± 0.0063	0.452 ± 0.0614	0.747 ± 0.0078
DiffAlign	XOR	0.538 ± 0.0072	0.719 ± 0.0560	0.794 ± 0.0083	0.777 ± 0.0075	0.580 ± 0.0060	0.356 ± 0.0512	0.750 ± 0.0075
AlignDiff	DiffAlign	0.537 ± 0.0072	0.513 ± 0.0367	0.815 ± 0.0085	0.773 ± 0.0074	0.606 ± 0.0064	0.508 ± 0.0607	0.731 ± 0.0073
AlignDiff	AlignDiff	0.578 ± 0.0079	0.929 ± 0.0659	0.833 ± 0.0086	0.773 ± 0.0075	0.593 ± 0.0062	0.605 ± 0.0678	0.761 ± 0.0076
AlignDiff	XOR	0.533 ± 0.0074	0.826 ± 0.0565	0.812 ± 0.0083	0.780 ± 0.0074	0.575 ± 0.0060	0.507 ± 0.0568	0.889 ± 0.0138
XOR	AlignDiff	0.492 ± 0.0066	0.429 ± 0.0355	0.788 ± 0.0084	0.766 ± 0.0074	0.565 ± 0.0062	0.416 ± 0.0494	0.730 ± 0.0072
XOR	DiffAlign	0.510 ± 0.0067	0.634 ± 0.0522	0.781 ± 0.0084	0.765 ± 0.0073	0.574 ± 0.0060	0.332 ± 0.0430	0.717 ± 0.0072
XOR	XOR	0.530 ± 0.0074	1.588 ± 0.1299	0.807 ± 0.0084	0.764 ± 0.0073	0.564 ± 0.0059	0.354 ± 0.0427	0.721 ± 0.0076
GRAPHEDX		0.492 ± 0.0066	0.429 ± 0.0355	0.781 ± 0.0084	0.764 ± 0.0073	0.565 ± 0.0062	0.354 ± 0.0427	0.717 ± 0.0072

Table 20: Comparison of MSE for nine combinations of our neural set distance surrogates under equal cost settings. The GRAPHEDX model was selected based on the best MSE on the validation set, while the reported results represent MSE on the test set. Green (yellow) numbers report the best (second best) performers.

Edge edit	Node edit	Mutag	Code2	Molhiv	Molpcba	AIDS	Linux	Yeast
DiffAlign	DiffAlign	1.205 ± 0.0159	2.451 ± 0.2141	1.855 ± 0.0197	1.825 ± 0.0178	1.417 ± 0.0146	0.988 ± 0.1269	1.630 ± 0.0161
DiffAlign	AlignDiff	1.211 ± 0.0164	2.116 ± 0.1581	1.887 ± 0.0199	1.811 ± 0.0174	1.319 ± 0.0140	1.078 ± 0.1168	1.791 ± 0.0185
DiffAlign	XOR	1.146 ± 0.0154	1.896 ± 0.1487	1.802 ± 0.0188	1.822 ± 0.0176	1.381 ± 0.0148	1.049 ± 0.1182	1.737 ± 0.0172
AlignDiff	DiffAlign	1.185 ± 0.0159	1.689 ± 0.1210	1.874 ± 0.0202	1.758 ± 0.0169	1.391 ± 0.0145	0.914 ± 0.1099	1.643 ± 0.0163
AlignDiff	AlignDiff	1.338 ± 0.0178	1.488 ± 0.1222	1.903 ± 0.0204	1.859 ± 0.0179	1.326 ± 0.0141	1.258 ± 0.1335	1.731 ± 0.0171
AlignDiff	XOR	1.196 ± 0.0164	1.741 ± 0.1151	1.870 ± 0.0196	1.815 ± 0.0174	1.374 ± 0.0146	1.128 ± 0.1330	1.802 ± 0.0194
XOR	AlignDiff	1.134 ± 0.0158	1.478 ± 0.1178	1.872 ± 0.0202	1.742 ± 0.0168	1.252 ± 0.0136	1.073 ± 0.1211	1.639 ± 0.0162
XOR	DiffAlign	1.148 ± 0.0157	1.489 ± 0.1220	1.804 ± 0.0192	1.757 ± 0.0171	1.340 ± 0.0140	0.931 ± 0.1149	1.603 ± 0.0160
XOR	XOR	1.195 ± 0.0172	2.507 ± 0.1979	1.855 ± 0.0195	1.677 ± 0.0161	1.319 ± 0.0141	1.193 ± 0.1490	1.638 ± 0.0169
GRAPHEDX		1.134 ± 0.0158	1.478 ± 0.1178	1.804 ± 0.0192	1.677 ± 0.0161	1.252 ± 0.0136	0.914 ± 0.1099	1.603 ± 0.0160

Table 21: Comparison of MSE for nine combinations under unequal cost settings. The GRAPHEDX model was selected based on the best MSE on the validation set, while the reported results represent MSE on the test set. Green (yellow) numbers report the best (second best) performers.

F.8 Comparison of performance of GRAPHEDX on unequal cost Edge-GED

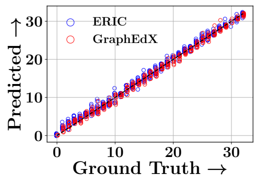
We consider another cost setting – where the node costs are explicitly set to 0, and $a^{\oplus} = 1, a^{\ominus} = 2$. In such a case, GRAPHEDX only consists of $\Delta^{\ominus}(R, R' \mid S)$ and $\Delta^{\oplus}(R, R' \mid S)$ terms. To showcase the importance of aligning edges through edge alignment, we generate an alternate model, where the alignment happens through the terms $\Delta^{\ominus}(X, X' \mid P)$ and $\Delta^{\oplus}(X, X' \mid P)$, where we set $b^{\oplus} = 1$ and $b^{\ominus} = 2$, and set the edge costs to 0. We call this model NodeSwap (w/o XOR), and the corresponding XOR variant as NodeSwap + XOR. In Table 22, we compare the performance variants of GRAPHEDX with NodeSwap (w/o XOR) and the rest of the baselines to predict the Edge GED score in an unequal cost setting. From the results, we can infer that the performance of edge-alignment based model to predict Edge-GED outperforms the corresponding node-alignment version.

	MSE ± STD			KTau		
	Mutag	Molhiv	Linux	Mutag	Molhiv	Linux
GMN-Match	11.276 ± 0.143	13.586 ± 0.171	4.893 ± 0.527	0.600	0.562	0.453
GMN-Embed	13.627 ± 0.179	16.482 ± 0.188	4.363 ± 0.420	0.556	0.529	0.484
ISONET	1.468 ± 0.020	2.142 ± 0.023	1.930 ± 0.186	0.846	0.802	0.659
GREED	11.906 ± 0.148	13.723 ± 0.136	3.847 ± 0.397	0.588	0.558	0.512
ERIC	1.900 ± 0.028	2.154 ± 0.024	3.361 ± 0.353	0.823	0.805	0.510
SimGNN	3.138 ± 0.052	3.771 ± 0.046	5.089 ± 0.524	0.784	0.736	0.410
H2MN	3.771 ± 0.062	3.735 ± 0.047	5.443 ± 0.566	0.748	0.741	0.358
GraphSim	4.696 ± 0.076	5.200 ± 0.074	6.597 ± 0.697	0.720	0.694	0.316
EGSC	1.871 ± 0.028	2.187 ± 0.025	2.803 ± 0.260	0.823	0.797	0.608
NodeSwap (w/o XOR)	1.246 ± 0.017	1.858 ± 0.019	0.997 ± 0.124	0.857	0.814	0.757
NodeSwap + XOR	11.984 ± 0.227	11.158 ± 0.196	10.959 ± 1.116	0.586	0.604	0.321
GRAPHEDX (w/o XOR)	1.174 ± 0.016	1.842 ± 0.019	0.976 ± 0.115	0.863	0.815	0.764
GRAPHEDX + XOR	1.125 ± 0.016	1.855 ± 0.020	0.922 ± 0.108	0.866	0.817	0.780

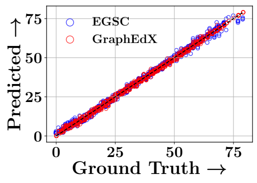
Table 22: Comparison of edge-alignment based GED scoring function with node-alignment based GED scoring function and state-of-the-art baselines under the cost setting: $a^{\ominus} = 2, a^{\oplus} = 1, b^{\ominus} = b^{\oplus} = 0$. In case of NodeSwap (w/o XOR), we swap the edge costs and node costs, and expect the model to learn the alignments in Edge GED through node alignment only. Green (yellow) numbers report the best (second best) performers.

F.9 Comparison of performance of our model with baselines using scatter plot

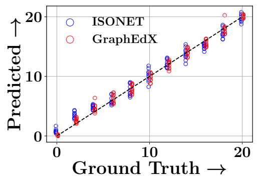
In Figure 23, we illustrate the performance of our model compared to the second-best performing model, under both equal and unequal cost settings, by visualizing the distribution of outputs of the predicted GEDs by both models. We observe that predictions from our model consistently align closer to the $y = x$ line across various datasets showcasing lower output variance as compared to the next best-performing model.



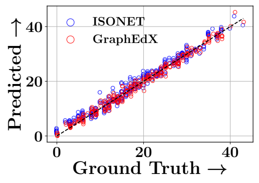
(a) Mutag equal cost



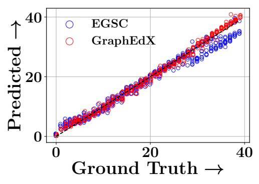
(b) Mutag unequal cost



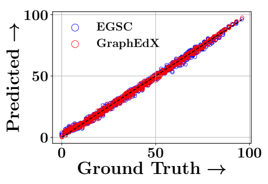
(c) Code2 equal cost



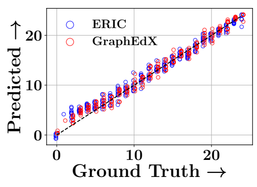
(d) Code2 unequal cost



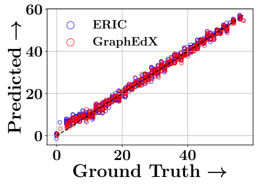
(e) Molhiv equal cost



(f) Molhiv unequal cost



(g) Molpcba equal cost

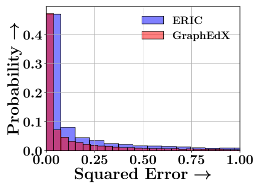


(h) Molpcba unequal cost

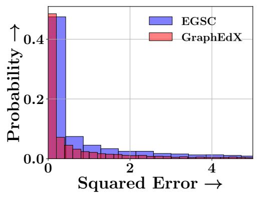
Figure 23: Scatter plot comparing the distribution of the predicted GED of our model with the next best-performing model across various datasets under both equal and unequal cost settings.

F.10 Comparison of performance of our model with baselines using error distribution

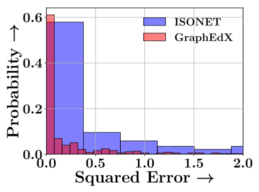
In Figure 24, we plot the distribution of error (MSE) of our model against the second-best performing model, under both equal and unequal cost settings. We observe that our model performs better, exhibiting a higher probability density for lower MSE values and a lower probability density for higher MSE values.



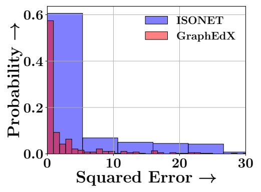
(a) Mutag equal cost



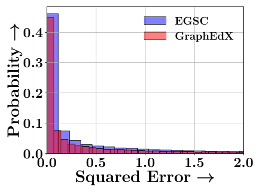
(b) Mutag unequal cost



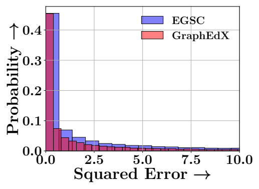
(c) Code2 equal cost



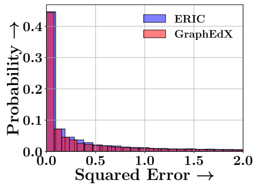
(d) Code2 unequal cost



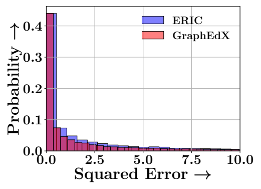
(e) Molhiv equal cost



(f) Molhiv unequal cost



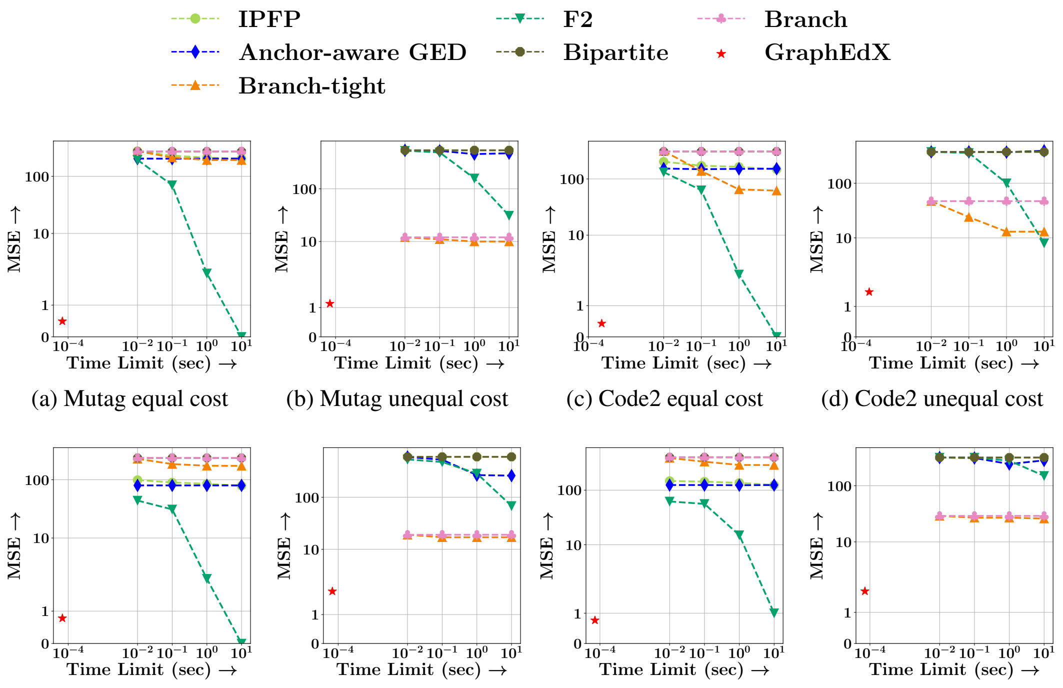
(g) Molpcba equal cost



(h) Molpcba unequal cost

Figure 24: Error distribution of our model compared to the next best-performing model across various datasets under both equal and unequal cost settings.

F.11 Comparison of Combinatorial Optimisation Gadgets for GED prediction



(a) Mutag equal cost (b) Mutag unequal cost (c) Code2 equal cost (d) Code2 unequal cost

(e) AIDS equal cost (f) AIDS unequal cost (g) Linux equal cost (h) Linux unequal cost

Figure 25: Performance of combinatorial optimization algorithms on various datasets under both equal and unequal cost settings is evaluated. We plot MSE against the time limit allocated to the combinatorial algorithms. Additionally, we include the amortized time of our model and its MSE.

We compare the runtime performance of six combinatorial optimization algorithms described in Appendix E (ipfp [11], anchor-aware GED [15], branch tight [8], F2 [29], bipartite [41] and branch [8]). We note that combinatorial algorithms are slow to approximate the GED between two graphs. Specifically, GRAPHEDX often predicts the GED in $\sim 10^{-4}$ seconds per graph, however, the performance of the combinatorial baselines are extremely poor under such a time constraint. Hence, we execute the combinatorial algorithms with four different time limits per graph: ranging from 10^{-2} seconds (100x our method) to 10 seconds (10^5x our method).

In Figure 25, we depict the MSE versus time limit for the aforementioned combinatorial algorithms under both equal and unequal cost settings. We also showcase the inference time per graph of our method in the figure. It is evident that even with a time limit scaled by 10^5x, most combinatorial algorithms struggle to achieve a satisfactory approximation for the GED.

F.12 Prediction timing analysis

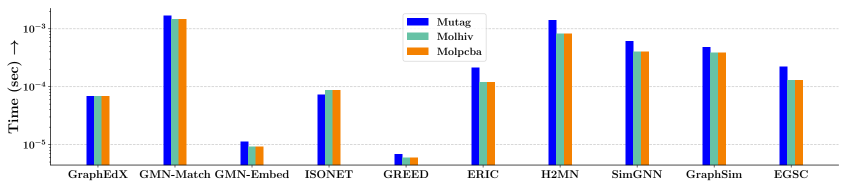


Figure 26: GED inference time comparison between our model and baselines. We notice that GRAPHEDX is consistently the third-fastest amongst all baselines. Although GMN-Embed and GREED have the lowest inference time, GRAPHEDX has much lower MSE consistently.

In Figure 26 illustrates the inference time per graph of our model versus under equal cost settings, averaged over ten runs. From the figure, we observe the following (1) GRAPHEDX outperforms most of the baselines in terms of inference time (2) GMN-Embed and GREED, run faster compared to all other methods due to lack of interaction between graphs, which results in poor performance at predicting the GED.

F.13 Visualization (Optimal edit path) + Pseudocode

In Algorithm 1, we present the pseudocode to generate the optimal edit path given the learnt node and edge alignments from GRAPHEDX. Figure 27 demonstrates how the operations in the edit path can be utilized to convert G to G'.

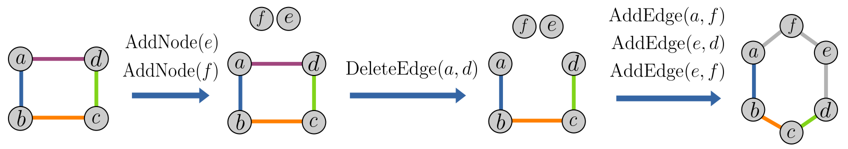


Figure 27: An example of the sequence of edit operations performed to convert one graph into another.

Algorithm 1 Generation of Edit Path

```
1:  function GETEDITPATH($G, G', \eta_G, \eta_{G'}$)
2:      $P, S \leftarrow$ GRAPHEDX($G, G', \eta_G, \eta_{G'}$)
3:      $P, S \leftarrow$ HUNGARIAN($P$), HUNGARIAN($S$)
4:      $o =$ NewList()
5:      for $(u, v) \in [N] \times [N]$ do
6:          if $P[u, v] = 1$ and $\eta_G[u] = 0$ and $\eta_{G'}[v] = 1$ then
7:              AddItem($o$,ADDNODE($u$))
8:      for $(u, v), (u', v') \in \{[N] \times [N]\} \times \{[N] \times [N]\}$ do
9:          if $S[(u, v), (u', v')] = 1$ and $A[u, v] = 0$ and $A'[u', v'] = 1$ then
10:             AddItem($o$,ADDEDGE($(u, v)$))
11:         if $S[(u, v), (u', v')] = 1$ and $A[u, v] = 1$ and $A'[u', v'] = 0$ then
12:             AddItem($o$,DELEDGE($(u, v)$))
13:     for $(u, v) \in [N] \times [N]$ do
14:         if $P[u, v] = 1$ and $\eta_G[u] = 1$ and $\eta_{G'}[v] = 0$ then
15:             AddItem($o$,DELNODE($u$))
16:     return $o$
```

F.14 Comparison of number of parameters

In Table 28, we present the number of parameters for each model used in the experiments.

	GMN-Match	GMN-Embed	ISONET	GREED	ERIC	SimGNN	H2MN	GraphSim	EGSC	GRAPHEDX
# Parameters	2300	2000	2595	2464	5932	1773	3004	6331	4278	3030

Table 28: Number of parameters of all methods