EPIC: Graph Augmentation with Edit Path Interpolation via Learnable Cost

Jaeseung Heo[1] , **Seungbeom Lee**[1] , **Sungsoo Ahn**[1,2] and **Dongwoo Kim**[1,2]
[1]Graduate School of Artificial Intelligence, POSTECH, South Korea
[2]Department of Computer Science & Engineering, POSTECH, South Korea
{jsheo12304, slee2020, sungsoo.ahn, dongwookim}@postech.ac.kr

Abstract

Data augmentation plays a critical role in improving model performance across various domains, but it becomes challenging with graph data due to their complex and irregular structure. To address this issue, we propose EPIC (**E**dit **P**ath **I**nterpolation via learnable **C**ost), a novel interpolation-based method for augmenting graph datasets. To interpolate between two graphs lying in an irregular domain, EPIC leverages the concept of graph edit distance, constructing an edit path that represents the transformation process between two graphs via edit operations. Moreover, our method introduces a context-sensitive cost model that accounts for the importance of specific edit operations formulated through a learning framework. This allows for a more nuanced transformation process, where the edit distance is not merely count-based but reflects meaningful graph attributes. With randomly sampled graphs from the edit path, we enrich the training set to enhance the generalization capability of classification models. Experimental evaluations across several benchmark datasets demonstrate that our approach outperforms existing augmentation techniques in many tasks.

1 Introduction

Graph data has become increasingly important in various domains, such as social networks, bioinformatics, and recommendation systems [Hu *et al.*, 2020; Morris *et al.*, 2020; Leskovec *et al.*, 2007]. Despite the increasing importance, the limited size and diversity of existing graph datasets often limit the performance of graph-based models. One way to overcome this limitation is to augment the existing dataset, a technique that has found success in the other domains [Shorten and Khoshgoftaar, 2019; Iwana and Uchida, 2021].

An interpolation between points is a recognized method for data augmentation, yet this approach does not seamlessly translate to graphs, due to the irregular structures inherent to them. Prior solutions have either adopted a continuous relaxation of graph structures into regular domains [Han *et al.*, 2022] or employed transplantation of graph components [Park *et al.*, 2022], thereby circumventing the difficulties associated with graph interpolation.

We propose a novel interpolation method for graphs based on the concept of graph edit distance [Ullmann, 1976]. The graph edit distance, a widely-accepted metric, quantifies graph similarity by determining the minimum number of necessary edit operations, such as insertions, deletions, or substitutions of nodes and edges. From the graph edit distance between two graphs, we construct a graph edit path representing the transformation process from one graph to the other through the edit operations. The intermediate graphs along this path can be seen as an interpolation between two graphs and are used to augment the training set.

However, the conventional edit distance with operation counts may overlook the significance of individual operations, such as those that affect functional groups in molecular graphs [Durant *et al.*, 2002; Zhang *et al.*, 2020]. In response to this, we propose a context-sensitive cost model built through an edit distance learning framework that maximizes the distance between graphs with different labels while minimizing those with the same label. This enables us to learn a cost model that better reflects the underlying graph data characteristics.

By combining the graph edit distance metric with the learned cost model, we present a novel graph dataset augmentation method named Edit Path Interpolation via learnable Cost (EPIC). Through rigorous experimental evaluations on various graph classification tasks, we demonstrate the superiority of our method in comparison to existing techniques. Additional experiments with noisy labels show the robustness of our approach against the others.

2 Related Work

2.1 Graph Edit Distance

The graph edit distance is a metric that quantifies the dissimilarity between two graphs [Ullmann, 1976]. It measures the minimum number of edit operations required to transform one graph into another. These edit operations include node and edge insertion, deletion, and substitution. The computational complexity of obtaining graph edit distance is known to NP-complete [Abu-Aisheh *et al.*, 2015]. A number of works addressing the problem of the high computational cost have been proposed. Cross *et al*. [1997] casts the optimization pro-

cess into a Bayesian framework with a genetic search algorithm. Myers *et al*. [2000] adopts the Levenshtein distance to model the probability distribution for structural errors in the graph-matching. In Justice and Hero [2006], a binary linear programming formulation of the graph edit distance for unweighted, undirected graphs is proposed. Fischer *et al*. [2015] proposes an approximated graph edit distance based on Hausdorff matching. Wang *et al*. [2021a] adopts the deep learning method to efficiently prune the search tree in computation.

Recently, deep neural network-based graph edit distance learning methods have been proposed. In contrast to the traditional fixed cost distance, these methods learn the cost of individual edit operations. One standard approach to learning costs using neural networks involves obtaining embeddings from node and edge attributes, which are then used to compute the edit distance in a supervised manner [Cortés *et al.*, 2019]. However, these approaches require ground truth node correspondences provided by an oracle between two graphs, which are inapplicable in various situations. Riba *et al*. [2021] learns the cost using node features extracted by graph neural networks and optimize the cost by approximating graph edit distance to Hausdorff distance. The Hausdorff distance is an effective method for approximating the graph edit distance in quadratic time, but it is not suitable for edit path construction since it allows one-to-many substitution between nodes.

2.2 Graph Augmentation

Augmentation methods for graph-structured data aim to improve the generalization ability of neural networks by creating diverse training samples. The most commonly used augmentation methods are based on random modifications of original data. For example, DropNode [Feng *et al.*, 2020] and DropEdge [Rong *et al.*, 2019] uniformly drop nodes and edges, respectively. A subgraph sampling [You *et al.*, 2020; Wang *et al.*, 2020] and motif swap [Zhou *et al.*, 2020] perturb the subgraph of the original graph via subgraph matching. FLAG [Kong *et al.*, 2022] adds adversarial perturbation to node features. These methods assign the same labels before and after the perturbation.

To overcome the simplicity of basic approaches, mixup-based augmentation has been proposed. Manifold Mixup [Wang *et al.*, 2021b] interpolates embeddings from the last layer for two graphs and uses it as a graph representation of the augmented graph. SubMix [Yoo *et al.*, 2022] proposes a node split and merge algorithm to perturb original graphs and then mix random subgraphs of multiple graphs. S-Mixup [Ling *et al.*, 2023] uses an alignment matrix from the graph matching network [Li *et al.*, 2019] to mix node features. G-Mixup [Han *et al.*, 2022] mixes graphons [Airoldi *et al.*, 2013] extracted from different classes of graphs. GREA [Liu *et al.*, 2022] proposes a rationalization identification algorithm that extracts a subgraph explaining a property of the graph best and uses the subgraph to curate augmented graphs.

3 EPIC: Edit Path Interpolation via Learnable Cost

In this section, we first describe a graph data augmentation method with a graph edit path. We then propose a method to learn a graph edit distance by learning the cost of individual edit operations.

3.1 Augmentation with Graph Edit Path

Construction of Graph Edit Path

We consider a graph $G = (\mathcal{V}, \mathcal{E})$ associated with node and edge attributes. The graph edit distance is a metric that quantifies the dissimilarity between two graphs. It measures the minimum number of edit operations required to transform one graph into another or computes the total cost of edit operations if the cost of individual operations varies. These edit operations involve node and edge insertion, deletion, and attribute substitution.

Once the graph edit distance is computed, a graph edit path can be obtained by applying a series of edit operations from a source graph to reach a target graph. It represents the step-by-step transformation from one graph to another while minimizing the edit distance.

The graph edit distance is generally invariant to the order of edit operations. However, there are certain dependencies between node and edge operations. The node deletion operation can only be performed after all the connected edges are deleted. The edge insertion operation can only be performed when the two target nodes are presented.

We only consider the node operations in order to reduce the computational complexity of graph edit distance. Specifically, when a new node is inserted into a graph, we perform edge insertion operations for all edges whose adjacent nodes are given after insertion. When a node is deleted, we perform edge deletion operations for all edges connected to the deleted node. When a node is substituted, we perform edge insertion, deletion, and substitution operations accordingly. The detailed algorithm is provided in Algorithm 1. By doing so, we can construct the graph edit path whose length is equal to the number of node edit operations in the computation of the graph edit distance. Figure 1 illustrates an example of a graph edit path between two graphs and possible augmentation.

Graph Augmentation and Label Assignment

We use the graph edit path to construct an augmented graph. We randomly sample two graphs in the training set. The graph edit distance between the two graphs is computed, then the graph edit path is constructed with node operations in random order. The samples obtained from a graph edit path are used as augmented graphs. To prevent the case where the augmented graph is already presented in the training set, we add an additional node feature indicating whether the graph is from the augmentation set or not.

To assign a label to the augmented graph, we use the cost of edit operations from the augmented graph to the source and the target graphs. Let $(o_1, ..., o_n)$ be a sequence of edit operations applied to transforming source graph G_S to target graph G_T, and $c(o)$ be the real-valued cost function of operation o, which is precisely defined in Subsection 3.2. With the

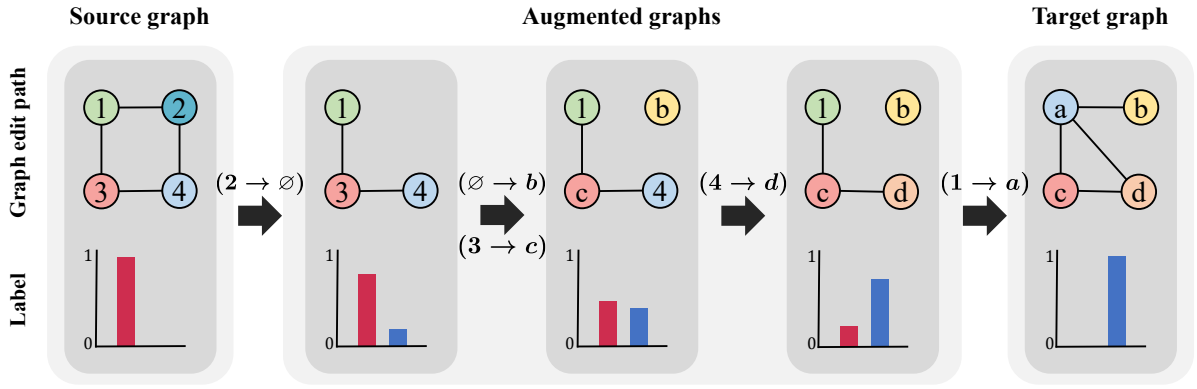


Figure 1: Illustration of graph edit path and their corresponding labels for augmentation.

Algorithm 1 Applying node operation

Input: A node operation $(u \rightarrow v)$, source graph $G_S = (\mathcal{V}_S, \mathcal{E}_S)$, target graph $G_T = (\mathcal{V}_T, \mathcal{E}_T)$, and the current graph $G_C = (\mathcal{V}_C, \mathcal{E}_C)$
Output: The updated graph $G_{C+1} = (\mathcal{V}_{C+1}, \mathcal{E}_{C+1})$

$G_{C+1} = \text{copy}(G_C)$
if $u = \varnothing$ **then**
  Insert the node v to $\mathcal{V}_{C+1}$.
else if $v = \varnothing$ **then**
  Delete the node u from $\mathcal{V}_{C+1}$.
else
  Substitute the node u with the node v in $\mathcal{V}_{C+1}$.
end if
for each edge (u, w) in $\mathcal{E}_C$ **do**
  if node $w \in \mathcal{V}_T$ and edge (v, w) not in $\mathcal{E}_T$ **then**
    Remove the edge (v, w) from $\mathcal{E}_{C+1}$.
  else if node $w \in \mathcal{V}_T$ and edge (v, w) in $\mathcal{E}_T$ **then**
    Substitute edge attributes of (v, w) in $\mathcal{E}_{C+1}$ with those of (v, w) in $\mathcal{E}_T$ if needed.
  end if
end for
for each edge $(v, x) \in \mathcal{E}_T$ adjacent to node v in $\mathcal{V}_T$ **do**
  if node $x \in \mathcal{V}_C$ and edge (u, x) not in $\mathcal{E}_C$ **then**
    Insert the edge (v, x) to $\mathcal{E}_{C+1}$.
  end if
end for

corresponding one-hot classification label of the source graph $\boldsymbol{y}_S$ and the target graph $\boldsymbol{y}_T$, the label of the augmented graph $\bar{\boldsymbol{y}}$ obtained by applying the first m operations $(o_1, ..., o_m)$ is computed as

$$\bar{\boldsymbol{y}} = \frac{\sum_{i=m+1}^{n} c\left(o_i\right)}{\sum_{i=1}^{n} c\left(o_i\right)} \boldsymbol{y}_S + \frac{\sum_{i=1}^{m} c\left(o_i\right)}{\sum_{i=1}^{n} c\left(o_i\right)} \boldsymbol{y}_T, \tag{1}$$

where $c(\cdot)$ measures the cost of an operation. The assigned label is inversely proportional to the operation cost to reach the source or target from the augmented graph.

3.2 Learning Costs of Edit Operations

The standard unit cost model [Ullmann, 1976; Fischer *et al.*, 2015; Riesen and Bunke, 2009] of the graph edit distance is incapable of measuring the importance of each operation as all operations are assigned the same cost, regardless of their significance or impact. However, the importance of edit operations would differ based on the context of the dataset. For example, changes in a functional group of molecular graphs can lead to larger semantic perturbation than the other parts in a property prediction task [Durant *et al.*, 2002; Zhang *et al.*, 2020]. Therefore, the edit operation leading to a large semantic modification should cost more than the others. To measure the importance of each operator in the computation of graph edit distance, we propose a learning algorithm for the operation cost based on a neural network model.

Triplet Loss for Learning Distance

A good cost function should be problem dependent. We use the triplet loss with known labels [Schroff *et al.*, 2015] to learn the graph edit distance and the operation costs therein. We assume that the pair of graphs within the same class has a relatively shorter edit distance than those of different classes. Let $\text{GED}(G, G')$ be the distance between graphs G and G'. We propose a triplet loss-based objective function to encode our intuition:

$$\begin{aligned} &\mathcal{L}(G, G^+, G^-) \\ &\quad = \max\left(\text{GED}(G, G^+) - \text{GED}(G, G^-) + \gamma, 0\right), \end{aligned} \tag{2}$$

where G^+ is a positive example whose label is the same as G, G^- is a negative example whose labels are different from G, and γ is a margin hyperparameter.

Graph Edit Distance as Constrained Optimization

The computational complexity of graph edit distance computation is NP-complete [Abu-Aisheh *et al.*, 2015]. Learning graph edit distance often requires relaxation to make the algorithm tractable [Fischer *et al.*, 2015; Riesen and Bunke, 2009]. To simplify the learning, we assume that the cost of node operation *subsumes* the cost of dependent edit operations, similar to the construction of the edit path. With the simplified assumption, we only need to consider the following three cases when computing the edit distance between source graph $G_S = (\mathcal{V}_S, \mathcal{E}_S)$ and target graph $G_T = (\mathcal{V}_T, \mathcal{E}_T)$ with operation cost function $c : \mathcal{V}_S \cup \varnothing \times \mathcal{V}_T \cup \varnothing \rightarrow \mathbb{R}$, where $\varnothing$ represents an empty node:

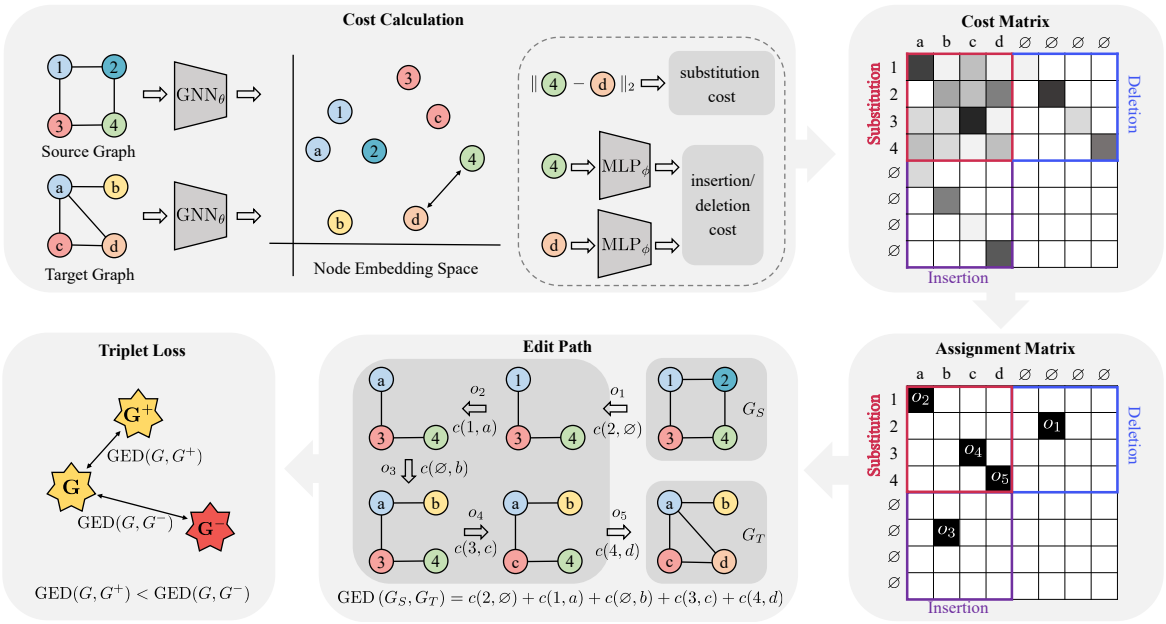


Figure 2: Overall illustration of graph edit distance learning. The assignment matrix can be obtained by either Hungarian or Sinkhorn-Knopp. For learning, we use Sinkhorn-Knopp, and for augmentation, we use Hungarian.

- Node u in $\mathcal{V}_S$ is substituted by node v in $\mathcal{V}_T$ with substitution cost $c(u, v)$.
- Node u in $\mathcal{V}_S$ is deleted with deletion cost $c(u, \varnothing)$.
- Node v in $\mathcal{V}_T$ is inserted with insertion cost $c(\varnothing, v)$.

We construct a cost matrix that encapsulates all required costs to compute the edit distance between two graphs. The cost matrix is constructed as

$$C = \begin{bmatrix} c(u_1, v_1) & \cdots & c(u_1, v_m) & c(u_1, \varnothing) & \cdots & \infty \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ c(u_n, v_1) & \cdots & c(u_n, v_m) & \infty & \cdots & c(u_n, \varnothing) \\ c(\varnothing, v_1) & \cdots & \infty & \infty & \cdots & \infty \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \infty & \cdots & c(\varnothing, v_m) & \infty & \cdots & \infty \end{bmatrix} \tag{3}$$

where n and m are the number of nodes in G_S and G_T, respectively.

With the cost matrix, the problem of computing the graph edit distance can be reduced to solving the assignment problem. Since the node in the source graph can only be substituted or deleted, only one operation can be performed in each of the first n rows. Similarly, since the node in the target graph can only be substituted or inserted, only one operation in each of the first m columns can be performed in edit distance computation. A binary assignment matrix X, whose size is the same as the cost matrix, is introduced to indicate which operation is performed in edit distance computation. With the assignment matrix, the computation of graph edit distance can be formulated as a constrained optimization problem

$$\begin{aligned} \mathrm{GED}(G_S, G_T) = &\min_X \sum_{i=1}^{n+m} \sum_{j=1}^{n+m} C_{ij} X_{ij} \\ \text{s.t.} \quad &\sum_{j=1}^{n+m} X_{ij} = 1, \quad 1 \le i \le n \\ &\sum_{i=1}^{n+m} X_{ij} = 1, \quad 1 \le j \le m, \quad X_{ij} \in \{0, 1\}. \end{aligned} \tag{4}$$

Design Cost Function with Neural Networks

We introduce the graph neural network framework to parameterize the cost function c. Specifically, we use the embedding distances between two nodes as the substitution cost. Let h_u and h_v be the output embedding of node $u \in G_S$ and $v \in G_T$ from a graph neural network. We use the distance between two embeddings as a substitution cost, i.e., $c_\theta(u, v) = ||h_u - h_v||_2$, where θ is the parameter of the graph neural network. The embeddings of the graph neural network encode the neighborhood structure of the target node. If the embeddings of two nodes are similar, then the two nodes are likely to play a similar role in the graph. Hence, the substitution cost measures the structural similarity between two nodes. For the insertion and deletion operations, we additionally introduce a multi-layer perceptron, i.e., $c_{\theta,\phi}(u, \varnothing) = \mathrm{MLP}_\phi(h_u)$ and $c_{\theta,\phi}(\varnothing, v) = \mathrm{MLP}_\phi(h_v)$, where ϕ denotes the parameters of MLP. The MLP computes the cost of insertion and deletion using the embedding of a node. We use the same network for both insertion and deletion. The graph neural networks encode the local structure of nodes into node embeddings. Consequently, by considering the costs of node operations, we effectively encapsulate the information regarding the neighborhood edges as well.

	Method	NCI1	BZR	COX2	Mutagen.	IMDB-B	IMDB-M	PROTEINS	ENZYMES	Rank
GIN	Vanilla	81.68(0.8)	87.07(2.7)	83.40(2.7)	81.57(0.7)	72.90(0.6)	48.40(1.2)	67.80(2.2)	46.33(2.3)	5.4
	DropEdge [Rong *et al.*, 2019]	74.61(0.7)	85.85(0.4)	80.43(2.7)	79.13(1.1)	71.20(1.6)	47.20(2.0)	68.97(1.6)	39.00(3.1)	8.6
	DropNode [Feng *et al.*, 2020]	73.05(1.8)	85.61(3.0)	78.51(1.6)	77.91(1.0)	72.90(3.4)	46.60(2.0)	67.53(3.3)	38.00(3.6)	9.4
	Subgraph [You *et al.*, 2020]	74.90(2.6)	79.76(3.3)	81.49(4.9)	69.25(1.8)	71.90(3.1)	46.13(3.0)	63.32(3.5)	34.50(8.4)	10.3
	FLAG [Kong *et al.*, 2022]	81.85(0.8)	88.29(1.6)	84.26(2.6)	81.98(0.6)	72.80(0.3)	48.60(1.4)	68.34(2.9)	46.17(2.5)	4.1
	SubMix [Yoo *et al.*, 2022]	81.31(0.6)	87.56(3.5)	83.62(2.3)	80.25(1.1)	72.90(0.9)	48.60(1.0)	70.76(2.2)	44.33(4.8)	4.8
	M-Mix [Wang *et al.*, 2021b]	81.00(0.4)	85.37(3.6)	**84.47**(1.0)	81.68(0.9)	**73.20**(0.6)	47.93(1.8)	68.70(1.1)	46.67(1.7)	4.9
	G-Mix [Han *et al.*, 2022]	80.92(1.9)	88.05(1.6)	82.98(0.8)	81.66(0.6)	73.00(0.9)	48.13(1.2)	69.87(2.4)	44.83(1.4)	5.1
	GREA [Liu *et al.*, 2022]	74.63(0.5)	85.85(2.2)	81.91(2.4)	79.98(0.4)	72.40(1.0)	46.27(0.7)	**73.36**(1.4)	46.83(2.1)	6.6
	S-Mix [Ling *et al.*, 2023]	81.48(1.4)	85.85(2.5)	83.83(0.5)	82.07(1.1)	73.00(2.0)	48.47(0.6)	69.42(1.6)	44.33(6.3)	4.4
	EPIC	**82.16**(0.7)	**88.78**(3.3)	**84.47**(1.9)	**82.30**(0.5)	**73.20**(0.4)	**49.07**(1.2)	71.21(1.4)	**47.50**(1.8)	**1.1**
GCN	Vanilla	81.07(0.7)	85.85(2.8)	84.26(2.2)	81.70(1.1)	70.60(1.3)	**48.27**(1.5)	64.13(1.6)	44.67(5.7)	5.1
	DropEdge [Rong *et al.*, 2019]	74.24(1.0)	82.93(4.3)	83.40(1.2)	80.37(0.6)	70.50(1.8)	45.67(1.5)	64.75(3.4)	39.33(1.8)	8.0
	DropNode [Feng *et al.*, 2020]	73.78(1.8)	80.73(2.4)	79.15(1.8)	78.32(1.6)	69.40(3.0)	39.80(3.7)	68.43(1.8)	36.17(3.6)	9.1
	Subgraph [You *et al.*, 2020]	70.11(5.5)	77.80(1.0)	78.09(3.2)	79.47(1.3)	59.00(6.6)	39.07(2.1)	66.28(2.5)	28.33(8.2)	10.5
	FLAG [Kong *et al.*, 2022]	82.02(0.6)	88.05(2.3)	85.32(1.4)	81.01(0.8)	71.00(2.0)	47.33(2.7)	67.62(1.2)	**47.50**(4.4)	3.1
	SubMix [Yoo *et al.*, 2022]	81.99(0.6)	86.34(2.0)	84.68(3.7)	80.99(0.6)	70.30(1.4)	46.47(2.5)	67.80(2.0)	43.67(4.9)	5.1
	M-Mix [Wang *et al.*, 2021b]	81.41(0.5)	84.15(2.3)	83.83(2.1)	81.96(0.6)	69.40(1.1)	46.40(2.7)	66.28(1.5)	44.00(5.3)	6.1
	G-Mix [Han *et al.*, 2022]	82.04(1.8)	87.32(2.4)	84.89(1.4)	80.32(0.7)	69.90(1.8)	45.87(3.0)	68.07(1.2)	46.17(4.3)	4.8
	GREA [Liu *et al.*, 2022]	75.31(0.4)	84.39(0.9)	80.43(0.5)	80.48(0.5)	65.50(0.0)	45.73(1.4)	**73.18**(0.7)	38.00(2.2)	7.4
	S-Mix [Ling *et al.*, 2023]	81.65(0.7)	87.56(2.8)	83.40(1.4)	81.63(0.4)	69.60(0.8)	46.20(0.8)	68.52(0.8)	45.00(3.2)	4.8
	EPIC	**82.14**(0.8)	**88.78**(2.3)	**85.53**(1.6)	**82.44**(0.7)	**71.70**(1.0)	47.93(1.3)	69.06(1.0)	45.00(3.0)	**1.5**

Table 1: Classification accuracy of TUDataset. We report the average and standard deviation (in brackets) over five seeds. We mark the best and the second-best performances in **bold** and underline, respectively. The rank column shows the average rank of model performance across all datasets.

Optimization with a Differentiable Assignment Matrix

To learn the graph edit distance, we need to minimize the loss in Equation 2 w.r.t θ and ϕ. However, this optimization involves a non-differentiable optimization problem w.r.t the assignment matrix X in Equation 4. The Hungarian algorithm [Kuhn, 1955] can be used to find the optimal assignment X for each iteration of the stochastic gradient descent step. However, the Hungarian algorithm is non-differentiable and has a computational complexity of $O(n^3)$, making it difficult to employ during gradient-based optimization. We instead employ the Sinkhorn-Knopp algorithm [Sinkhorn and Knopp, 1967] to address this issue to obtain a differentiable assignment matrix. The Sinkhorn-Knopp algorithm transforms a non-negative matrix into a doubly stochastic matrix to approximate the Hungarian algorithm. Specifically, Sinkhorn-Knopp iteratively updates a soft assignment matrix $\tilde{X}$ via two intermediate variables $\boldsymbol{u}$ and $\boldsymbol{v}$. Once $\boldsymbol{u}$ and $\boldsymbol{v}$ are initialized as a vector of ones, i.e., $\boldsymbol{u}^{(0)} = [1, \cdots, 1]^\top$, at k-th iteration of Sinkhorn-Knopp approximates the assignment matrix via

$$\boldsymbol{u}^{(k)} = \frac{X^{(k-1)}\mathbf{1}}{K\boldsymbol{v}^{(k-1)}}, \quad \boldsymbol{v}^{(k)} = \frac{X^{(k-1)\top}\mathbf{1}}{K^\top \boldsymbol{u}^{(k-1)}},$$
$$\tilde{X}^{(k)} = \mathrm{diag}(\boldsymbol{u}^{(k)}) K \,\mathrm{diag}(\boldsymbol{v}^{(k)}), \tag{5}$$

where each entry of matrix K is parameterized by the cost matrix and a regularizer parameter δ as $K_{ij} = \exp(-C_{ij}/\delta)$, and $\mathbf{1}$ is a vector of ones. δ is a regularization term controlling the sharpness of the assignment matrix.

Note that the back-propagation algorithm needs to optimize the entire iterative process of the Sinkhorn-Knopp approximation. In experiments, we set the number of maximum iterations to 10 to reduce the computational cost. After learning the cost function, to augment a pair of randomly selected graphs, we first compute the cost matrix and then create a graph edit path using the optimal assignment from the Hungarian algorithm. Figure 2 shows the overall illustration of our proposed approach.

4 Experiments

In this section, we first show the effect of EPIC in graph classification tasks over 11 datasets. We further evaluate the robustness of GNNs with our method against corrupted labels. We provide additional analysis of our model selection process.

4.1 Effect of Augmentation for Graph Classification

Datasets. We used eight classification datasets: NCI1, BZR, COX2, Mutagenicity, IMDB-BINARY, IMDB-MULTI, PROTEINS, ENZYMES from TUDataset [Morris *et al.*, 2020] and three classification dataset: BBBP, BACE, HIV

from MoleculeNet [Wu *et al.*, 2018]. The datasets cover a wide range of tasks, including social networks, bioinformatics, and molecules.

Baselines. For baseline augmentation models, we use two graph augmentation methods as baselines: one that modifies a single graph, DropEdge [Rong *et al.*, 2019], DropNode [Feng *et al.*, 2020], Subgraph [You *et al.*, 2020] and FLAG [Kong *et al.*, 2022], another that mixes information from two graphs, SubMix [Yoo *et al.*, 2022], Manifold-Mixup (M-Mixup) [Wang *et al.*, 2021b], G-Mixup [Han *et al.*, 2022], GREA [Liu *et al.*, 2022], and S-Mixup [Ling *et al.*, 2023]. We also report the performance of a vanilla model without augmentation.

Implementation details. We first learn the cost of edit operations for each dataset. We use Adam optimizer [Kingma and Ba, 2014] with a learning rate decay of 0.1 every 25 epochs. We train the cost function for 100 epochs on TUDataset. While we use the Sinkhorn-Knopp approximation with $k = 10$ in Equation 5 for training, the Hungarian algorithm is used for inference to obtain an optimal assignment given costs. We perform graph classification tasks with GIN [Xu *et al.*, 2018] and GCN [Kipf and Welling, 2016] as a backbone model for augmentation. When we train each backbone model, for a fair comparison, we use the same hyperparameters and architecture tuned from the vanilla model for our method and other baseline models. We follow the Open Graph Benchmark setting [Hu *et al.*, 2020] for MoleculeNet dataset. When training classification models, we compute the edit path between randomly paired graphs in each batch and use randomly chosen graphs from the edit path as augmentation. We use the validation set to choose the portion of augmented data points.

Classification results. Table 1 shows the overall results of the TUDataset for graph classification tasks. Our augmentation method outperforms the other baselines on seven and five datasets with GIN and GCN backbones, respectively, and achieves the second-best performance on one and two datasets with GIN and GCN. Table 2 shows the AUC-ROC scores with MoleculeNet datasets. We achieve the best performance on all datasets except for one second-best performance with GIN and GCN backbones over other baselines. The results from two benchmarks show our augmentation method consistently improves the generalization of GNNs.

4.2 Robustness Analysis

Verma *et al.* [2019] theoretically demonstrates that mixup-based augmentation enhances the robustness of deep neural networks against noisy labels. We also conduct a similar study. Following the settings used in Han *et al.* [2022] and Ling *et al.* [2023], we randomly corrupt labels in the training set of IMDB-BINARY, IMDB-MULTI, and Mutagenicity datasets and test the model performance with the uncorrupted test set. We run the experiments with three different proportions of noise: $\{0.2, 0.4, 0.6\}$ based on GIN. Except for the noise, we use the same setting used in Subsection 4.1. Table 3 shows the classification accuracy with different proportions of noisy labels. EPIC outperforms the other baseline models, except for one case, showing the robustness of our augmentation under the noisy environment.

	Method	BBBP	BACE	HIV	Rank
GIN	Vanilla	65.90(1.9)	77.01(2.7)	75.10(2.7)	6.3
	DropEdge	65.32(1.8)	74.50(0.9)	75.57(0.6)	8.0
	DropNode	64.32(5.0)	76.37(3.0)	75.36(1.3)	8.3
	Subgraph	62.85(5.7)	76.15(4.3)	73.84(0.9)	10.0
	FLAG	65.64(0.8)	79.45(4.5)	75.24(1.9)	5.7
	SubMix	65.58(5.0)	77.50(5.5)	76.36(1.3)	4.3
	M-Mix	64.48(1.6)	75.30(2.6)	75.55(2.0)	8.0
	G-Mix	64.33(3.2)	78.54(1.9)	76.06(0.7)	5.7
	GREA	68.57(0.6)	79.97(1.2)	76.18(1.1)	2.3
	S-Mix	65.89(2.0)	75.04(5.5)	75.63(3.0)	6.3
	EPIC	**68.88**(2.1)	**81.00**(1.1)	**76.38**(0.3)	**1.0**
GCN	Vanilla	66.08(3.4)	76.35(4.2)	75.45(0.7)	7.0
	DropEdge	65.71(2.5)	72.79(7.9)	75.90(1.6)	7.7
	DropNode	68.33(3.0)	71.37(5.0)	74.41(0.9)	7.7
	Subgraph	63.09(0.6)	76.39(1.3)	73.75(1.7)	9.3
	FLAG	68.14(3.5)	77.87(1.9)	76.74(3.2)	3.3
	SubMix	67.68(1.2)	75.19(5.6)	75.61(2.1)	6.3
	M-Mix	67.38(2.0)	**79.67**(0.9)	75.23(1.2)	5.3
	G-Mix	65.10(2.3)	77.54(5.3)	75.99(0.9)	6.0
	GREA	68.23(1.9)	77.29(4.0)	76.85(1.7)	3.3
	S-Mix	65.65(2.7)	74.04(2.6)	75.32(3.8)	8.7
	EPIC	**68.52**(1.7)	78.05(1.3)	**77.30**(3.1)	**1.3**

Table 2: Classification AUC-ROC of MoleculeNet

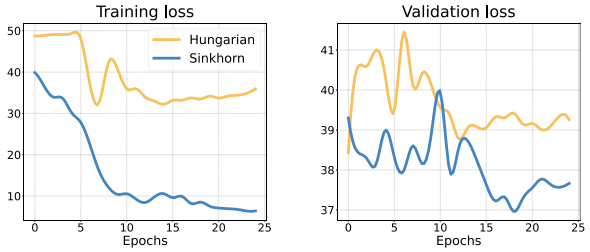


Figure 3: Training and validation loss curves on HIV dataset with Hungarian and Sinkhorn-Knopp algorithms in the training process.

4.3 Ablation Studies

In this subsection, we show the result of ablation studies on each component in EPIC. Further ablation studies are shown in Appendix A.

Cost function variations. We test the effectiveness of the learnable cost function against other variations of the cost function. We use two variations of the cost function: unit cost, which measures the number of edit operations, and feature-distance cost, which measures the distance between two input node features, adopted from Ling *et al.* [2023]. The result in Table 4a shows that the learnable cost outperforms the other cost functions across all datasets. The results empirically verify our claim that the good cost function should be problem-dependent and can be learned from the dataset. We

Method	IMDB-BINARY			IMDB-MULTI			Mutagenicity		
	20%	40%	60%	20%	40%	60%	20%	40%	60%
Vanilla	72.50(1.2)	66.10(1.6)	44.30(6.6)	48.00(1.2)	46.07(1.1)	42.33(3.7)	75.30(0.7)	68.52(1.4)	46.38(3.1)
DropEdge [Rong *et al.*, 2019]	70.30(4.1)	66.20(5.1)	48.30(4.0)	44.80(2.4)	42.60(3.1)	37.93(2.2)	75.47(0.9)	67.71(1.5)	45.13(4.1)
DropNode [Feng *et al.*, 2020]	70.90(1.0)	68.80(3.4)	52.40(2.3)	45.60(1.9)	43.00(2.5)	39.53(1.9)	72.98(1.5)	66.26(1.5)	45.89(3.1)
SubMix [Yoo *et al.*, 2022]	72.20(1.0)	67.70(2.0)	43.10(9.4)	48.67(1.2)	45.53(1.3)	40.60(5.2)	73.00(1.5)	66.86(2.6)	**59.03**(2.5)
ManifoldMix [Verma *et al.*, 2019]	71.90(1.1)	69.90(3.8)	49.70(6.5)	48.07(1.3)	46.67(1.1)	43.40(3.3)	75.49(0.9)	67.43(3.6)	45.91(2.7)
G-Mix [Han *et al.*, 2022]	71.60(1.9)	70.00(4.3)	44.60(10.0)	47.33(0.8)	44.80(2.1)	40.67(1.7)	76.06(1.5)	69.25(0.8)	45.27(4.1)
GREA [Liu *et al.*, 2022]	71.80(1.8)	67.50(5.8)	49.20(1.8)	45.53(1.5)	44.13(3.7)	39.73(1.2)	76.60(1.8)	68.63(1.6)	49.64(7.3)
S-Mix [Ling *et al.*, 2023]	72.60(1.0)	70.30(3.6)	52.50(7.4)	48.40(0.5)	46.60(1.5)	42.07(4.3)	75.56(0.5)	68.22(3.8)	58.50(5.3)
EPIC	**72.90**(1.0)	**71.30**(3.4)	**52.60**(10.3)	**49.00**(0.9)	**47.13**(1.7)	**44.40**(3.7)	**76.78**(1.2)	**69.48**(1.9)	53.90(5.6)

Table 3: Robustness analysis on IMDB-BINARY, IMDB-MULTI, and Mutagenicity datasets.

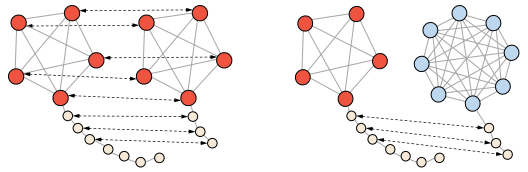
Method	NCI1 ↑	Mutagen.↑	IMDB-B↑	IMDB-M ↑
Unit cost	81.56(1.3)	81.73(0.2)	71.90(0.9)	48.53(1.0)
Feature-distance	81.07(0.9)	81.68(0.8)	72.20(1.2)	48.53(1.0)
EPIC	**81.70**(1.4)	**82.30**(0.5)	**73.20**(0.4)	**49.07**(1.2)

(a) Classification accuracy with augmentation

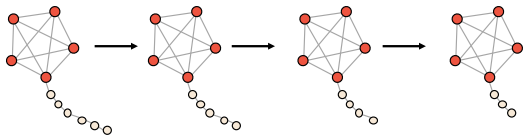
Method	NCI1 ↑	Mutagen.↑	IMDB-B↑	IMDB-M ↑
Unit cost	60.49(0.4)	56.69(2.0)	55.50(2.8)	36.33(2.2)
Feature-distance	60.70(0.8)	55.51(1.8)	56.00(2.8)	36.20(1.6)
EPIC	**72.69**(0.3)	**75.33**(1.0)	**67.70**(1.8)	**39.33**(2.8)

(b) Distance-based classification accuracy

Table 4: Comparisons between three different cost functions.



(a) Nodes with substitution operations



(b) Sampled edit path

Figure 4: Examples of learned edit distance from a lollipop dataset. The dashed lines in (a) represent substitution operations.

also classify the graphs in the test dataset based on their distance to the closest graph in each class. If the cost is learned properly, the distance from a test graph to the graph in the same class should be close to each other. Table 4b shows the result of distance-based classification. In most cases, EPIC outperforms the other fixed-cost methods. Results for all TUDataset can be found in Appendix B.

Hungarian vs Sinkhorn-Knopp. We investigate the impact of a differentiable assignment matrix with the Sinkhorn-Knopp algorithm against a fixed assignment matrix from the Hungarian algorithm in the process of training. In Figure 3, we present the training and validation curve on the HIV dataset. In general, the Sinkhorn-Knopp algorithm shows a more stable learning process than the Hungarian algorithm. The training loss with Hungarian has not been stabilized after 20 epochs in HIV datasets. Moreover, the validation loss of Sinkhorn-Knopp is consistently lower than Hungarian after certain iterations. We conjecture that the non-smooth loss surface of Hungarian makes the gradient descent work hard, and eventually, the model fails to reach a good local minimum, whereas the smooth loss surface of Sinkhorn-Knopp results in a better performance despite being an approximation of Hungarian.

4.4 Qualitative Analysis

To examine how the results of the learned distance are reflected in the augmented graph, we conduct an experiment with a lollipop dataset whose structural properties are easily visualized. The (m, n)-*lollipop* graph consists of a head, a complete graph with m nodes, and a tail, a chain structured n nodes. The lollipop dataset consists of graphs with varying m and n. The label of a graph is the size of the head, i.e., m. Figure 4 displays examples of the trained graph edit distances and the corresponding edit path. With a positive pair, the learned graph edit distance substitutes the head nodes from the source graph for those of the target graph. Eventually, it maintains the complete subgraph of the head along the edit path. Additionally, we present the assigned cost for substitution to each node in Appendix C, which consistently shows the effectiveness of the cost function for EPIC.

5 Conclusion

In this paper, we propose a novel approach for graph dataset augmentation based on the graph edit distance. Our method overcomes the limitations of linear interpolation techniques in the non-Euclidean domain and provides a tailored augmentation solution for graph data. Through extensive experiments on benchmark datasets, we have demonstrated the effectiveness of our approach in improving the performance and robustness of graph-based models.

		k									
	Dataset	1	2	3	4	5	6	7	8	9	10
GIN	BZR	87.56(2.3)	87.32(2.4)	**89.51**(2.5)	86.59(2.6)	88.05(1.6)	87.32(1.8)	87.32(2.4)	87.32(2.0)	88.05(3.2)	88.78(3.3)
	COX2	82.98(1.8)	82.13(3.6)	83.19(4.2)	83.40(2.7)	82.55(2.9)	**84.68**(1.8)	81.06(4.4)	84.47(1.6)	84.47(1.2)	84.47(1.9)
	IMDB-B	72.30(1.3)	72.20(0.8)	72.80(0.6)	72.90(0.5)	72.60(0.7)	72.70(0.9)	72.70(0.8)	72.50(0.7)	72.60(1.1)	**73.20**(0.4)
	IMDB-M	48.40(0.7)	48.33(0.2)	48.67(0.9)	48.53(0.7)	**49.27**(0.6)	48.67(0.2)	48.93(0.9)	49.13(1.6)	48.33(0.8)	49.07(1.2)
	ENZYMES	45.00(3.4)	44.83(3.2)	46.50(3.9)	43.67(4.7)	46.00(5.5)	47.67(2.2)	45.17(2.4)	**49.33**(3.2)	46.00(3.7)	47.50(1.8)
	Rank	7.4	8.6	3.8	6.8	4.6	3.6	6.2	3.8	5.0	**2.2**

Table 5: Classification performance under varying values of k.

	Method	NCI1	BZR	COX2	Mutagen.	IMDB-B	IMDB-M	PROTEINS	ENZYMES
GIN	BFS	81.75(0.8)	**88.78**(2.0)	**84.68**(2.7)	**82.51**(1.0)	73.00(0.5)	**49.60**(0.8)	70.94(1.8)	46.83(2.5)
	Random	**82.16**(0.7)	**88.78**(3.3)	84.47(1.9)	82.30(0.5)	**73.20**(0.4)	49.07(1.2)	**71.21**(1.4)	**47.50**(1.8)

Table 6: Comparison of randomly ordered augmentation and BFS-ordered augmentation.

A Further Ablation Studies

In this section, we describe the ablation studies for model configuration. Specifically, we examine methods for determining the operation order in the edit path and the number of iterations in the Sinkhorn-Knopp algorithm.

Iterations of the Sinkhorn-Knopp algorithm. We adopt the Sinkhorn-Knopp algorithm to approximate the Hungarian algorithm to make the assignment matrix differentiable for optimization during training as described in Subsection 3.2. To analyze the effect of the number of iterations of the Sinkhorn-Knopp algorithm, we evaluate the classification performance under varying values of k in Equation 5. Table 5 presents the results of an ablation study. We further experiment to observe how well the assignment matrix, X_S, derived from the Sinkhorn-Knopp algorithm approximates the assignment matrix, X_H, from the Hungarian algorithm, as the parameter k varies. Firstly, we calculate both assignment matrices from our learned cost function on the test dataset and compute the Frobenius norm of the difference between the two matrices, $||X_A||_F = ||X_S - X_H||_F$, and then, measure an average of each Frobenius norm across the entire dataset. Figure 5 shows the average of Frobenius norm with varying k. We observe that the error decreases as k increases, suggesting that the Sinkhorn-Knopp algorithm's assignment matrix becomes more similar to the assignment matrix from the Hungarian algorithm. However, as k increases, the computational cost required for training also increases, indicating a trade-off relationship.

Operation ordering. When calculating the edit distance, the specific operation order does not affect the final result. To construct the edit path, we need to determine the operation order. In our analysis, we investigate the difference between two choices of operation orders. To maintain the local connectivity of the target graph within the edit path, we employ a breadth-first search (BFS) on the target graph, and node insertion/deletion/substitution is performed following the BFS order. The results in Table 6 compare randomly ordered augmentation with BFS-ordered augmentation on eight datasets. We find no significantly better ordering choices between these two methods. To minimize additional computational overhead, we utilize random ordering for the main result.

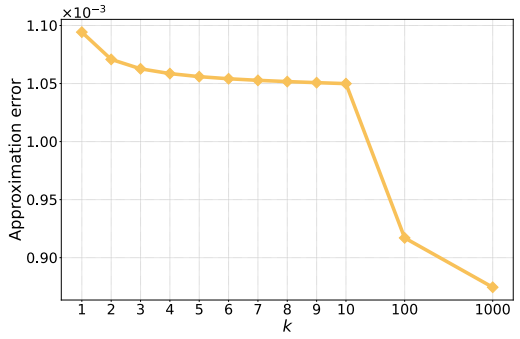


Figure 5: Difference between assignment matrix from Hungarian and Sinkhorn algorithm with varying k in Equation 5.

B Results on TUDataset Based on Cost Function Variations

In this subsection, we show the full result of classification described in Table 4 based on different cost functions: the unit cost, which counts edit operations, and the feature-distance cost, derived from Ling *et al*. [2023], which measures the distance between input node features. Table 7a demonstrates that the learnable cost outperforms other cost functions on all datasets. The result empirically proves our claim that an effective cost function should be problem dependent and learned from the dataset.

Also, if the cost is learned correctly, the graph with the shortest graph edit distance to a given graph should be in the same class. Under this premise, we classify the graphs in the test dataset based on their distance to the closest graph in each

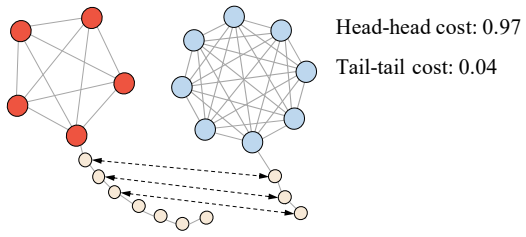
Method	NCI1 ↑	BZR↑	COX2↑	Mutagen.↑	PROTEINS↑	ENZYMES↑	IMDB-B↑	IMDB-M ↑
Unit cost	81.56(1.3)	85.37(2.7)	84.04(2.5)	81.73(0.2)	70.13(0.9)	45.33(3.2)	71.90(0.9)	48.53(1.0)
Feature-distance	81.07(0.9)	87.80(2.3)	82.34(2.4)	81.68(0.8)	71.12(.12)	47.33(1.8)	72.20(1.2)	48.53(1.0)
EPIC	**81.70**(1.4)	**88.78**(3.3)	**84.47**(1.9)	**82.30**(0.5)	**71.21**(1.4)	**47.50**(1.8)	**73.20**(0.4)	**49.07**(1.2)

(a) Classification accuracy with augmentation

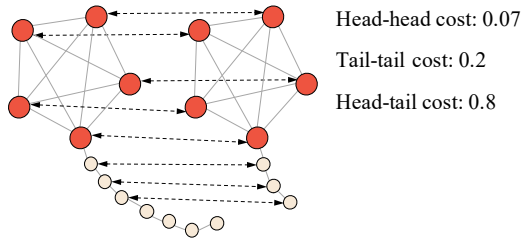
Method	NCI1 ↑	BZR↑	COX2↑	Mutagen.↑	PROTEINS↑	ENZYMES↑	IMDB-B↑	IMDB-M ↑
Unit cost	60.49(0.4)	72.20(3.7)	52.34(3.8)	56.69(2.0)	56.86(1.5)	**25.83**(3.0)	55.50(2.8)	36.33(2.2)
Feature-distance	60.70(0.8)	72.93(4.0)	52.55(3.2)	55.51(1.8)	**57.31**(1.7)	**25.83**(2.9)	56.00(2.8)	36.20(1.6)
EPIC	**72.69**(0.3)	**77.80**(2.9)	**60.43**(3.5)	**75.33**(1.0)	56.95(0.8)	23.67(4.7)	**67.70**(1.8)	**39.33**(2.8)

(b) Distance-based classification accuracy

Table 7: Comparisons between three different cost functions. Three different cost functions: unit, feature-distance, and EPIC, are tested on TUDataset. GIN is used as a backbone.



(a) Average substitution costs between negative pair



(b) Average substitution costs between positive pair

Figure 6: Analysis of substitution cost on *lollipop* graph. Normalized average substitution costs between source nodes and target nodes for *negative* pair are shown in Figure 6a and *positive* pair in Figure 6b. The costs are averaged across the entire dataset.

class. Table 7b shows the result of distance-based classification. In most cases, EPIC outperforms the other fixed-cost methods.

C Further Analysis on *Lollipop* Dataset

To further show the effectiveness of the cost function for EPIC, we analyze the cost for substitution to each node on the *lollipop* dataset. (m, n)-*lollipop* graph consists of a head, a complete graph with m nodes, and a tail, a chain structured n nodes, with varying m and n. We use the number of heads as the label of a graph, i.e., m. Figure 6 shows the head-to-head, head-to-tail, and tail-to-tail substitution costs. For instance, the head-to-head substitution cost measures the average substitution costs between the head nodes from the source graph with those of the target graph.

With the *negative pair*, the average node substitution cost for the head nodes is 0.97, whereas the average for the tail nodes is 0.04. These results align with our intuition that editing the head, closely related to the label, incurs a high cost. Thus, any modification to this part leads to substantial changes in the label. With the *positive pair*, the head-to-head and tail-to-tail substitution costs are 0.07 and 0.2, respectively. Those values are much smaller than 0.8 of head-tail costs. The lower cost for the head-head substitution in a positive pair resulted from the fact that it involves the exchange of nodes associated with the same label. Similarly, the low cost for the tail-to-tail substitution aligns well with our intuition since the nodes are unrelated to the label. In contrast, the head-to-tail substitution is assigned a high cost because it substitutes a label-associated part with a non-associated part. This preserves the head part of every graph on the edit path between *positive*-pair, ensuring that the augmented graph keeps the same label over the path.

Acknowledgements

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No. RS-2019-II191906: Artificial Intelligence Graduate School Program (POSTECH)), Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education (2022R1A6A1A03052954, 2022R1C1C1013366, RS-2024-00337955), and the Technology Innovation Program (No. 20014926, Development of BIT Convergent AI Architecture, Its Validation and Candidate Selection for COVID19 Antibody, Repositioning and Novel Synthetic Chemical Therapeutics) funded by the Ministry of Trans, Industry & Energy (MOTIE, Korea).

Contribution Statement

Jaeseung Heo and Seungbeom Lee contributed equally to the paper as the first authors.

References

[Abu-Aisheh *et al.*, 2015] Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. An exact graph edit distance algorithm for solving pattern recognition problems. In *4th International Conference on Pattern Recognition Applications and Methods 2015*, 2015.

[Airoldi *et al.*, 2013] Edo M Airoldi, Thiago B Costa, and Stanley H Chan. Stochastic blockmodel approximation of a graphon: Theory and consistent estimation. *Advances in Neural Information Processing Systems*, 26, 2013.

[Cortés *et al.*, 2019] Xavier Cortés, Donatello Conte, and Hubert Cardot. Learning edit cost estimation models for graph edit distance. *Pattern Recognition Letters*, 125:256–263, 2019.

[Cross *et al.*, 1997] Andrew DJ Cross, Richard C Wilson, and Edwin R Hancock. Inexact graph matching using genetic search. *Pattern Recognition*, 30(6):953–970, 1997.

[Durant *et al.*, 2002] Joseph L Durant, Burton A Leland, Douglas R Henry, and James G Nourse. Reoptimization of mdl keys for use in drug discovery. *Journal of chemical information and computer sciences*, 42(6):1273–1280, 2002.

[Feng *et al.*, 2020] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. Graph random neural networks for semi-supervised learning on graphs. *Advances in neural information processing systems*, 33:22092–22103, 2020.

[Fischer *et al.*, 2015] Andreas Fischer, Ching Y Suen, Volkmar Frinken, Kaspar Riesen, and Horst Bunke. Approximation of graph edit distance based on hausdorff matching. *Pattern Recognition*, 48(2):331–343, 2015.

[Han *et al.*, 2022] Xiaotian Han, Zhimeng Jiang, Ninghao Liu, and Xia Hu. G-mixup: Graph data augmentation for graph classification. In *International Conference on Machine Learning*, pages 8230–8248. PMLR, 2022.

[Hu *et al.*, 2020] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

[Iwana and Uchida, 2021] Brian Kenji Iwana and Seiichi Uchida. An empirical survey of data augmentation for time series classification with neural networks. *Plos one*, 16(7):e0254841, 2021.

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

[Kingma and Ba, 2014] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Kipf and Welling, 2016] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[Kong *et al.*, 2022] Kezhi Kong, Guohao Li, Mucong Ding, Zuxuan Wu, Chen Zhu, Bernard Ghanem, Gavin Taylor, and Tom Goldstein. Robust optimization as data augmentation for large-scale graphs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 60–69, 2022.

[Kuhn, 1955] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[Leskovec *et al.*, 2007] Jure Leskovec, Lada A Adamic, and Bernardo A Huberman. The dynamics of viral marketing. *ACM Transactions on the Web (TWEB)*, 1(1):5–es, 2007.

[Li *et al.*, 2019] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *International conference on machine learning*, pages 3835–3845. PMLR, 2019.

[Ling *et al.*, 2023] Hongyi Ling, Zhimeng Jiang, Meng Liu, Shuiwang Ji, and Na Zou. Graph mixup with soft alignments. *arXiv preprint arXiv:2306.06788*, 2023.

[Liu *et al.*, 2022] Gang Liu, Tong Zhao, Jiaxin Xu, Tengfei Luo, and Meng Jiang. Graph rationalization with environment-based augmentations. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1069–1078, 2022.

[Morris *et al.*, 2020] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020.

[Myers *et al.*, 2000] Richard Myers, RC Wison, and Edwin R Hancock. Bayesian graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):628–635, 2000.

[Park *et al.*, 2022] Joonhyung Park, Hajin Shim, and Eunho Yang. Graph transplant: Node saliency-guided graph mixup with local structure preservation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 7966–7974, 2022.

[Riba *et al.*, 2021] Pau Riba, Andreas Fischer, Josep Lladós, and Alicia Fornés. Learning graph edit distance by graph neural networks. *Pattern Recognition*, 120:108132, 2021.

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

[Rong *et al.*, 2019] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

[Schroff *et al.*, 2015] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[Shorten and Khoshgoftaar, 2019] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

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

[Ullmann, 1976] Julian R Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1):31–42, 1976.

[Verma *et al.*, 2019] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. In *International conference on machine learning*, pages 6438–6447. PMLR, 2019.

[Wang *et al.*, 2020] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, and Bryan Hooi. Graphcrop: Subgraph cropping for graph classification. *arXiv preprint arXiv:2009.10564*, 2020.

[Wang *et al.*, 2021a] Runzhong Wang, Tianqi Zhang, Tianshu Yu, Junchi Yan, and Xiaokang Yang. Combinatorial learning of graph edit distance via dynamic embedding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5241–5250, 2021.

[Wang *et al.*, 2021b] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, and Bryan Hooi. Mixup for node and graph classification. In *Proceedings of the Web Conference 2021*, pages 3663–3674, 2021.

[Wu *et al.*, 2018] Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

[Xu *et al.*, 2018] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[Yoo *et al.*, 2022] Jaemin Yoo, Sooyeon Shim, and U Kang. Model-agnostic augmentation for accurate graph classification. In *Proceedings of the ACM Web Conference 2022*, pages 1281–1291, 2022.

[You *et al.*, 2020] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in neural information processing systems*, 33:5812–5823, 2020.

[Zhang *et al.*, 2020] Shichang Zhang, Ziniu Hu, Arjun Subramonian, and Yizhou Sun. Motif-driven contrastive learning of graph representations. *arXiv preprint arXiv:2012.12533*, 2020.

[Zhou *et al.*, 2020] Jiajun Zhou, Jie Shen, and Qi Xuan. Data augmentation for graph classification. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 2341–2344, 2020.