Model-Agnostic Augmentation for Accurate Graph Classification

Jaemin Yoo
jaeminyoo@snu.ac.kr
Seoul National University
Seoul, South Korea

Sooyeon Shim
syshim77@snu.ac.kr
Seoul National University
Seoul, South Korea

U Kang
ukang@snu.ac.kr
Seoul National University
Seoul, South Korea

ABSTRACT

Given a graph dataset, how can we augment it for accurate graph classification? Graph augmentation is an essential strategy to improve the performance of graph-based tasks, and has been widely utilized for analyzing web and social graphs. However, previous works for graph augmentation either a) involve the target model in the process of augmentation, losing the generalizability to other tasks, or b) rely on simple heuristics that lead to unreliable results. In this work, we introduce five desired properties for effective augmentation. Then, we propose NodeSam (Node Split and Merge) and SubMix (Subgraph Mix), two model-agnostic algorithms for graph augmentation that satisfy all desired properties with different motivations. NodeSam makes a balanced change of the graph structure to minimize the risk of semantic change, while SubMix mixes random subgraphs of multiple graphs to create rich soft labels combining the evidence for different classes. Our experiments on social networks and molecular graphs show that NodeSam and SubMix outperform existing approaches in graph classification.

CCS CONCEPTS

• **Computing methodologies** → **Supervised learning by classification**; • **Information systems** → **Social networks**.

KEYWORDS

graph classification, data augmentation, model-agnostic methods

ACM Reference Format:
Jaemin Yoo, Sooyeon Shim, and U Kang. 2022. Model-Agnostic Augmentation for Accurate Graph Classification. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3485447.3512175

1 INTRODUCTION

How can we augment graphs for accurate graph classification? Data augmentation is an essential strategy to maximize the performance of estimators by enlarging the distribution covered by training data. The technique has been used widely in various data domains such as images [30], time series [39], and language processing [8]. The problem of graph augmentation has also attracted wide attention in the web domain [35, 36], where a community structure works as an essential evidence for classifying graph labels. An augmentation method provides rich variants of community structures, allowing one to understand the complex relationships between users.

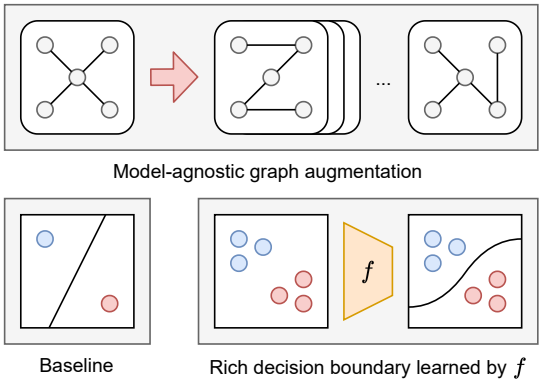


Figure 1: Illustration of how model-agnostic augmentation works. Each circle at the bottom represents a graph, whose label is represented as color. The augmented graphs allow a classifier f to learn a rich decision boundary.

Previous approaches on graph augmentation are categorized to model-specific [34, 53] and model-agnostic ones [35, 38, 55]. Model-specific approaches often make a better performance than model-agnostic ones, because they are designed for specific target models. However, their performance is not generalized to other settings of models and problems, and a careful tuning of hyperparameters is required even with a small change of experimental setups. On the other hand, model-agnostic approaches work generally well with various models and problems, even though their best performance can be worse than that of model-specific ones. Figure 1 illustrates how model-agnostic augmentation works for a graph classifier f that classifies each graph into the red or the blue class.

However, previous works on model-agnostic augmentation [35, 38, 55] rely on simple heuristics such as removing random edges or changing node attributes rather than carefully designed operations. Such heuristics provide no theoretical guarantee for essential properties of data augmentation such as the unbiasedness or linear scalability. As a result, previous approaches often make unreliable results, losing the main advantage over model-specific approaches. Moreover, our experiments on benchmark datasets show that they often decrease the accuracy of target models in graph classification, where the structural characteristic of each graph plays an essential role for predicting its label (details are in Section 4).

In this work, we first propose five properties that an augmentation algorithm should satisfy to maximize its effectiveness. These properties are designed carefully to include the degree of augmentation, the preservation of graph size, and the scalability to large

WWW '22, April 25–29, 2022, Virtual Event, Lyon, France

ACM ISBN 978-1-4503-9096-5/22/04…$15.00
https://doi.org/10.1145/3485447.3512175

Table 1: Comparison between various approaches for graph augmentation with respect to desired properties. P*i* refers to Property *i* (see Section 2.2). Our proposed methods satisfy all the desired properties, while the baselines do not.

Method	P1	P2	P3	P4	P5
DropEdge [28]				✓	✓
GraphCrop [35]			✓	✓	✓
NodeAug [38]			✓	✓	✓
MotifSwap [55]	✓	✓			
NodeSam (proposed)	✓	✓	✓	✓	✓
SubMix (proposed)	✓	✓	✓	✓	✓

graphs. We then propose two novel algorithms, NodeSam and SubMix, which satisfy all these properties. NodeSam (Node Split and Merge) performs split and merge operations on nodes, minimizing the degree of structural change while augmenting both the node- and edge-level information. SubMix (Subgraph Mix) combines multiple graphs by swapping random subgraphs to maximize the degree of augmentation, generating rich soft labels for classification.

Our contributions are summarized as follows:

- **Objective formulation.** We propose desired properties that are essential for effective graph augmentation, providing a clear objective for augmentation algorithms.
- **Algorithms.** We propose NodeSam and SubMix, effective model-agnostic algorithms for graph augmentation. NodeSam is a stable and balanced approach that makes a minimal change of the graph structure, while SubMix generates more diverse samples through abundant augmentation.
- **Theory.** We theoretically analyze the characteristics of our proposed approaches and demonstrate that they satisfy all the desired properties even in the worst cases.
- **Experiments.** We perform experiments on nine datasets to show the effectiveness of our methods for improving graph classifiers. Our methods make up to 2.1× larger improvement of accuracy compared with the best competitors.

The rest of this paper is organized as follows. In Section 2, we define the problem of graph augmentation and present the desired properties. In Section 3, we propose our NodeSam and SubMix and discuss their theoretical properties. We show experimental results in Section 4 and introduce related works in Section 5. We conclude in Section 6. All of our implementation and datasets are available at https://github.com/snudatalab/GraphAug.git.

2 PROBLEM AND DESIRED PROPERTIES

We formally define the graph augmentation problem and present desired properties for an effective augmentation algorithm. Table 1 compares various methods for graph augmentation regarding the desired properties that we propose in this section.

2.1 Problem Definition

Given a set of graphs, graph augmentation is to generate a new set of graphs that have similar characteristics to the given graphs. We give the formal definition as Problem 1.

Problem 1 (Graph augmentation). *We have a set $\mathcal{G}$ of graphs. Each graph $G \in \mathcal{G}$ consists of a set $\mathcal{V}$ of nodes, a set $\mathcal{E}$ of edges, and a feature matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$, where d is the number of features. Then, the problem is to make a set $\bar{\mathcal{G}}$ of new graphs that are more suitable than $\mathcal{G}$ for the training of a model f, improving its performance.*

Although any task can benefit from graph augmentation, we use graph classification as the target task to solve by a classifier f. This is because graph classification is more sensitive to the quality of augmentation than node-level tasks are, such as node classification or link prediction. In graph classification, naive augmentation can easily decrease the accuracy of f if it changes the characteristic of a graph that is essential for its classification (details in Section 4). On the other hand, in node-level tasks such as node classification, even a simple heuristic algorithm can improve the accuracy of models by changing the local neighborhood of each target node [28, 38]. Thus, the accuracy of graph classification is a suitable measure for comparing different approaches for graph augmentation.

2.2 Desired Properties

Our goal is to generate a set of augmented graphs that maximize the performance of a graph classifier f as presented in Problem 1. The main difficulty of augmentation is that the *semantic information* of a graph, which means the unique characteristic that determines its label, is not given clearly. For example, in the classification task of molecular graphs, it is difficult even for domain experts to check whether an augmented graph has the same chemical property as in the original graph. This makes it difficult for an augmentation algorithm to safely enlarge the data distribution.

We propose five desired properties for an effective augmentation algorithm to maximize the degree of augmentation while minimizing the risk of changing semantic information. Property 1 and 2 are for preserving the basic structural information of a graph in terms of the size and connectivity, respectively.

Property 1 (Preserving size). *Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ and an augmentation function h, let $\bar{G} = h(G)$. Then, h should make an unbiased change of the graph size by satisfying $\mathbb{E}[|\bar{\mathcal{V}}| - |\mathcal{V}|] = 0$ and $\mathbb{E}[|\bar{\mathcal{E}}| - |\mathcal{E}|] = 0$, where $\bar{G} = (\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathbf{X}})$.*

Property 2 (Preserving connectivity). *Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ and an augmentation function h, let $\bar{G} = h(G)$. Then, $\bar{G}$ should follow the connectivity information of G. In other words, $\bar{G}$ should be connected if and only if G is connected.*

At the same time, it is necessary for an augmentation algorithm to make meaningful changes to the given graph; Property 1 and 2 are satisfied even with the identity function. In this regard, we introduce Property 3 and 4 that force an augmentation algorithm to make node- and edge-level changes at the same time.

Property 3 (Changing nodes). *Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ and an augmentation function h, let $\bar{G} = h(G)$. Then, h should make a change of nodes in $\mathcal{V}$ by satisfying either $\mathbb{E}[(|\bar{\mathcal{V}}| - |\mathcal{V}|)^2] > 0$ or $\mathbb{E}[\|\bar{\mathbf{X}} - \mathbf{X}\|_F^2] > 0$, where $\bar{G} = (\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathbf{X}})$, and $\|\cdot\|_F$ is the Frobenius norm of a matrix.*

Property 4 (Changing edges). *Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ and an augmentation function h, let $\bar{G} = h(G)$. Then, h should make a change of $|\mathcal{E}|$, i.e., $\mathbb{E}[(|\bar{\mathcal{E}}| - |\mathcal{E}|)^2] > 0$, where $\bar{G} = (\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathbf{X}})$.*

Algorithm 1 NodeSam (Node Split and Merge)

Input: Target graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$
Output: Augmented graph $\bar{G} = (\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathbf{X}})$
1: $G', v_i, v_j, v_k \leftarrow \text{Split}(G)$ ▷ Algorithm 2
2: $G'' \leftarrow \text{Adjust}(G, G', v_i, v_j, v_k)$ ▷ Algorithm 4
3: $\bar{G} \leftarrow \text{Merge}(G'')$ ▷ Algorithm 3

Lastly, we require the augmentation to be done in linear time with the graph size to support scalability in large graphs, which is essential for real-world applications [12, 43].

Property 5 (Linear complexity). *Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ and an augmentation function h, let $\bar{G} = h(G)$. Then, the time and space complexities of h for generating $\bar{G}$ should be $O(d|\mathcal{V}| + |\mathcal{E}|)$, where d is the number of features, i.e., $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$.*

We briefly review the previous approaches in Table 1 in terms of the desired properties. DropEdge [28] and GraphCrop [35] make a subgraph of the given graph as a result. This makes a high risk of semantic change, as we have no clue for the essential part that determines the characteristic of the graph. NodeAug [38] also changes the graph properties by adding and removing edges near a random node. MotifSwap [55] preserves the properties of the given graph with respect to both nodes and edges, but fails to make a sufficient amount of change. Moreover, MotifSwap is not scalable to large graphs, as its running time is not linear with the number of edges due to the global enumeration to find all open triangles.

3 PROPOSED METHODS

In this work, we propose two effective algorithms for graph augmentation, which satisfy all the desired properties in Table 1. NodeSam (Node Split and Merge) performs opposite split and merge operations over nodes to make a balanced change of graph properties, while SubMix (Subgraph Mix) combines multiple graphs by mixing random subgraphs to enlarge the space of augmentation.

3.1 NodeSam

The main idea of NodeSam is to perform opposite operations at once to make a balanced change of the graph properties. NodeSam first splits a random node into a pair of adjacent nodes, increasing the number of nodes by one. NodeSam then merges a random pair of nodes into a single node, making the generated graph have the same number of nodes as in the original graph. The combination of these opposite operations allows it to change the node- and edge-level information at the same time while preserving the structure of the original graph such as the connectivity.

Algorithm 1 describes the process of NodeSam, where the split and merge operations are done at lines 1 and 3, respectively. The adjustment operation in line 2 is introduced to make an unbiased change of the number of edges by inserting additional edges to the graph. We first discuss the split and merge operations in detail and then present the adjustment operation in Section 3.1.1.

Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, NodeSam performs the split and merge operations following Algorithms 2 and 3, respectively. The split operation selects a random node v_i and splits it into nodes v_j and v_k, making an edge (v_j, v_k) and copying its feature as $\mathbf{x}_i = \mathbf{x}_j = \mathbf{x}_k$. The edges attached to v_i are split into v_j and v_k following the binomial distribution $\mathcal{B}(|\mathcal{N}_i|, 0.5)$, where $\mathcal{N}_i$ is the set of neighbors of v_i. The merge operation selects a random pair of adjacent nodes v_o and v_p and merges them into a new single node v_q with a feature vector $\mathbf{x}_q = (\mathbf{x}_o + \mathbf{x}_p)/2$. The edges connected to either v_o or v_p are connected to v_q, while the edge (v_o, v_p) is removed.

Algorithm 2 Split in NodeSam

Input: Target graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$
Output: Intermediate graph $G' = (\mathcal{V}', \mathcal{E}', \mathbf{X}')$, target node v_i, and generated nodes v_j and v_k
1: $v_i \leftarrow$ Select a node from $\mathcal{V}$ uniformly at random
2: $v_j, v_k \leftarrow$ Make new nodes to insert to G
3: $\mathbf{x}_j, \mathbf{x}_k \leftarrow$ Make new features such that $\mathbf{x}_i = \mathbf{x}_j = \mathbf{x}_k$
4: $h \leftarrow$ Make a function that randomly returns v_j or v_k
5: $\mathcal{V}' \leftarrow (\mathcal{V} \setminus \{v_i\}) \cup \{v_j, v_k\}$
6: $\mathcal{E}' \leftarrow \{(a, b) \mid (a, b) \in \mathcal{E} \wedge a \neq v_i \wedge b \neq v_i\} \cup \{(v_j, v_k)\}$
7: $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(h(a), b) \mid (a, b) \in \mathcal{E} \wedge a = v_i\}$
8: $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(a, h(b)) \mid (a, b) \in \mathcal{E} \wedge b = v_i\}$
9: $\mathbf{X}' \leftarrow$ Remove $\mathbf{x}_i$ from $\mathbf{X}$, and add $\mathbf{x}_j$ and $\mathbf{x}_k$ to it

Algorithm 3 Merge in NodeSam

Input: Graph $G'' = (\mathcal{V}'', \mathcal{E}'', \mathbf{X}'')$ generated from Adjust
Output: Augmented graph $\bar{G} = (\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathbf{X}})$
1: $v_o, v_p \leftarrow$ Select adjacent nodes from $\mathcal{V}''$ uniformly at random
2: $v_q \leftarrow$ Make a new node to insert to G''
3: $\mathbf{x}_q \leftarrow$ Make a new feature such that $\mathbf{x}_q = (\mathbf{x}_o + \mathbf{x}_p)/2$
4: $\bar{\mathcal{V}} \leftarrow (\mathcal{V}'' \setminus \{v_o, v_p\}) \cup \{v_q\}$
5: $\bar{\mathcal{E}} \leftarrow$ Remove all edges from $\mathcal{E}''$ connected to either v_o or v_p
6: $\bar{\mathcal{E}} \leftarrow \bar{\mathcal{E}} \cup \{(v_q, b) \mid (a, b) \in \mathcal{E}'' \wedge a \in \{v_o, v_p\}\}$
7: $\bar{\mathcal{E}} \leftarrow \bar{\mathcal{E}} \cup \{(a, v_q) \mid (a, b) \in \mathcal{E}'' \wedge b \in \{v_o, v_p\}\}$
8: $\bar{\mathbf{X}} \leftarrow$ Remove $\mathbf{x}_o$ and $\mathbf{x}_p$ from $\mathbf{X}''$, and add $\mathbf{x}_q$ to it

3.1.1 Adjustment Operation. The basic version of NodeSam with only the split and merge operations have two limitations. First, the split operation weakens the relationships between the nodes in $\mathcal{N}_i$. That is, the number of common neighbors between any two nodes in $\mathcal{N}_i$ is likely to decrease in the graph G' generated from the split. This happens if v_i forms triangles with its neighbors, since the split eliminates these triangles by changing them into loops of length four. Second, the number of edges tends to decrease in augmented graphs, since the merge operation can remove more than one edge. This happens if there are triangles containing both of the target nodes v_o and v_p, since the other two edges except (v_o, v_p) in each triangle are combined into a single one.

To address the two limitations, we propose an adjustment operation of Algorithm 4 that inserts additional edges to nodes v_j and v_k, which are generated from the split. First, we randomly select a subset $\mathcal{S}$ of nodes from $\mathcal{T}_i$, which is the set of all nodes that form triangles with v_i in the original graph G. Then, we add $|\mathcal{S}|$ edges to the graph by the following process: for each node $u \in \mathcal{S}$, we insert edge (u, s) to the graph, where s is v_j (or v_k) if u is connected with v_k (or v_j). Note that all nodes in $\mathcal{S}$ have an edge with either v_j or v_k before the adjustment since they are neighbors of v_i in G.

Figure 2 illustrates how NodeSam works in an example graph of seven nodes. NodeSam first splits a random node v_i into a pair of

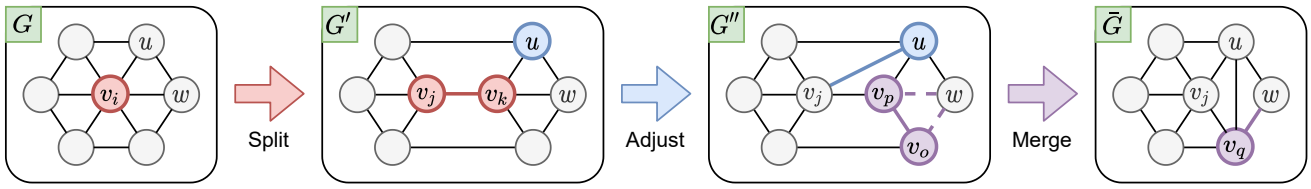


Figure 2: Illustration of how the three operations of NodeSam work in an example graph: Split, Adjust, and Merge. The adjustment operation preserves the numbers of edges and triangles of G by inserting an edge between nodes u and v_j.

Algorithm 4 Adjust in NodeSam

Input: Original graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$,
graph $G' = (\mathcal{V}', \mathcal{E}', \mathbf{X}')$ generated from Split,
target node $v_i \in \mathcal{V}$ of Split, and
nodes $v_j \in \mathcal{V}'$ and $v_k \in \mathcal{V}'$ generated from Split
Output: Intermediate graph $G'' = (\mathcal{V}'', \mathcal{E}'', \mathbf{X}'')$

1: $t_i \leftarrow$ Count the number of triangles in G containing v_i
2: $d_i \leftarrow$ Get the degree of v_i in G
3: $c_i \leftarrow |\mathcal{E}| - 3t_i/d_i - 2$
4: $h_i \leftarrow ((c_i^2 + 4t_i|\mathcal{V}| - 6t_i)^{1/2} - c_i)/2$
5: $b \leftarrow$ Make a function that returns a random value in $[0, 1)$
6: $\mathcal{T}_i \leftarrow$ Take all nodes included in the triangles containing v_i
7: $\mathcal{S} \leftarrow \{u \mid (u \in \mathcal{T}_i \setminus \{v_i\}) \wedge (b(u) < h_i/(|\mathcal{T}_i| - 1))\}$
8: $\mathcal{V}'', \mathbf{X}'' \leftarrow \mathcal{V}', \mathbf{X}'$
9: $\mathcal{E}'' \leftarrow \mathcal{E}' \cup \{(u, v_j) \mid u \in \mathcal{S}\} \cup \{(u, v_k) \mid u \in \mathcal{S}\}$

nodes v_j and v_k, decreasing the number of triangles from six to four. Then, the adjustment operation selects a random subset $\mathcal{S} = \{u\}$ of nodes from $\mathcal{T}_i$, which is $\mathcal{V} \setminus \{v_i\}$ in this example, and connects u with v_j. Lastly, the merge operation combines a random pair of nodes v_o and v_p into node v_q. Note that the numbers of edges and triangles of G are preserved in the augmented graph $\bar{G}$ even with a different structure due to edge (u, v_j) made by the adjustment.

The size of $\mathcal{S}$ is determined randomly in line 7 of Algorithm 4 following a binomial distribution whose mean is given as $\mathbb{E}[|\mathcal{S}|] = h_i$, where h_i is a number chosen to estimate the number of edges removed by the merge operation as follows:

$$h_i = \frac{1}{2}((c_i^2 + 4t_i|\mathcal{V}| - 6t_i)^{1/2} - c_i), \tag{1}$$

where t_i is the number of triangles containing v_i in G, d_i is the node degree of v_i in G, and $c_i = |\mathcal{E}| - 3t_i/d_i - 2$ (see Lemma 1).

Local estimation. All variables that compose h_i in Equation (1) are computed from the direct neighborhood of v_i, except for $|\mathcal{V}|$ and $|\mathcal{E}|$ that are known in advance. This allows NodeSam to be run in linear time with the number of edges, supporting scalability to large real-world graphs. At the same time, since the value of h_i is positively correlated with the number t_i of the triangles containing v_i, the adjustment effectively compensates for the triangles that are removed during the split; more triangles are likely to be removed with large t_i, but it tends to insert more edges in the adjustment. As a result, we address the two limitations of the naive version of NodeSam with a carefully chosen value of h_i.

3.1.2 Satisfying Desired Properties. NodeSam satisfies all the desired properties in Table 1. It is straightforward that Property 3 and 4 are satisfied since NodeSam changes the node features and the set of edges at every augmentation. We show in Lemma 1, 2, and 3 that NodeSam also satisfies the rest of the properties.

Lemma 1. *Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, let $\bar{G} = (\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathbf{X}})$ be the result of NodeSam. Then, $\mathbb{E}[|\bar{\mathcal{V}}| - |\mathcal{V}|] = 0$ and $\mathbb{E}[|\bar{\mathcal{E}}| - |\mathcal{E}|] = 0$.*

Proof. The proof is straightforward for $\mathcal{V}$, since the number of nodes does not change by NodeSam. The proof for $\mathcal{E}$ requires a series of estimations for the properties of intermediate graphs, and thus the full proof is given in Appendix A. The idea is that the expected number of edges removed by the merge operation is the same as h_i, which is the expected number of edges added by the adjustment operation as shown in line 4 of Algorithm 4. □

Lemma 2. *Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, let $\bar{G} = (\bar{\mathcal{V}}, \bar{\mathcal{E}}, \mathbf{X})$ be a graph generated by NodeSam. Then, $\bar{G}$ is connected if and only if G is connected.*

Proof. The proof is in Appendix B.1. □

Lemma 3. *Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, the time and space complexities of NodeSam are both $O(d|\mathcal{V}| + |\mathcal{E}|)$, where d is the number of features.*

Proof. The proof is in Appendix B.2. □

3.2 SubMix

SubMix aims to make a large degree of augmentation by combining multiple graphs. This is done by swapping subgraphs of different graphs, motivated by mixing approaches in the image domain [14, 50, 52]. The main idea is to treat the adjacency matrix of each graph like an image and replace a random patch of the matrix with that of another graph, as depicted in Figure 3 that compares our SubMix to CutMix [50] in the image domain. The red subgraph corresponds to the head of the fox in the augmented image.

The overall process of SubMix is shown as Algorithm 5. Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ and a set $\mathcal{G}$ of all available graphs, SubMix makes an augmented graph by replacing a subgraph of G with that of another graph G' chosen from $\mathcal{G} \setminus \{G\}$. First, SubMix samples ordered sets S and S' of nodes from G and G', respectively, where $|S| = |S'|$. Then, SubMix makes a one-to-one mapping ϕ from the nodes in S' to those in S based on their order in each ordered set, and uses it to transfer the induced subgraph of S' into G, replacing the induced subgraph of S. The edges that connect S to the rest of the graph G are then connected to the new nodes.

As a result, SubMix makes the set $\bar{\mathcal{E}} = \mathcal{E}_1 \cup \mathcal{E}_2$ of edges for the augmented graph $\bar{G}$, where $\mathcal{E}_1$ and $\mathcal{E}_2$ are extracted from G and G', respectively. The difference between $\mathcal{E}_1$ and $\mathcal{E}_2$ is that $\mathcal{E}_1$ contains the edges incident to at least one node in S, while $\mathcal{E}_2$ contains only the edges whose two connected nodes are both in S'. The reason is because the main target of augmentation is G. The subgraph of G'

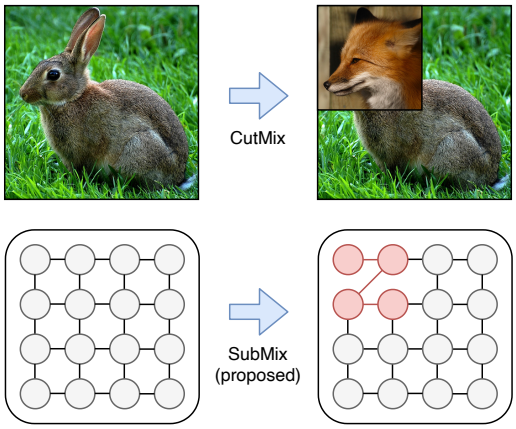


Figure 3: Comparison between image and graph mixing augmentation. SubMix, our proposed approach, generalizes CutMix [6] to the graph domain. The image patch in the top row corresponds to the red inserted subgraph in the bottom.

is inserted into G without changing the edges that connect S with $\mathcal{V} \setminus S$. See Figure 3 for a visual illustration.

One notable difference from NodeSam is that SubMix takes the label of G as an additional input and changes it. The label $\bar{\mathbf{y}}$ of $\bar{G}$ is softly determined as $\bar{\mathbf{y}} = q\mathbf{y} + (1-q)\mathbf{y}'$, where $\mathbf{y}$ and $\mathbf{y}'$ are the one-hot labels of G and G', respectively, and $q = |\mathcal{E}_1|/|\bar{\mathcal{E}}|$. This is based on an assumption that edges are crucial factors to determine the label of a graph, and thus the ratio of included edges quantifies how much it contributes to making $\bar{\mathbf{y}}$.

3.2.1 Selecting Subgraphs with Diffusion. The core part of SubMix is the choice of ordered sets S and S'. The list of nodes included in each set determines the shape of the subgraph, and their order determines how the nodes in S' are connected to the nodes in $\mathcal{V} \setminus S$. A naive approach is to select a random subset from each graph without considering the structural information, but this is likely to make disconnected subgraphs containing few edges.

Instead, we utilize graph diffusion to select connected and clustered subgraphs from G and G'. The process of subgraph sampling is shown as Algorithm 6. Given a root node r, a diffusion operator propagates a signal from r to all other nodes following the graph structure. This assigns large affinity scores to the nodes close to r or having many common neighbors with r. Thus, selecting the top k nodes having the largest affinity scores provides a meaningful substructure around r containing a sufficient number of edges, and the chosen nodes can be used directly as the subset S.

We use personalized PageRank (PPR) as the diffusion function, which is a popular approach to measure the personalized scores of nodes. PPR [17] converts the adjacency matrix $\mathbf{A}$ of each graph to a matrix $\mathbf{S}$ of scores by diffusing the graph signals from all nodes: $\mathbf{S} = \sum_{k=0}^{\infty} \alpha(1-\alpha)^k (\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})^k$, where $\mathbf{D}$ is the degree matrix such that $D_{ii} = \sum_k A_{ik}$, and $\alpha = 0.15$ is the teleport probability. The i-th column of $\mathbf{S}$ contains the affinity scores of nodes with respect to node i. Thus, given a root node r, we return the r-th column of $\mathbf{S}$ as the result $\mathbf{c}$ of diffusion in line 3 of Algorithm 6.

Connected subgraphs. In Algorithm 6, it is essential to guarantee the connectivity of nodes S and S' to allow SubMix to replace

Algorithm 5 SubMix (Subgraph Mix)

Input: Target graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ with its label y,
set $\mathcal{G}$ of all available graphs with labels $\mathcal{Y}$, and
target ratio $p \in (0, 1)$ of augmentation
Output: Augmented graph $\bar{G} = (\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathbf{X}})$ and its label $\bar{\mathbf{y}}$
1: $G', y' \leftarrow$ Pick a random graph from $\mathcal{G} \setminus \{G\}$ and its label
2: $S, S' \leftarrow \text{Sample}(G, G', p)$ ▷ Algorithm 6
3: $\phi \leftarrow$ Make the one-to-one mapping from S' to S
4: $\mathcal{E}_1 \leftarrow \{(u, v) \mid (u, v) \in \mathcal{E} \wedge \neg(u \in S \wedge v \in S)\}$
5: $\mathcal{E}_2 \leftarrow \{(\phi(u), \phi(v)) \mid (u, v) \in \mathcal{E}' \wedge (u \in S' \wedge v \in S')\}$
6: $\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathbf{X}} \leftarrow \mathcal{V}, \mathcal{E}_1 \cup \mathcal{E}_2, \mathbf{X}$
7: $\bar{\mathbf{X}}[\phi(S')] \leftarrow \mathbf{X}'[S']$ ▷ Replace a subset of features
8: $\mathbf{y}, \mathbf{y}' \leftarrow$ Represent y and y' as one-hot vectors, respectively
9: $\bar{\mathbf{y}} \leftarrow (|\mathcal{E}_1|/|\bar{\mathcal{E}}|)\mathbf{y} + (1 - |\mathcal{E}_1|/|\bar{\mathcal{E}}|)\mathbf{y}'$

Algorithm 6 Sample in SubMix

Input: Target graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$,
another graph $G' = (\mathcal{V}', \mathcal{E}', \mathbf{X}')$ selected from $\mathcal{G} \setminus \{G\}$, and
target ratio $p \in (0, 1)$ of augmentation
Output: Ordered sets $S \subseteq \mathcal{V}$ and $S' \subseteq \mathcal{V}'$ of connected nodes
1: $r, r' \leftarrow$ Pick random nodes from G and G', respectively
2: $\psi \leftarrow$ Make a function that finds the connected component
3: $k \leftarrow \text{uniform}(0, p) \cdot \min(|\psi(G, r)|, |\psi(G', r')|)$
4: **for** $(G_t, r_t) \in \{(G, r), (G', r')\}$ **do**
5: $\quad \mathbf{c}_t \leftarrow$ Compute the scores of nodes by $\text{Diffuse}(G_t, r_t)$
6: $\quad S_t \leftarrow$ Select k nodes having the largest scores in $\mathbf{c}_t$
7: $\quad$ **if** S_t contains a disconnected node **then**
8: $\quad\quad S_t \leftarrow$ Take the first k nodes from $\text{BFS}(G_t, r_t)$
9: $\quad$ **end if**
10: $\quad S \leftarrow S_t$ if $G_t = G$ otherwise $S' \leftarrow S_t$
11: **end for**

meaningful substructures. We guarantee the connectivity with two ideas in the algorithm. First, we bound the number of selected nodes by the size of the connected component containing each root node, since the input graphs can be disconnected. Second, if the nodes selected by PPR make disconnected subgraphs, we resample the nodes by the breadth-first search (BFS) that is guaranteed to make connected subgraphs. This is to prevent rare cases where PPR returns a disconnected graph, which has not occurred in all of our experiments but can happen theoretically.

Lemma 4. *Each set of nodes returned by Algorithm 6 contains only connected nodes for both G and G' and for any value of k.*

Proof. The proof is straightforward as we run BFS on the connected component of each root if S (or S') is not connected. □

Order of nodes. The order of selected nodes determines how the nodes in S are matched with those in S', playing an essential role for the result of augmentation. One advantage of diffusion is that the selected nodes are ordered by their affinity scores, making nodes at the same relative position around the root nodes r and r' to be matched between S and S' in the replacement. The root r of S is always replaced with r' of S', and the replacement of all remaining nodes is determined by their affinity scores. This makes

the generated graph $\bar{G}$ more plausible than in the naive approach that matches the nodes in S randomly with those in S'.

3.2.2 Satisfying Desired Properties. SubMix satisfies all the desired properties in Table 1. It is straightforward that Property 3 and 4 are satisfied since SubMix changes the node features and the set of edges at every augmentation. We show in Lemma 1, 2, and 3 that SubMix also satisfies the rest of the properties.

Preserving size. Since SubMix combines multiple graphs, it is not possible to directly show the satisfaction of Property 1 for any given graph G. For instance, the number of edges always increases if G is a chain graph and all other graphs in $\mathcal{G}$ are cliques. Thus, we assume that the target graph G is selected uniformly at random from $\mathcal{G}$ and prove the unbiasedness with $\mathbb{E}[|\mathcal{E}| - |\bar{\mathcal{E}}|] = 0$.

Lemma 5. *Given a set $\mathcal{G}$ of graphs, we sample different graphs $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ and $G' = (\mathcal{V}', \mathcal{E}', \mathbf{X}')$ from $\mathcal{G}$ uniformly at random. Let $\bar{G} = (\bar{\mathcal{V}}, \bar{\mathcal{E}})$ be an augmented graph generated by SubMix. Then, $\mathbb{E}[|\mathcal{V}| - |\bar{\mathcal{V}}|] = 0$ and $\mathbb{E}[|\mathcal{E}| - |\bar{\mathcal{E}}|] = 0$.*

Proof. The proof is in Appendix B.3. □

Other properties. Lemma 6 shows that SubMix preserves the connectivity of the given graph, while Lemma 7 shows that SubMix runs in linear time with respect to the number of edges.

Lemma 6. *Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, let $\bar{G} = (\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathbf{X}})$ be an augmented graph generated by SubMix. Then, $\bar{G}$ is connected if and only if G is connected.*

Proof. The proof is in Appendix B.4. □

Lemma 7. *Given graphs $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ and $G' = (\mathcal{V}', \mathcal{E}', \mathbf{X}')$, the time and space complexities of SubMix are $O(pd|\mathcal{V}| + |\mathcal{E}| + |\mathcal{E}'|)$, where p is the ratio of sampling, and d is the number of features.*

Proof. The proof is in Appendix B.5. □

4 EXPERIMENTS

We perform experiments to answer the following questions:

- Q1. **Accuracy (Section 4.2).** Do NodeSam and SubMix improve the accuracy of graph classifiers? Are they better than previous approaches for graph augmentation?
- Q2. **Desired properties (Section 4.3).** Do NodeSam and SubMix satisfy the desired properties of Table 1 in real-world graphs as we claim theoretically in Section 3?
- Q3. **Ablation study (Section 4.4).** Do our ideas for improving NodeSam and SubMix, such as the adjustment or diffusion operation, increase the accuracy of graph classifiers?

4.1 Experimental Setup

We introduce our experimental setup including datasets, baseline approaches, hyperparameters, and graph classifiers.

Datasets. We use 9 benchmark datasets [23, 41] summarized in Table 2, which were used in previous works for graph classification. D&D, ENZYMES, MUTAG, NCI1, NCI109, PROTEINS, and PTC-MR [42] are datasets of molecular graphs that represent chemical compounds. COLLAB [42] and Twitter [25] are datasets of social networks. The numbers of nodes and edges in Table 2 are from all the graphs in each dataset. Detailed information of node features in the datasets are described in Appendix C.

Table 2: Summary of datasets.

Dataset	Graphs	Nodes	Edges	Features	Labels
D&D[1]	1,178	334,925	843,046	89	2
ENZYMES[1]	600	19,580	37,282	3	6
MUTAG[1]	188	3,371	3,721	7	2
NCI1[1]	4,110	122,747	132,753	37	2
NCI109[1]	4,127	122,494	132,604	38	2
PROTEINS[1]	1,113	43,471	81,044	3	2
PTC-MR[1]	344	4,915	5,054	18	2
COLLAB[1]	5,000	372,474	12,286,079	369	3
Twitter[1]	144,033	580,768	717,558	1,323	2

[1] https://chrsmrrs.github.io/datasets

Baselines. We consider the following baselines for graph augmentation. DropEdge [28] removes an edge uniformly at random, while DropNode removes a node and all connected edges. AddEdge inserts an edge between a random pair of nodes. ChangeAttr augments the one-hot feature vector of a random node by changing the nonzero index. GraphCrop [35] selects a subgraph by diffusion and uses it as an augmented graph. NodeAug [38] combines ChangeAttr, DropEdge, and AddEdge to change the local neighborhood of a random target node. MotifSwap [55] swaps the two edges in an open triangle to preserve the connectivity during augmentation.

Classifier. We use GIN [41] as a graph classifier for evaluating the accuracy, which is one of the most popular models for graph classification and shows great performance in many domains. The hyperparameters are searched in the same space as in their original paper [41] to ensure that the improvement of accuracy comes from the augmentation, not from hyperparameter tuning: batch size in $\{32, 128\}$ and the dropout probability in $\{0, 0.5\}$.

Training details. Following the experimental process of GIN [41], which we use as the classifier, we run 10-fold cross-validation for evaluation. The indices of chosen graphs for each fold are included in the provided code repository. The Adam optimizer [15] is used, and the learning rate starts from 0.01 and decreases by half at every 50 epochs until it reaches 350 epochs. We set the ratio p of augmentation for SubMix, which is the only hyperparameter of our proposed approaches, to 0.4. All of our experiments were done at a workstation with Intel Core i7-8700 and RTX 2080.

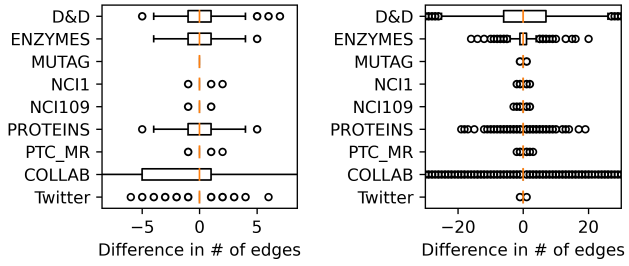
4.2 Accuracy of Graph Classification (Q1)

Table 3 compares the accuracy of graph classification with various augmentation methods. The Rank column is made by getting the ranks of methods in each dataset and computing their average and standard deviation over all datasets. For example, if the rank of a method is 1, 3, and 3 in three datasets, respectively, its rank is reported as 2.3 ± 1.2. The rank measures the performance of each method differently from the average accuracy: one can achieve the highest rank even though its average accuracy is low.

NodeSam and SubMix improve the average accuracy of GIN by 1.71 and 1.75 percent points, which are 2.0× and 2.1× larger than the improvement of the best competitor, respectively. Note that all approaches except NodeSam and SubMix decrease the accuracy of GIN in some datasets; GraphCrop even decreases the average accuracy of GIN, implying that they give a wrong bias to the classifier by distorting the data distribution with unjustified changes.

Table 3: Accuracy of graph classification with various graph augmentation methods. The values in parentheses are the ranks of methods in each dataset, and the Rank column shows the average and standard deviation of ranks over all datasets. The proposed methods NodeSam and SubMix achieve the best average accuracy and the highest ranks at the same time.

Method	D&D	ENZY.	MUTAG	NCI1	N109	PROT.	PTC-MR	COLLAB	Twitter	Average	Rank
Baseline	76.40 (4)	50.33 (10)	89.94 (4)	82.68 (9)	81.80 (9)	75.38 (9)	63.94 (7)	82.66 (7)	66.05 (7)	74.35 (8)	7.33 ± 2.18
GraphCrop	77.08 (2)	51.00 (9)	77.11 (10)	80.46 (10)	79.77 (10)	75.20 (10)	61.87 (10)	83.50 (2)	66.15 (3)	72.46 (10)	7.33 ± 3.77
DropEdge	76.14 (6)	53.67 (6)	81.93 (9)	82.82 (7)	82.60 (7)	75.74 (4)	63.68 (8)	82.50 (9)	66.05 (8)	73.90 (9)	7.11 ± 1.62
NodeAug	76.14 (8)	54.67 (5)	86.14 (7)	83.16 (4)	82.36 (8)	75.56 (6)	66.24 (2)	81.32 (10)	65.98 (9)	74.62 (7)	6.56 ± 2.55
AddEdge	76.14 (7)	55.17 (3)	85.67 (8)	83.99 (2)	83.06 (5)	75.38 (8)	64.27 (5)	82.80 (5)	66.10 (6)	74.73 (6)	5.44 ± 2.07
ChangeAttr	75.72 (9)	53.33 (8)	90.44 (3)	83.02 (5)	83.57 (2)	75.47 (7)	62.47 (9)	82.76 (6)	66.36 (2)	74.79 (5)	5.67 ± 2.83
DropNode	75.55 (10)	55.17 (3)	87.28 (6)	82.85 (6)	83.04 (6)	75.65 (5)	**66.59 (1)**	82.54 (8)	66.11 (5)	74.97 (4)	5.56 ± 2.60
MotifSwap	76.23 (5)	53.50 (7)	90.47 (2)	82.82 (7)	83.28 (4)	75.92 (3)	65.79 (3)	82.84 (3)	65.93 (10)	75.20 (3)	4.89 ± 2.62
SubMix	**78.10 (1)**	57.50 (2)	89.94 (4)	**84.33 (1)**	**84.37 (1)**	**76.19 (1)**	63.97 (6)	**83.74 (1)**	**66.44 (1)**	76.06 (2)	**2.00 ± 1.80**
NodeSam	76.57 (3)	**60.00 (1)**	**90.96 (1)**	83.33 (3)	83.52 (3)	76.10 (2)	65.48 (4)	82.82 (4)	66.13 (4)	**76.10 (1)**	2.78 ± 1.20



(a) NodeSam **(b) SubMix**

Figure 4: Box plots representing the number of edges that our proposed methods change through augmentation. Both methods make unbiased augmentation, where the variance of changes depends on the characteristic of each dataset.

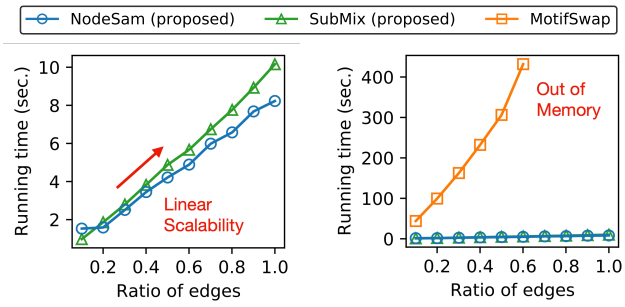


Figure 5: Computational time of our proposed methods and MotifSwap with respect to the number of edges. Our methods achieve linear scalability, while MotifSwap does not.

4.3 Preserving Desired Properties (Q2)

The main motivation of our NodeSam and SubMix is to satisfy the desired properties of Table 1. We present empirical results that support our theoretical claims given in Section 3.1.2 and 3.2.2.

Figure 4 visualizes the change in the number of edges during augmentation as box plots. Both NodeSam and SubMix perform unbiased augmentation as shown in the orange lines that appear in the center of each plot. On the other hand, they make a sufficient change of edges through augmentation, generating diverse examples for the training of a classifier. SubMix makes a larger degree of augmentation than NodeSam, especially in the D&D dataset, as it combines multiple graphs of different structures.

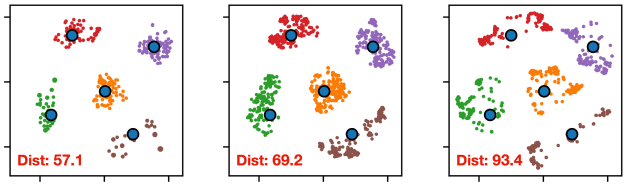
Figure 5 shows the scalability of NodeSam and SubMix with the number of edges in a graph. We use the Reddit [51] dataset for this experiment, which is large enough to cause scalability issues as it contains 232,965 nodes and 11,606,919 edges. We randomly make nine subgraphs with different sizes to measure the running time of methods. The figure shows that NodeSam and SubMix have linear scalability with similar patterns, while the computational time of MotifSwap increases much faster, causing out-of-memory errors. This supports our claims in Lemma 3 and 7.

Figure 6 visualizes the space of augmentation for our proposed algorithms and MotifSwap, the strongest baseline. We first use the Weisfeiler-Lehman (WL) kernel [29] to extract embedding vectors of graphs and then use the t-SNE algorithm [33] to visualize them in the 2D space. The distance value in each figure represents the average distance between each original graph and its augmented graphs in the 2D space. SubMix shows the largest space of augmentation by combining multiple graphs, effectively filling in the space between given examples. MotifSwap makes augmented graphs only near the original graphs, making the smallest average distance.

4.4 Ablation Study (Q3)

We perform an ablation study for both NodeSam and SubMix, and present the result in Figure 7. SplitOnly and MergeOnly refer to the split and merge operations of Algorithm 2 and 3, respectively. NodeSamBase refers to the naive version of NodeSam, which does not perform the adjustment operation of Algorithm 4. SubMixBase refers to the naive version of SubMix, which selects the target sets of nodes for the replacement uniformly at random, without using the diffusion operation of Algorithm 6. The vertical and horizontal axes of the figure are the average accuracy and rank over all datasets, corresponding to the last two columns of Table 3, respectively.

We have two observations from Figure 7. First, NodeSam and SubMix with all techniques achieve the best accuracy among all versions, showing that our proposed techniques are effective for improving their performance by satisfying the desired properties. Second, every basic version of our approaches still improves the



(a) MotifSwap (b) NodeSam (c) SubMix

Figure 6: Space of augmentation for various algorithms. The large blue points are original graphs, while the small points represent augmented graphs. SubMix has the largest space of augmentation, while MotifSwap has the smallest one.

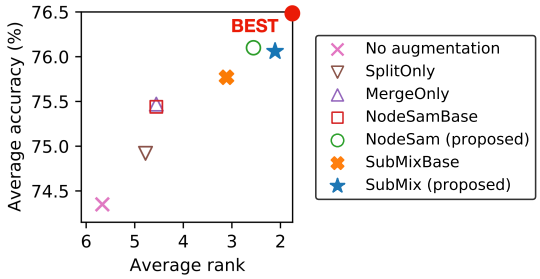


Figure 7: Ablation study for our NodeSam and SubMix. Both of them make the best average accuracy and rank based on our proposed ideas (details in Section 4.4).

baseline significantly, which is to use raw graphs without augmentation. This is clear when compared with existing methods shown in Table 3 that make marginal improvements or even decrease the baseline accuracy. Such improvement exhibits the effectiveness of our ideas, which are building blocks of NodeSam and SubMix.

We also perform an additional study for NodeSam and its basic versions in Appendix D, which shows that NodeSamBase without the adjustment operation is likely to decrease the number of edges due to the merge operation that eliminates additional triangles.

5 RELATED WORKS

We review related works for graph augmentation, graph classification, and image augmentation.

Model-agnostic graph augmentation. Our work focuses on model-agnostic augmentation, which is to perform augmentation independently from target models. Previous works can be categorized by the purpose of augmentation: node-level tasks [28, 38, 46], graph-level tasks [10, 35, 55], or graph contrastive learning [48, 49]. Such methods rely on heuristic operations that make no theoretical guarantee for the degree of augmentation or the preservation of graph properties. We propose five desired properties for effective graph augmentation and show that our approaches satisfy all of them, resulting in the best accuracy in classification.

Model-specific graph augmentation. Model-specific methods for graph augmentation make changes optimized for the target model. For example, changing the representation vectors of graphs learned by the target classifier by a mixup algorithm [37] is model-specific. Such model-specific approaches include manifold mixup [34], dynamic augmentation of edges [3, 53], and adversarial perturbations [7, 18]. Methods for graph adversarial training [4, 20, 40, 57] also belong to this category as they require the predictions of target models. Such approaches participate directly in the training process, and thus cannot be used generally in various settings.

Graph classification. Graph classification is a core problem in the graph domain, which is to predict the label of each graph. Compared to node classification [16] that focuses on the properties of individual nodes, graph classification aims at extracting meaningful substructures of nodes to make a better representation of a graph. A traditional way to solve the problem is to utilize kernel methods [24, 26, 29, 42], while graph neural networks [13, 16, 17, 41, 45, 47] have shown a better performance with advanced pooling methods [1, 21, 44] to aggregate node embeddings. We use graph classification as a downstream task for evaluating augmented graphs, since accurate classification requires separating the core information of each graph from random and unimportant components.

Image augmentation. Data augmentation is studied actively in the image domain. Basic approaches include color and geometric transformations [11, 27], masking [5, 31, 54], adversarial perturbations [22], pixel-based mixing [19, 52], and generative models [9, 56]. Such approaches utilize the characteristic of images where slight changes of pixel values do not change the semantic information of each image. On the other hand, patch-based mixing approaches that replace a random patch of an image to that of another image [14, 32, 50] give us a motivation to propose SubMix, which replaces an induced subgraph of a graph, instead of an image patch. This is based on the idea that images are grid-structured graphs whose pixels correspond to individual nodes.

6 CONCLUSION

We propose NodeSam (Node Split and Merge) and SubMix (Subgraph Mix), effective algorithms for model-agnostic graph augmentation. We first present desired properties for graph augmentation including the preservation of original properties, the guarantee of sufficient augmentation, and the linear scalability. Then, we show that both NodeSam and SubMix achieve all properties unlike all of the existing approaches. NodeSam aims to make a minimal change of the graph structure by performing opposite operations at once: splitting a node and merging two nodes. On the other hand, SubMix aims to increase the degree of augmentation by combining random subgraphs of different graphs with different labels. Experiments on nine datasets show that NodeSam and SubMix achieve the best accuracy in graph classification with up to 2.1× larger improvement of accuracy compared with previous approaches.

ACKNOWLEDGMENTS

This work was supported by Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) [No.2020-0-00894, Flexible and Efficient Model Compression Method for Various Applications and Environments], [No.2021-0-01343, Artificial Intelligence Graduate School Program (Seoul National University)], and [NO.2021-0-0268, Artificial Intelligence Innovation Hub (Artificial Intelligence Institute, Seoul National University)]. The Institute of Engineering Research at Seoul National University provided research facilities for this work. The ICT at Seoul National University provides research facilities for this study. U Kang is the corresponding author.

REFERENCES

[1] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. 2020. Spectral Clustering with Graph Neural Networks for Graph Pooling. In *ICML*.

[2] Ronald V. Book. 1974. Comparing Complexity Classes. *J. Comput. Syst. Sci.* 9, 2 (1974), 213–229. https://doi.org/10.1016/S0022-0000(74)80008-5

[3] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2020. Measuring and Relieving the Over-Smoothing Problem for Graph Neural Networks from the Topological View. In *AAAI*.

[4] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2018. Adversarial Attack on Graph Structured Data. In *ICML*.

[5] Terrance Devries and Graham W. Taylor. 2017. Improved Regularization of Convolutional Neural Networks with Cutout. *CoRR* abs/1708.04552 (2017). arXiv:1708.04552

[6] Boxin Du and Hanghang Tong. 2019. MrMine: Multi-resolution Multi-network Embedding. In *CIKM*.

[7] Fuli Feng, Xiangnan He, Jie Tang, and Tat-Seng Chua. 2019. Graph Adversarial Training: Dynamically Regularizing Based on Graph Structure. *CoRR* abs/1902.08226 (2019). arXiv:1902.08226

[8] Steven Y. Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard H. Hovy. 2021. A Survey of Data Augmentation Approaches for NLP. In *Findings of ACL*.

[9] Maayan Frid-Adar, Idit Diamant, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. 2018. GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification. *Neurocomputing* 321 (2018), 321–331.

[10] Kun Fu, Tingyun Mao, Yang Wang, Daoyu Lin, Yuanben Zhang, Junjian Zhan, Xian Sun, and Feng Li. 2021. TS-Extractor: large graph exploration via subgraph extraction based on topological and semantic information. *J. Vis.* 24, 1 (2021), 173–190.

[11] Adrian Galdran, Aitor Alvarez-Gila, Maria Inês Meyer, Cristina López Saratxaga, Teresa Araujo, Estíbaliz Garrote, Guilherme Aresta, Pedro Costa, Ana Maria Mendonça, and Aurélio J. C. Campilho. 2017. Data-Driven Color Augmentation Techniques for Deep Skin Image Analysis. *CoRR* abs/1703.03702 (2017). arXiv:1703.03702

[12] Saehan Jo, Jaemin Yoo, and U Kang. 2018. Fast and Scalable Distributed Loopy Belief Propagation on Real-World Graphs. In *WSDM*.

[13] Jinhong Jung, Jaemin Yoo, and U Kang. 2020. Signed Graph Diffusion Network. *CoRR* abs/2012.14191 (2020). arXiv:2012.14191

[14] Jang-Hyun Kim, Wonho Choo, and Hyun Oh Song. 2020. Puzzle Mix: Exploiting Saliency and Local Statistics for Optimal Mixup. In *ICML*.

[15] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

[16] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[17] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion Improves Graph Learning. In *NeurIPS*.

[18] Kezhi Kong, Guohao Li, Mucong Ding, Zuxuan Wu, Chen Zhu, Bernard Ghanem, Gavin Taylor, and Tom Goldstein. 2020. FLAG: Adversarial Data Augmentation for Graph Neural Networks. *CoRR* abs/2010.09891 (2020). arXiv:2010.09891

[19] Jin-Ha Lee, Muhammad Zaigham Zaheer, Marcella Astrid, and Seung-Ik Lee. 2020. SmoothMix: a Simple Yet Effective Data Augmentation to Train Robust Classifiers. In *CVPR*.

[20] Jiaqi Ma, Shuangrui Ding, and Qiaozhu Mei. 2020. Towards More Practical Adversarial Attacks on Graph Neural Networks. In *NeurIPS*.

[21] Yao Ma, Suhang Wang, Charu C. Aggarwal, and Jiliang Tang. 2019. Graph Convolutional Networks with EigenPooling. In *KDD*.

[22] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. In *CVPR*.

[23] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. 2020. TUDataset: A collection of benchmark datasets for learning with graphs. *CoRR* abs/2007.08663 (2020). arXiv:2007.08663

[24] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, Yang Liu, and Santhoshkumar Saminathan. 2016. subgraph2vec: Learning Distributed Representations of Rooted Sub-graphs from Large Graphs. *CoRR* abs/1606.08928 (2016). arXiv:1606.08928

[25] Shirui Pan, Jia Wu, and Xingquan Zhu. 2015. CogBoost: Boosting for Fast Cost-Sensitive Graph Classification. *IEEE Trans. Knowl. Data Eng.* 27, 11 (2015), 2933–2946. https://doi.org/10.1109/TKDE.2015.2391115

[26] Bastian Rieck, Christian Bock, and Karsten M. Borgwardt. 2019. A Persistent Weisfeiler-Lehman Procedure for Graph Classification. In *ICML*.

[27] Ignacio Rocco, Relja Arandjelovic, and Josef Sivic. 2019. Convolutional Neural Network Architecture for Geometric Matching. *IEEE Trans. Pattern Anal. Mach. Intell.* 41, 11 (2019).

[28] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2020. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *ICLR*.

[29] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. 2011. Weisfeiler-Lehman Graph Kernels. *J. Mach. Learn. Res.* 12 (2011).

[30] Connor Shorten and Taghi M. Khoshgoftaar. 2019. A survey on Image Data Augmentation for Deep Learning. *J. Big Data* 6 (2019), 60.

[31] Krishna Kumar Singh and Yong Jae Lee. 2017. Hide-and-Seek: Forcing a Network to be Meticulous for Weakly-Supervised Object and Action Localization. In *ICCV*.

[32] Ryo Takahashi, Takashi Matsubara, and Kuniaki Uehara. 2020. Data Augmentation Using Random Image Cropping and Patching for Deep CNNs. *IEEE Trans. Circuits Syst. Video Technol.* 30, 9 (2020).

[33] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008).

[34] Vikas Verma, Meng Qu, Alex Lamb, Yoshua Bengio, Juho Kannala, and Jian Tang. 2019. GraphMix: Regularized Training of Graph Neural Networks for Semi-Supervised Learning. *CoRR* abs/1909.11715 (2019). arXiv:1909.11715

[35] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, and Bryan Hooi. 2020. GraphCrop: Subgraph Cropping for Graph Classification. *CoRR* abs/2009.10564 (2020). arXiv:2009.10564

[36] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, and Bryan Hooi. 2021. CurGraph: Curriculum Learning for Graph Classification. In *WWW*.

[37] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, and Bryan Hooi. 2021. Mixup for Node and Graph Classification. In *WWW*.

[38] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, Juncheng Liu, and Bryan Hooi. 2020. NodeAug: Semi-Supervised Node Classification with Data Augmentation. In *KDD*.

[39] Qingsong Wen, Liang Sun, Fan Yang, Xiaomin Song, Jingkun Gao, Xue Wang, and Huan Xu. 2021. Time Series Data Augmentation for Deep Learning: A Survey. In *IJCAI*.

[40] Kaidi Xu, Hongge Chen, Sijia Liu, Pin-Yu Chen, Tsui-Wei Weng, Mingyi Hong, and Xue Lin. 2019. Topology Attack and Defense for Graph Neural Networks: An Optimization Perspective. In *IJCAI*.

[41] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICLR*.

[42] Pinar Yanardag and S. V. N. Vishwanathan. 2015. Deep Graph Kernels. In *KDD*.

[43] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *KDD*.

[44] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. In *NeurIPS*.

[45] Jaemin Yoo, Hyunsik Jeon, and U Kang. 2019. Belief Propagation Network for Hard Inductive Semi-Supervised Learning. In *IJCAI*.

[46] Jaemin Yoo, U Kang, Mauro Scanagatta, Giorgio Corani, and Marco Zaffalon. 2020. Sampling Subgraphs with Guaranteed Treewidth for Accurate and Efficient Graphical Inference. In *WSDM*.

[47] Jaemin Yoo, Junghun Kim, Hoyoung Yoon, Geonsoo Kim, Changwon Jang, and U Kang. 2021. Accurate Graph-Based PU Learning without Class Prior. In *ICDM*.

[48] Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. 2021. Graph Contrastive Learning Automated. In *ICML*.

[49] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph Contrastive Learning with Augmentations. In *NeurIPS*.

[50] Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Seong Joon Oh, Youngjoon Yoo, and Junsuk Choe. 2019. CutMix: Regularization Strategy to Train Strong Classifiers With Localizable Features. In *ICCV*.

[51] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *ICLR*.

[52] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. 2018. mixup: Beyond Empirical Risk Minimization. In *ICLR*.

[53] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver J. Woodford, Meng Jiang, and Neil Shah. 2020. Data Augmentation for Graph Neural Networks. *CoRR* abs/2006.06830 (2020). arXiv:2006.06830

[54] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. 2020. Random Erasing Data Augmentation. In *AAAI*.

[55] Jiajun Zhou, Jie Shen, and Qi Xuan. 2020. Data Augmentation for Graph Classification. In *CIKM*.

[56] Xinyue Zhu, Yifan Liu, Jiahong Li, Tao Wan, and Zengchang Qin. 2018. Emotion Classification with Data Augmentation Using Generative Adversarial Networks. In *PAKDD*.

[57] Daniel Zügner and Stephan Günnemann. 2019. Adversarial Attacks on Graph Neural Networks via Meta Learning. In *ICLR*.

A UNBIASEDNESS OF NODESAM

A.1 Properties of Intermediate Graphs

We have four graphs G, G', G'', and $\bar{G}$ presented in Algorithm 1, which denote the original graph, the graph after the split, the graph after the adjustment, and the final graph generated by NodeSam, respectively. Let v_i be the target node of the split operation, d_i be the degree of v_i in G, and h_i be the *expected* number of edges added in the adjustment operation. Let $T(\cdot)$ be the function that counts the number of triangles in a graph. Then, we analyze the properties of graphs generated during NodeSam in Lemma 8 to 11.

Lemma 8. $\mathbb{E}[T(G')] = T(G) - t_i/2$.

Proof. Each triangle in G containing v_i has a probability of 0.5 to be removed in the split operation, due to the random selection of the target node from v_j and v_k. Thus, it is expected that the half of all triangles containing v_i are removed in G'. □

Lemma 9. *In the adjustment operation of Algorithm 4, let l be the number of new triangles created in G'' by connecting an edge between a target node $u \in \mathcal{S}$ and either v_j or v_k. Then, $\mathbb{E}[l] = |\mathcal{N}_{ui}|/2 + 1$, where $\mathcal{N}_{ui}$ is the set of common neighbors between u and v_i in G.*

Proof. Node u is connected to either v_j or v_k in G', since it is a neighbor of v_i in the the original graph G before the split is done. Assume that u is connected to v_j in G' without loss of generality. Then, the adjustment for u makes an edge (u, v_k). A single triangle (u, v_j, v_k) is always created by this operation, since edges (u, v_j) and (v_j, v_k) already exist in G'. The number of additional triangles created by the adjustment is the same as the number of common neighbors between u and v_k in G', denoted by $|\mathcal{N}'_{uk}|$. Since we split $\mathcal{N}_i$ by the same probability into v_j and v_k during the split operation, $\mathbb{E}[|\mathcal{N}'_{uk}|] = |\mathcal{N}_{ui}|/2$. We prove the lemma by adding one. □

Lemma 10. $\mathbb{E}[T(G'')] = T(G) + h_i(t_i/d_i + 1) - t_i/2$.

Proof. Following Lemma 9, $|\mathcal{N}_{ui}|/2 + 1$ triangles are expected to be created for each target node $u \in \mathcal{S}$ during the adjustment. We estimate $|\mathcal{N}_{ui}|$ by $2t_i/d_i$ based on the relation

$$\sum_{k \in \mathcal{N}_i} |\mathcal{N}_{ki}| = 2t_i,$$

where $\mathcal{N}_i$ is the set of neighbors of v_i in G. We get

$$\mathbb{E}[T(G'') - T(G')] = h_i(t_i/d_i + 1)$$

by repeating the adjustment h_i times for every possible $u \in \mathcal{S}$. We prove the lemma by combining it with Lemma 8. □

Lemma 11. $\mathbb{E}[|\bar{\mathcal{E}}|] = |\mathcal{E}''| - 3T(G'')/|\mathcal{E}''| - 1$.

Proof. Let v_o and v_p be the target nodes of the merge operation, and let t''_{op} be the number of triangles containing v_o and v_p in G''. A single edge (v_o, v_p) is always removed by the merge operation as v_o and v_p are merged into one. The number of additional edges removed by the merge operation is given as t''_{op}, which is estimated by $3T(G'')/|\mathcal{E}''|$ based on the relation

$$\sum_{(i,j) \in \mathcal{E}''} t''_{ij} = 3T(G'').$$

We prove the lemma by adding one to $3T(G'')/|\mathcal{E}''|$. □

A.2 Proof of Lemma 1

We restate Lemma 1 given in Section 3.1.2 and present the full proof based on Lemma 8 to 11.

Lemma 1. *Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, let $\bar{G} = (\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathbf{X}})$ be the result of NodeSam. Then, $\mathbb{E}[|\bar{\mathcal{V}}| - |\mathcal{V}|] = 0$ and $\mathbb{E}[|\bar{\mathcal{E}}| - |\mathcal{E}|] = 0$.*

Proof. We prove only the edge part, since it is straightforward that NodeSam does not change the number of nodes. Let m be the expected number of edges removed by the merge operation:

$$m \equiv \mathbb{E}[|\mathcal{E}''| - |\bar{\mathcal{E}}|]. \tag{2}$$

Then, m is rewritten as follows by Lemma 11:

$$m = \mathbb{E}[3T(G'')/|\mathcal{E}''|] + 1. \tag{3}$$

If we ignore the dependence between $T(G'')$ and $|\mathcal{E}''|$, which is negligible in real-world graphs that sastify $|\mathcal{E}| \gg h_i$, Equation (3) changes into

$$m = \frac{3\mathbb{E}[T(G'')]}{\mathbb{E}[|\mathcal{E}''|]} + 1. \tag{4}$$

The first term $\mathbb{E}[T(G'')]$ is rewritten by Lemma 10:

$$\mathbb{E}[T(G'')] = |\mathcal{V}|t_i/3 + h_i(t_i/d_i + 1) - t_i/2, \tag{5}$$

where the global number $T(G)$ of triangles is replaced with a local estimation $|\mathcal{V}|t_i/3$, since we select v_i from G uniformly at random and the relation $\mathbb{E}_i[t_i] = 3T(G)$ holds between $T(G)$ and t_i.

The second term $\mathbb{E}[|\mathcal{E}''|]$ is given by the definition of h_i:

$$\mathbb{E}[|\mathcal{E}''|] = |\mathcal{E}| + h_i + 1. \tag{6}$$

We apply Equation (5) and (6) to Equation (4) to get

$$m = \frac{3(t_i/d_i + 1)h_i + (|\mathcal{V}| - 3/2)t_i}{h_i + |\mathcal{E}| + 1} + 1. \tag{7}$$

If we denote the first term of the right hand side of Equation (7) as A, we can verify that Equation (1) is the solution of $h_i = A$:

$$\begin{aligned} &h_i = A \\ &h_i(h_i + |\mathcal{E}| + 1) = 3(t_i/d_i + 1)h_i + (|\mathcal{V}| - 3/2)t_i \\ &h_i^2 + (|\mathcal{E}| - 3t_i/d_i - 2)h_i - (|\mathcal{V}| - 3/2)t_i = 0. \end{aligned} \tag{8}$$

Then, the right hand side of Equation (7) changes into $h_i + 1$. By the definition of m and h_i, we finally get

$$\mathbb{E}[|\mathcal{E}''| - |\bar{\mathcal{E}}|] = \mathbb{E}[|\mathcal{E}''|] - |\mathcal{E}| - 1 + 1,$$

which changes into $\mathbb{E}[|\bar{\mathcal{E}}| - |\mathcal{E}|] = 0$ and proves the lemma. □

B PROOFS OF LEMMA 2 TO 7

B.1 Proof of Lemma 2

Proof. The split and merge operations preserve the connectivity, since they are transformations between a single node and a pair of adjacent nodes. The adjustment also preserves the connectivity since it makes new edges only between two-hop neighbors. □

B.2 Proof of Lemma 3

Proof. The time complexity of the split operation is $O(d|\mathcal{V}| + |\mathcal{E}|)$, and the complexity of the merge operation is the same. The time complexity of the adjustment operation is $O(|\mathcal{V}| + |\mathcal{E}|)$ since it does not change $\mathbf{X}$. The space complexities are always smaller than or equal to the time complexities [2]. We prove the lemma by combining the complexities of all these three operations. □

B.3 Proof of Lemma 5

Proof. The proof is straightforward for $\mathcal{V}$, since SubMix does not change the number of nodes. For $\mathcal{E}$, let $\mathcal{E}_s$ and $\mathcal{E}'_s$ be the edges of induced subgraphs of S and S' in graphs G and G', respectively. Then, the following equations hold, since a) SubMix replaces only the edges in $\mathcal{E}_s$, b) G and G' are selected independently, and c) S and S' are selected by the same diffusion algorithm:

$$\begin{aligned}\mathbb{E}_{G,\bar{G}}[|\mathcal{E}| - |\bar{\mathcal{E}}|] &= \mathbb{E}_{G,G'}[|\mathcal{E}_s| - |\mathcal{E}'_s|] \\ &= \mathbb{E}_{G}[|\mathcal{E}_s|] - \mathbb{E}_{G'}[|\mathcal{E}'_s|] = 0.\end{aligned}$$

The fact that $\mathbb{E}[|\mathcal{E}| - |\bar{\mathcal{E}}|] = 0$ proves the lemma. □

B.4 Proof of Lemma 6

Proof. Let G' be a graph chosen for the augmentation of G by SubMix. The sets S and S' of nodes selected for the replacement are connected due to Lemma 6. If we treat the induced subgraphs of S and S' as supernodes, the replacement of S with S' does not change the connectivity of G, proving the lemma. □

B.5 Proof of Lemma 7

Proof. The time complexity of Algorithm 5 without including the sampling function is $O(pd|\mathcal{V}| + |\mathcal{E}| + |\mathcal{E}'|)$. We assume that the number of labels is negligible. The time complexity of Algorithm 6 is $O(|\mathcal{E}| + |\mathcal{E}'|)$, since the PPR diffusion is $O(|\mathcal{E}|)$ and $O(|\mathcal{E}'|)$ for G and G', respectively. The space complexities are always smaller than or equal to the time complexities [2]. □

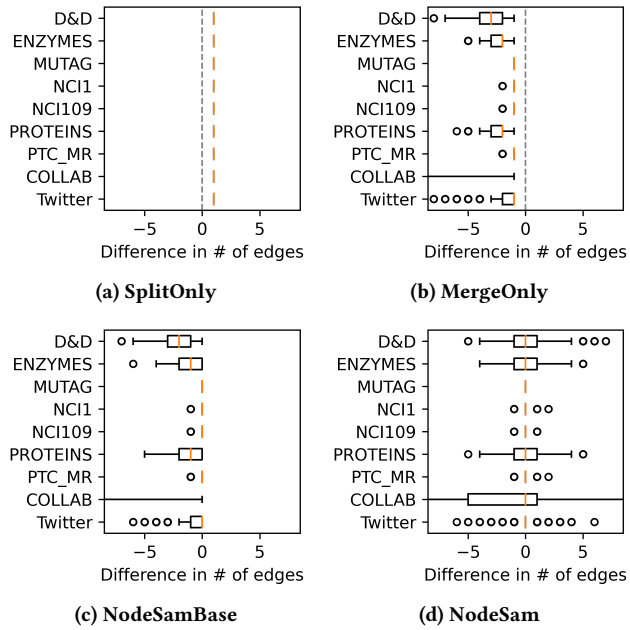
C NODE FEATURE INFORMATION

We describe detailed information of node features in our datasets, which are summarized in Table 2.

- **Molecular graphs.** Each node represents an element in a chemical compound, and contains a one-hot feature representing its atomic type, such as carbon or oxygen.
- **Twitter.** Each graph represents a tweet, and each node is a keyword or a symbol appearing in a tweet. Every node has a one-hot feature representing its keyword or symbol.
- **COLLAB.** Since the dataset does not contain node features, we compute the degree of every node and assign a one-hot feature vector to each node based on its degree. Thus, the length of feature vectors is the same as the number of unique degrees in the dataset. The same approach was also done in previous work [41] for graph classification.

D ABLATION STUDY FOR NODESAM

We perform an additional ablation study for NodeSam to confirm the effect of the adjustment operation for making unbiased changes of the number of edges. Figure 8 shows box plots for the difference in the number of edges for all variants of NodeSam: (a) SplitOnly, (b) MergeOnly, (c) NodeSamBase, and (d) NodeSam. The detailed information of such variants is discussed in Section 4.4. The figure shows that the split operation increases the number of edges by one in all cases, since it splits a node into two by inserting a single edge, while the merge operation decreases the number of edges by more than one. This is because the merge operation eliminates the triangles that contain both of the target nodes. Our NodeSam with the adjustment operation makes unbiased changes, compensating for the removed edges, compared with NodeSamBase.



(a) SplitOnly (b) MergeOnly

(c) NodeSamBase (d) NodeSam

Figure 8: Box plots illustrating the difference in the number of edges. NodeSamBase, which does not perform the adjustment operation, tends to decrease the number of edges due to the merge operation that removes additional edges.