SUGAR: Subgraph Neural Network with Reinforcement Pooling and Self-Supervised Mutual Information Mechanism

Qingyun Sun[1,2], Jianxin Li[1,2], Hao Peng[1], Jia Wu[3], Yuanxing Ning[1,2], Phillip S. Yu[4], Lifang He[5]
[1] Beijing Advanced Innovation Center for Big Data and Brain Computing, Beihang University, Beijing 100191, China
[2] School of Computer Science and Engineering, Beihang University, Beijing 100191, China
[3] Department of Computing, Macquarie University, Sydney, Australia
[4] Department of Computer Science, University of Illinois at Chicago, Chicago, USA
[5] Department of Computer Science and Engineering, Lehigh University, Bethlehem, PA, USA
{sunqy,lijx,penghao,ningyx}@act.buaa.edu.cn,jia.wu@mq.edu.au,psyu@uic.edu,lih319@lehigh.edu

ABSTRACT

Graph representation learning has attracted increasing research attention. However, most existing studies fuse all structural features and node attributes to provide an overarching view of graphs, neglecting finer substructures' semantics, and suffering from interpretation enigmas. This paper presents a novel hierarchical subgraph-level selection and embedding-based graph neural network for graph classification, namely SUGAR, to learn more discriminative subgraph representations and respond in an explanatory way. SUGAR reconstructs a sketched graph by extracting striking subgraphs as the representative part of the original graph to reveal subgraph-level patterns. To adaptively select striking subgraphs without prior knowledge, we develop a reinforcement pooling mechanism, which improves the generalization ability of the model. To differentiate subgraph representations among graphs, we present a self-supervised mutual information mechanism to encourage subgraph embedding to be mindful of the global graph structural properties by maximizing their mutual information. Extensive experiments on six typical bioinformatics datasets demonstrate a significant and consistent improvement in model quality with competitive performance and interpretability.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Learning latent representations**; • **Mathematics of computing** → **Graph algorithms**.

KEYWORDS

Graph Neural Networks, Graph Classification, Graph Pooling, Reinforcement Learning, Mutual Information

ACM Reference Format:
Qingyun Sun, Jianxin Li, Hao Peng, Jia Wu, Yuanxing Ning, Phillip S. Yu, Lifang He. 2021. SUGAR: Subgraph Neural Network with Reinforcement Pooling and Self-Supervised Mutual Information Mechanism. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3442381.3449822

WWW '21, April 19–23, 2021, Ljubljana, Slovenia

ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3449822

1 INTRODUCTION

Graphs have been widely used to model the complex relationships between objects in many areas including computer vision [14, 59], natural language processing [36], anomaly detection [1, 33], academic network analysis [44, 62], bioinformatics analysis [12, 46], etc. By learning a graph-based representation, it is possible to capture the sequential, topological, geometric, and other relational characteristics of structured data. However, graph representation learning is still a non-trivial task because of its complexity and flexibility. Moreover, existing methods mostly focus on node-level embedding [17, 21, 48], which is insufficient for subgraph analysis. Graph-level embedding [32, 40, 61, 63] is critical in a variety of real-world applications such as predicting the properties of molecules in drug discovery [28], and community analysis in social networks [25]. In this paper, we focus on graph-level representation for both subgraph discovery and graph classification tasks in an integrative way.

In the literature, a substantial amount of research has been devoted to developing graph representation techniques, ranging from traditional graph kernel methods to recent graph neural network methods. In the past decade, many graph kernel methods [22] have been proposed that directly exploit graph substructures decomposed from it using kernel functions, rather than vectorization. Due to its specialization, these methods have shown competitive performance in particular application domains. However, there are limitations in two aspects. Firstly, kernel functions are mostly handcrafted and heuristic [3, 40, 41, 57], which are inflexible and may suffer from poor generalization performance. Secondly, the embedding dimensionality usually grows exponentially with the growth in the substructure's size, leading to sparse or non-smooth representations [7].

With the recent advances in deep learning, Graph Neural Networks (GNNs) [53] have achieved significant success in mining graph data. GNNs attempt to extend the convolution operation from regular domains to arbitrary topologies and unordered structures, including spatial-based [16, 32, 42] and spectral-based methods [6, 10, 26]. Most of the current GNNs are inherently flat, as they only propagate node information across edges and obtain graph representations by globally summarizing node representations. These summarisation approaches include averaging over all nodes [13], adding a virtual node [27], using connected layers [16] or convolutional layers [63], etc.

However, graphs have a wide spectrum of structural properties, ranging from nodes, edges, motifs to subgraphs. The local substructures (i.e., motifs and subgraphs) in a graph always contain vital characteristics and prominent patterns [47], which cannot be captured during the global summarizing process. For example, the functional units in organic molecule graphs are certain substructures consisting of atoms and the bonds between them [9, 12]. To overcome the problem of flat representation, some hierarchical methods [15, 61] use local structures implicitly by coarsening nodes progressively, which often leads to the unreasonableness of the classification result. To exploit substructures with more semantics, some researchers [24, 31, 35, 58] exploit motifs (i.e., small, simple structures) to serve as local structure features explicitly. Despite its success, it requires domain expertise to carefully design specific motif extracting rules for various applications carefully. Moreover, subgraphs are exploited to preserve higher-order structural information by motif combination [58], subgraph isomorphism counting [5], rule-based extraction [56], etc. However, effectively exploiting higher-order structures for graph representation is a non-trivial problem due to the following major challenges: (1) *Discrimination.* Generally, fusing all features and relations to obtain an overarching graph representation always brings the potential concerns of over-smoothing, resulting in the features of graphs being indistinguishable. (2) *Prior knowledge.* Preserving structural features in the form of similarity metrics or motif is always based on heuristics and requires substantial prior knowledge, which is tedious and ad-hoc. (3) *Interpretability.* Many methods exploit substructures by coarsening them progressively. This is not suitable to give prominence to individual substructures, resulting in a lack of sufficient interpretability.

To address the aforementioned challenges, we propose a novel **SU**b**G**r**A**ph neural network with **R**einforcement pooling and self-supervised mutual information mechanism, named **SUGAR**. Our goal is to develop an effective framework to adaptively select and learn discriminative representations of striking subgraphs that generalize well without prior knowledge and respond in an explanatory way. SUGAR reconstructs a sketched graph to reveal subgraph-level patterns, preserving structural information in a three-level hierarchy: node, intra-subgraph, and inter-subgraph. To obtain more representative information without prior knowledge, we design a reinforcement pooling mechanism to select more striking subgraphs adaptively by a reinforcement learning algorithm. Moreover, to discriminate subgraph embeddings among graphs, a self-supervised mutual information mechanism is also introduced to encourage subgraph representations preserving global properties by maximizing mutual information between local and global graph representations. Extensive experiments on six typical bioinformatics datasets demonstrate significant and consistent improvements in model quality. We highlight the advantages of SUGAR[1] as follows:

- **Discriminative.** SUGAR learns discriminative subgraph representations among graphs, which are aware of both local and global properties.
- **Adaptable.** SUGAR adaptively finds the most striking subgraphs given any graph without prior knowledge, which allows it to perform in a superior way across various types of graphs.
- **Interpretable.** SUGAR explicitly indicates which subgraphs are dominating the learned result, which provides insightful interpretation into downstream applications.

[1]Code is available at https://github.com/RingBDStack/SUGAR.

2 RELATED WORK

This section briefly reviews graph neural networks, graph pooling, and self-supervised learning on graphs.

2.1 Graph Neural Networks

Generally, graph neural networks follow a message-passing scheme recursively to embed graphs into a continuous and low-dimensional space. Prevailing methods capture graph properties in different granularities, including node [20, 32, 45], motif [24, 31, 35], and subgraph [2, 56, 58].

Several works [20, 32, 45, 50, 54] generate graph representations by globally fusing node features. PATCHY-SAN [32] represents a graph as a sequence of nodes and generates local normalized neighborhood representations for each node. RNN autoencoder based methods [20, 45] capture graph properties by sampling node sequence, which is implicit and unaware of exact graph structures. Graph capsule networks [50, 54] capture node features in the form of capsules and use a routing mechanism to generate high-level features. However, these methods are incapable of exploiting the hierarchical structure information of graphs.

Local substructures (i.e., motifs and subgraphs) are exploited to capture more complex structural characteristics. Motif-based methods [24, 31, 35, 58] are limited to enumerate exact small structures within graphs as local structure features. The motif extraction rules must be manually designed with prior knowledge. Other works exploit subgraphs by motif combination [58], subgraph isomorphism counting [5], and rule-based extraction [56] for graph-level tasks (e.g., subgraph classification [2], graph evolution prediction [30], and graph classification [5, 56, 58]). NEST [58] explores subgraph-level patterns by various combinations of motifs. The most relevant work to ours is SGN [56], which detects and selects appropriate subgraphs based on pre-defined rules, expanding the structural feature space effectively. Compared to our model, the subgraph detection and selection procedure is based on heuristics, and it is difficult for SGN to provide sufficient information when subgraphs become too large.

The aforementioned methods have many limitations in terms of discrimination, prior knowledge, and interpretability. In our framework, we address these problems by representing graphs as adaptively selected striking subgraphs.

2.2 Graph Pooling

Graph pooling is investigated to reduce the entire graph information into a coarsened graph, which broadly falls into two categories: cluster pooling and top-k selection pooling.

Cluster pooling methods (e.g., DiffPool [61], EigenPooling [29] and ASAP [39]) group nodes into clusters and coarsen the graph based on the cluster assignment matrix. Feature and structure information is utilized implicitly during clustering, leading to a lack of interpretability. Furthermore, the number of clusters is always determined by a heuristic or performs as a hyper-parameter, and cluster operations always lead to high computational cost.

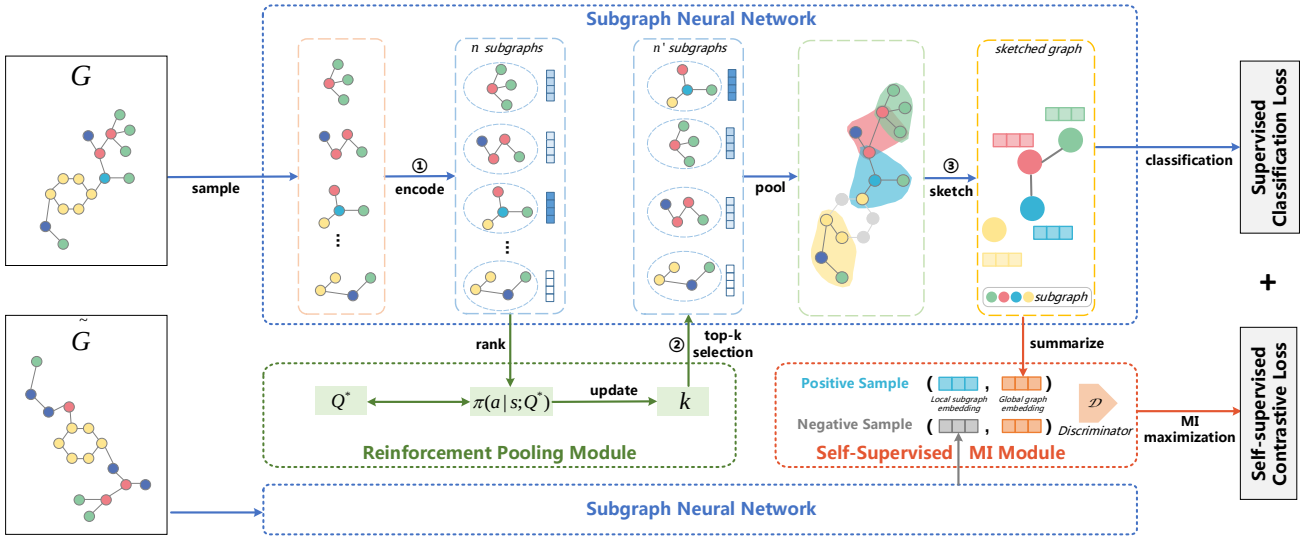


Figure 1: An illustration of the SUGAR architecture. Assuming the single graph setup (i.e., G is provided as input and $\tilde{G}$ is an alternative graph providing negative samples), SUGAR consists of the following steps: ① Subgraph sampling and encoding: for each graph, a fixed number of subgraphs is sampled and encoded by an intra-subgraph attention mechanism; ② Subgraph selection: striking subgraphs are selected by a reinforcement learning module and pooled into a sketched graph; ③ Subgraph sketching: every supernode (i.e., subgraph) in the sketched graph is fed into an inter-subgraph attention layer; Subgraph representations are further enhanced by maximizing mutual information between local subgraph (in cyan) and global graph (in orange) representations; the graph classification result is voted by classifying subgraphs.

Top-k selection pooling methods compute the importance scores of nodes and select nodes with pooling ratio k to remove redundant information. Top-k pooling methods are generally more memory efficient as they avoid generating dense cluster assignments. [8] and [15] select nodes based on their scalar projection values on a trainable vector. SortPooling [63] sorts nodes according to their structural roles within the graph using the WL kernel. SAGPool [23] uses binary classification to decide the preserving nodes. To our best knowledge, current top-k selection pooling methods are mostly based on heuristics, since they cannot parameterize the optimal pooling ratio [23]. The pooling ratio is always taken as a hyper-parameter and tuned during the experiment [15, 23], which lacks a generalization ability. However, the pooling ratios are diverse in different types of graphs and should be chosen adaptively.

Our framework adopts a reinforcement learning algorithm to optimize the pooling ratio, which can be trained with graph neural networks in an end-to-end manner.

2.3 Self-Supervised Learning on Graphs

Self-supervised learning has shown superiority in boosting the performance of many downstream applications in computer vision [18, 19] and natural language processing [11, 60]. Recent works [37, 38, 43, 49] harness self-supervised learning for GNNs and have shown competitive performance. DGI [49] learns a node encoder that maximizes the mutual information between patch representations and corresponding high-level summaries of graphs. GMI [37] generalizes the conventional computations of mutual information from vector space to the graph domain, where measuring the mutual information from the two aspects of node features and topological structure. InfoGraph [43] learns graph embeddings by maximizing the mutual information between graph embeddings and substructure embeddings of different scales (e.g., nodes, edges, triangles). Our approach differs in that we aim to obtain subgraph-level representations mindful of the global graph structural properties.

3 OUR APPROACH

This section proposes the framework of SUGAR for graph classification, addressing the challenges of discrimination, prior knowledge, and interpretability simultaneously. The overview of SUGAR is shown in Figure 1. We first introduce the subgraph neural network, and then followed by the reinforcement pooling mechanism and the self-supervised mutual information mechanism.

We represent a graph as $G = (V, X, A)$, where $V = \{v_1, v_2, \cdots, v_N\}$ denotes the node set, $X \in \mathbb{R}^{N \times d}$ denotes the node features, and $A \in \mathbb{R}^{N \times N}$ denotes the adjacency matrix. N is the number of nodes, and d is the dimension of the node feature. Given a dataset $(\mathcal{G}, \mathcal{Y}) = \{(G_1, y_1), (G_2, y_2), \cdots (G_n, y_n)\}$, where $y_i \in \mathcal{Y}$ is the label of $G_i \in \mathcal{G}$, the task of graph classification is to learn a mapping function $f : \mathcal{G} \rightarrow \mathcal{Y}$ that maps graphs to the label sets.

3.1 Subgraph Neural Network

As the main component of SUGAR, the subgraph neural network reconstructs a sketched graph by extracting striking subgraphs as the original graph's representative part to reveal subgraph-level patterns. In this way, the subgraph neural network preserves the graph properties through a three-level hierarchy: node, intra-subgraph,

and inter-subgraph. Briefly, there are three steps to build a subgraph neural network: (1) sample and encode subgraphs from the original graph; (2) select striking subgraphs by a reinforcement pooling module; (3) build a sketched graph and learn subgraph embeddings by an attention mechanism and a self-supervised mutual information mechanism.

Step-1: Subgraph sampling and encoding. First, we sample n subgraphs from the original graph. We sort all nodes in the graph by their degree in descending order and select the first n nodes as the central nodes of subgraphs. For each central node, we extract a subgraph using the breadth-first search (BFS) algorithm. The number of nodes in each subgraph is limited to s. The limitation of n and s is to maximize the original graph structure's coverage with a fixed number of subgraphs. Then, we obtain a set of subgraphs $\{g_1, g_2, \cdots, g_n\}$.

Second, we learn a GNN-based encoder, $\mathcal{E} : \mathbb{R}^{s\times d} \times \mathbb{R}^{s\times s} \rightarrow \mathbb{R}^{s\times d_1}$, to acquire node representations within subgraphs, where d_1 is the dimension of node representation. Then the node representations $\mathbf{H}(g_i) \in \mathbb{R}^{s\times d_1}$ for nodes in subgraph g_i can be obtained by the generalized equation:

$$\mathbf{H}(g_i) = \mathcal{E}(g_i) = \{\mathbf{h}_j | v_j \in V(g_i)\}. \tag{1}$$

We unify the formulation of $\mathcal{E}$ as a message passing framework:

$$\mathbf{h}_i^{(l+1)} = \mathrm{U}^{(l+1)}(\mathbf{h}_i^{(l)}, \mathrm{AGG}(\mathrm{M}^{(l+1)}(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)}) | v_j \in N(v_i))), \tag{2}$$

where $\mathrm{M}(\cdot)$ denotes the message generation function, $\mathrm{AGG}(\cdot)$ denotes the aggregation function, and $\mathrm{U}(\cdot)$ denotes the state updating function. Various formulas of GNNs can be substituted for Eq. (1).

Third, to encode node representations into a unified subgraph embedding space, we leverage an intra-subgraph attention mechanism to learn the node importance within a subgraph. The attention coefficient $c_j^{(i)}$ for v_j is computed by a single forward layer, indicating the importance of v_j to subgraph g_i:

$$c_j^{(i)} = \sigma(\mathbf{a}_{intra}^{\mathrm{T}} \mathbf{W}_{intra} \mathbf{h}_j^{(i)}), \tag{3}$$

where $\mathbf{W}_{intra} \in \mathbb{R}^{d_1\times d_1}$ is a weight matrix, and $\mathbf{a}_{intra} \in \mathbb{R}^{d_1}$ is a weight vector. $\mathbf{W}_{intra}$ and $\mathbf{a}_{intra}$ are shared among nodes of all subgraphs. Then, we normalize the attention coefficients across nodes within a subgraph via a softmax function. After this, we can compute the representations $\mathbf{z}_i$ of g_i as follows:

$$\mathbf{z}_i = \sum_{v_j \in \mathbf{V}(g_i)} c_j^{(i)} \mathbf{h}_j^{(i)}. \tag{4}$$

Step-2: Subgraph selection. To denoise randomly sampled subgraphs, we need to select subgraphs with prominent patterns, typically indicated by particular subgraph-level features and structures. We adopt *top-k* sampling with an adaptive pooling ratio $k \in (0, 1]$ to select a portion of subgraphs. Specifically, we employ a trainable vector $\mathbf{p}$ to project all subgraph features to 1D footprints $\{val_i | g_i \in G\}$. val_i measures how much information of subgraph g_i can be retained when projected onto the direction of $\mathbf{p}$. Then, we take the $\{val_i\}$ as the importance values of subgraphs and rank the subgraphs in descending order. After that, we select the top $n' = \lceil k \cdot n \rceil$ subgraphs and omit all other subgraphs at the current batch. During the training phase, val_i of subgraph g_i on $\mathbf{p}$ is computed as:

$$val_i = \frac{\mathbf{z}_i \mathbf{p}}{\|\mathbf{p}\|}, \quad idx = rank(\{val_i\}, n'), \tag{5}$$

where $rank(\{val_i\}, n')$ is the operation of subgraph ranking, which returns the indices of the n'-largest values in $\{val_i\}$. idx returned by $rank(\{val_i\}, n')$ denotes the indices of selected subgraphs. k is updated every epoch by a reinforcement learning mechanism introduced in Section 3.2.

Step-3: Subgraph sketching. Since we learned the independent representations of subgraph-level features, we assemble the selected subgraphs to reconstruct a sketched graph to capture their inherent relations. First, as shown in Fig. 1, we reduce the original graph into a sketched graph $G^{ske} = (V^{ske}, E^{ske})$ by treating the selected subgraphs as supernodes. The connectivity between supernodes is determined by the number of common nodes in the corresponding subgraphs. Specifically, an edge $e(i, j)$ will be added to the sketched graph when the number of common nodes in g_i and g_j exceeds a predefined threshold b_{com}.

$$V^{ske} = \{g_i\}, \forall\ i \in idx; \quad E^{ske} = \{e_{i,j}\}, \forall\ \left|V(g_i) \bigcap V(g_j)\right| > b_{com}. \tag{6}$$

Second, an inter-subgraph attention mechanism is adopted to learn the mutual influence among subgraphs from their vectorized feature. More specifically, the attention coefficient α_{ij} of subgraph g_i on g_j can be calculated by the multi-head attention mechanism as in [48]. Then the subgraph embeddings can be calculated as:

$$\mathbf{z}_i' = \frac{1}{M} \sum_{m=1}^{M} \sum_{e_{ij} \in E^{ske}} \alpha_{ij}^m \mathbf{W}_{inter}^m \mathbf{z}_i, \tag{7}$$

where α_{ij} is the attention coefficient, $\mathbf{W}_{inter} \in \mathbb{R}^{d_2\times d_1}$ is a weight matrix, and M is the number of independent attention.

Third, the subgraph embeddings will be further enhanced by a self-supervised mutual information mechanism introduced in Section 3.3.

After obtaining the subgraph embeddings, we convert them to label prediction through a softmax function. The probability distribution on class labels of different subgraphs can provide an insight into the impacts of subgraphs on the entire graph. Finally, the graph classification results are voted by subgraphs. Concretely, the classification results of all the subgraphs are ensembled by applying sum operation as the final probability distribution of the graph. The indexed class with the maximum probability is assumed to be the predicted graph label.

3.2 Reinforcement Pooling Module

To address the challenge of prior knowledge in *top-k* sampling, we present a novel reinforcement learning (RL) algorithm to update the pooling ratio k adaptively, even when inputting subgraphs of varying sizes and structures. Since the pooling ratio k in *top-k* sampling does not directly attend the graph classification, it cannot be updated by backpropagation. We use an RL algorithm to find optimal $k \in (0, 1]$ rather than tuning it as a hyper-parameter. We model the updating process of k as a finite horizon Markov decision process (MDP). Formally, the state, action, transition, reward and termination of the MDP are defined as follows:

- *State.* The state s_e at epoch e is represented by the indices of selected subgraphs idx defined in Eq. (5) with pooling ratio k:

$$s_e = idx_e \tag{8}$$

- *Action.* RL agent updates k by taking action a_e based on reward. We define the action a as add or minus a fixed value $\Delta k \in [0, 1]$ from k.
- *Transition.* After updating k, we use *top-k* sampling defined in Eq. (5) to select a new set of subgraphs in the next epoch.
- *Reward.* Due to the black-box nature of GNN, it is hard to sense its state and cumulative reward. So we define a discrete reward function $reward(s_e, a_e)$ for each a_e at s_e directly based on the classification results:

$$reward(s_e, a_e) = \begin{cases} +1, & if\ acc_e > acc_{e-1}, \\ 0, & if\ acc_e = acc_{e-1}, \\ -1, & if\ acc_e < acc_{e-1}. \end{cases} \tag{9}$$

  where acc_e is the classification accuracy at epoch e. Eq. (9) indicates if the classification accuracy with a_e is higher than the previous epoch, the reward for a_e is positive, and vice versa.
- *Termination.* If the change of k among ten consecutive epochs is no more than Δk, the RL algorithm will stop, and k will remain fixed during the next training process. This means that RL finds the optimal threshold, which can retain the most striking subgraphs. The terminal condition is formulated as:

$$Range(\{k_{e-10}, \cdots, k_e\}) \leq \Delta k. \tag{10}$$

Since this is a discrete optimization problem with a finite horizon, we use *Q-learning* [52] to learn the MDP. *Q-learning* is an off-policy reinforcement learning algorithm that seeks to find the best action to take given the current state. It fits the Bellman optimality equation as follows:

$$Q^*(s_e, a_e) = reward(s_e, a_e) + \gamma \arg\max_{a'} Q^*(s_{e+1}, a'), \tag{11}$$

where $\gamma \in [0, 1]$ is a discount factor of future reward. We adopt a ε-*greedy* policy with an explore probability ε:

$$\pi(a_e|s_e; Q^*) = \begin{cases} \text{random action}, & \text{w.p. } \varepsilon \\ \arg\max_{a_e} Q^*(s_e, a), & \text{otherwise} \end{cases} \tag{12}$$

This means that the RL agent explores new states by selecting an action at random with probability ε instead of selecting actions based on the max future reward.

The RL agent and graph classification model can be trained jointly in an end-to-end manner, and the complete process of the RL algorithm is shown in Lines 15-18 of Algorithm 1. We have tried other RL algorithms such as multi-armed bandit and DQN, but their performance is not as good as the Q-learning algorithm. The experiment results in Section 4.4 verify the effectiveness of the reinforcement pooling module.

3.3 Self-Supervised Mutual Information Module

Since our model relies on extracting striking subgraphs as the representative part of the original graph, we utilize the mutual information (MI) to measure the expressive ability of the obtained subgraph representations. To discriminate the subgraph representations among graphs, we present a novel method that maximizes the MI between local subgraph representations and the global graph representation. All of the derived subgraph representations are constrained to to be mindful of the global structural properties, rather than enforcing the overarching graph representation to contain all properties.

To obtain the global graph representation $\mathbf{r}$, we leverage a READOUT function to summarize the obtained subgraph-level embeddings into a fixed length vector:

$$\mathbf{r} = \text{READOUT}(\{\mathbf{z}'_i\}_{i=1}^{n'}). \tag{13}$$

The READOUT function can be any permutation-invariant function, such as averaging and graph-level pooling. Specifically, we apply a simple averaging strategy as the READOUT function here.

We use the Jensen-Shannon (JS) MI estimator [34] on the local/global pairs to maximize the estimated MI over the given subgraph/graph embeddings. The JS MI estimator has an approximately monotonic relationship with the KL divergence (the traditional definition of mutual information), which is more stable and provides better results [19]. Concretely, a discriminator $\mathcal{D} : \mathbb{R}^{d_2} \times \mathbb{R}^{d_2} \to \mathbb{R}$ is introduced, which takes a subgraph/graph embedding pair as input and determines whether they are from the same graph. We apply a bilinear score function as the discriminator:

$$\mathcal{D}(\mathbf{z}'_i, \mathbf{r}) = \sigma(\mathbf{z}_i'^T \mathbf{W}_{MI} \mathbf{r}), \tag{14}$$

where $\mathbf{W}_{MI}$ is a scoring matrix and $\sigma(\cdot)$ is the sigmoid function.

The self-supervised MI mechanism is contrastive, as our MI estimator is based on classifying local/global pairs and negative-sampled counterparts. Specifically, the negative samples are provided by pairing subgraph representation $\tilde{\mathbf{z}}$ from an alternative graph $\tilde{G}$ with $\mathbf{r}$ from G. As a critical implementation detail of contrastive methods, the negative sampling strategy will govern the specific kinds of structural information to be captured. In our framework, we take another graph in the batch as the alternative graph $\tilde{G}$ to generate negative samples in a batch-wise fashion. To investigate the impact of the negative sampling strategy, we also devise another MI enhancing method named SUGAR-MICorrupt, which samples negative samples in a corrupted graph (i.e. $\widetilde{G}(V, \widetilde{X}, A) = C(G(V, X, A))$). Following the setting in [49], the corruption function $C(\cdot)$ preserves original vertexes V and adjacency matrix A, whereas it corrupts features, $\widetilde{X}$, by row-wise shuffling of X. We further analyze these two negative sampling strategies in Section 4.5.

The self-supervised MI objective can be defined as a standard binary cross-entropy (BCE) loss:

$$\begin{aligned} \mathcal{L}_{MI}^G = \frac{1}{n' + n_{neg}} (& \sum_{g_i \in G}^{n'} \mathbb{E}_{pos} \left[\log(\mathcal{D}(\mathbf{z}'_i, \mathbf{r})) \right] + \\ & \sum_{g_j \in \tilde{G}}^{n_{neg}} \mathbb{E}_{neg} \left[\log(1 - \mathcal{D}(\tilde{\mathbf{z}}'_j, \mathbf{r})) \right]), \end{aligned} \tag{15}$$

where n_{neg} denotes the number of negative samples. The BCE loss $\mathcal{L}_{MI}^G$ amounts to maximizing the mutual information between z'_i and $\mathbf{r}$ based on the Jensen-Shannon divergence between the joint distribution (positive samples) and the product of marginals (negative samples) [34, 49]. The effectiveness of the self-supervised MI mechanism and several insights are discussed in Section 4.5.

Algorithm 1: The overall process of SUGAR

Input: Graphs with labels $\{G = (V, X, A), y\}$; Number of subgraphs n; Subgraph size s; Initialized pooling ratio k_0; Number of epochs, batches: E, B;
Output: Graph label y

```
// Subgraph sampling
```
Sort all nodes within a graph by their degree in descending order;
Extract subgraphs for the first n nodes;
```
// Train SUGAR
```
for $e = 1, 2, \cdots, E$ **do**
  for $b = 1, 2, \cdots, B$ **do**
    $\mathbf{H}(g_i), \forall g_i \in \mathcal{G}_b \leftarrow$ Eq. (1); `// Subgraph encoding`
    $\mathbf{z} \leftarrow$ Eq. (4); `// Intra-subgraph attention`
    $idx \leftarrow$ Eq. (5); `// Subgraph selection`
    $G^{ske} \leftarrow G$; `// Subgraph sketching`
    $\mathbf{z}'_i \leftarrow$ Eq. (7); `// Inter-subgraph attention`
    `// Self-Supervised MI`
    Sample negative samples;
    $\mathbf{r} \leftarrow$ Eq. (13);
    $\mathcal{L}^G_{MI} \leftarrow$ Eqs. (14) and (15);
    $\mathcal{L} \leftarrow$ Eq. (16);
  end
  `// RL process`
  if *Eq.* (10) *is False* **then**
    $reward(s_e, a_e) \leftarrow$ Eq. (9);
    $a_e \leftarrow$ Eq. (12);
    $k \leftarrow a_e \cdot \Delta k$;
  end
end

3.4 Proposed SUGAR

Optimization. We combine the purely supervised classification loss $\mathcal{L}_{Classify}$ and the self-supervised MI loss $\mathcal{L}^G_{MI}$ in Eq. (15), which acts as a regularization term. The graph classification loss function $\mathcal{L}_{Classify}$ is defined based on cross-entropy. The loss $\mathcal{L}$ of SUGAR is defined as follows:

$$\mathcal{L} = \mathcal{L}_{Classify} + \beta \sum_{G \in \mathcal{G}} \mathcal{L}^G_{MI} + \lambda \left\| \Theta \right\|^2, \tag{16}$$

where β controls the contribution of the self-supervised MI enhancement, and λ is a coefficient for L2 regularization on Θ, which is a set of trainable parameters in this framework. In doing so, the model is trained to predict the entire graph properties while keeping rich discriminative intermediate subgraph representations aware of both local and global structural properties.

Algorithm description. Since graph data in the real world are most large in scale, we employ the mini-batch technique in the training process. Algorithm 1 outlines the training process of SUGAR.

4 EXPERIMENTS

In this section, we describe the experiments conducted to demonstrate the efficacy of SUGAR for graph classification. The experiments aim to answer the following five research questions:

- **Q1.** How does SUGAR perform in graph classification?
- **Q2.** How do the exact subgraph encoder architecture and subgraph size influence the performance of SUGAR?
- **Q3.** How does the reinforcement pooling mechanism influence the performance of SUGAR?
- **Q4.** How does the self-supervised mutual information mechanism influence the performance of SUGAR?
- **Q5.** Does SUGAR select subgraphs with prominent patterns and provide insightful interpretations?

4.1 Experimental Setups

Datasets. We use six bioinformatics datasets namely MUTAG [9], PTC [46], PROTEINS [4], D&D [12], NCI1 [51], and NCI109 [51]. The dataset statistics are summarized in Table 1.

Table 1: Statistics of Datasets.

Dataset	# Graphs	# Classes	Max. Nodes	Avg. Nodes	Node Labels
MUTAG [9]	188	2	28	17.93	7
PTC [46]	344	2	64	14.29	18
PROTEINS [4]	1113	2	620	39.06	3
D&D [12]	1178	2	5748	284.32	82
NCI1 [51]	4110	2	111	29.87	37
NCI109 [51]	4127	2	111	29.6	38

Baselines. We consider a number of baselines, including graph kernel based methods, graph neural network based methods, and graph pooling methods to demonstrate the effectiveness and robustness of SUGAR. *Graph kernel based baselines* include Weisfeiler-Lehman Subtree Kernel (WL) [40], Graphlet kernel (GK) [41], and Deep Graph Kernels (DGK) [57]. *Graph neural network based baselines* include PATCHY-SAN [32], Dynamic Edge CNN (ECC) [42], GIN [55], Graph Capsule CNN (GCAPS-CNN) [50], CapsGNN [54], Anonymous Walk Embeddings (AWE) [20], Sequence-to-sequence Neighbors-to-node Previous Predicted (S2S-N2N-PP) [45], Network Structural Convolution (NEST) [58], and MA-GCNN [35]. *Graph pooling baslines* include SortPool [63], DiffPool [61], gPool [15], EigenPooling [29], and SAGPool [23].

Parameter settings. The common parameters for training the models are set as *Momentum* = 0.9, *Dropout* = 0.5, and L2 norm regularization weight decay = 0.01. Node features are one-hot vectors of node categories. We adopt GCN [21] with 2 layers and 16 hidden units as our subgraph encoder. Subgraph embedding dimension d' is set to 96. In the reinforcement pooling module, we set $\gamma = 1$ in (11) and $\varepsilon = 0.9$ in (12). For each dataset, the parameters $n, s, m, \Delta k$ are set based on the following principles: (1) Subgraph number n and subgraph size s are set based on the average size of all graphs; (2) n_{neg} is set to the same value as n'; (3) Δk is set to $\frac{1}{n}$.

4.2 Overall Evaluation (Q1)

In this subsection, we evaluate SUGAR for graph classification on the aforementioned six datasets. We performed 10-fold cross-validation on each of the datasets. The accuracies, standard deviations, and ranks are reported in Table 2 where the best results are

Table 2: Summary of experimental results: "average accuracy±standard deviation (rank)".

Method	Dataset						Avg. Rank
	MUTAG	PTC	PROTEINS	D&D	NCI1	NCI109	
WL [40]	82.05±0.36 (13)	-	-	79.78±0.36 (5)	82.19± 0.18 (6)	82.46±0.24 (3)	6.75
GK [41]	83.50±0.60 (12)	59.65±0.31 (9)	-	74.62±0.12 (13)	-	-	11.33
DGK [57]	87.44±2.72 (9)	60.08±2.55 (8)	75.68±0.54 (12)	-	80.31±0.46 (9)	80.32±0.33 (7)	9.00
PATCHY-SAN [32]	92.63±4.21 (3)	62.29±5.68 (7)	75.89±2.76 (11)	77.12±2.41 (10)	78.59±1.89 (10)	-	8.20
ECC [42]	89.44 (6)	-	-	73.65 (14)	83.80 (2)	81.87 (4)	6.50
GIN [55]	89.40±5.60 (7)	64.60±7.00 (5)	76.20±2.80 (10)	-	82.70±1.70 (5)	-	6.75
GCAPS-CNN [50]	-	66.01±5.91 (4)	76.40±4.17 (7)	77.62±4.99 (9)	82.72±2.38 (4)	81.12±1.28 (6)	6.00
CapsGNN [54]	86.67±6.88 (10)	-	76.28±3.63 (8)	75.38±4.17 (12)	78.35±1.55 (11)	-	10.25
AWE [20]	87.87±9.76 (8)	-	-	71.51±4.02 (15)	-	-	11.50
S2S-N2N-PP [45]	89.86±1.10 (5)	64.54±1.10 (6)	76.61±0.50 (4)	-	83.72±0.40 (3)	83.64±0.30 (2)	4.00
NEST [58]	91.85±1.57 (4)	67.42±1.83 (3)	76.54±0.26 (6)	78.11±0.36 (8)	81.59±0.46 (8)	81.72±0.41 (5)	5.67
MA-GCNN [35]	93.89±5.24 (2)	71.76±6.33 (2)	79.35±1.74 (2)	81.48±1.03 (3)	81.77±2.36 (7)	-	3.20
SortPool [63]	85.83±1.66 (11)	58.59±2.47 (10)	75.54±0.94 (13)	79.37±0.94 (6)	74.44±0.47 (13)	-	10.60
DiffPool [61]	-	-	76.25 (9)	80.64 (4)	-	-	6.50
gPool [15]	-	-	77.68 (3)	82.43 (2)	-	-	2.50
EigenPool [29]	-	-	76.60 (5)	78.60 (7)	77.00 (12)	74.90 (8)	8.00
SAGPool [23]	-	-	71.86±0.97 (14)	76.45±0.97 (11)	67.45±1.11 (14)	74.06±0.78 (9)	12.00
SUGAR (Ours)	**96.74±4.55(1)**	**77.53±2.82(1)**	**81.34±0.93(1)**	**84.03±1.33(1)**	**84.39±1.63(1)**	**84.82±0.81(1)**	**1.00**

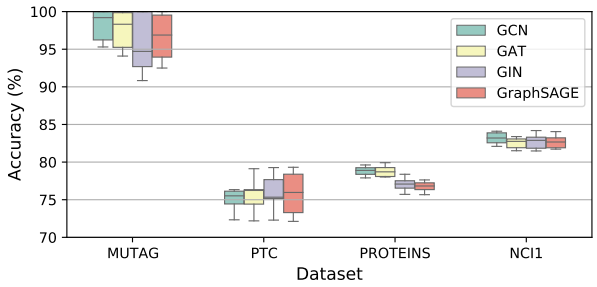


Figure 2: SUGAR with different encoder architecture.

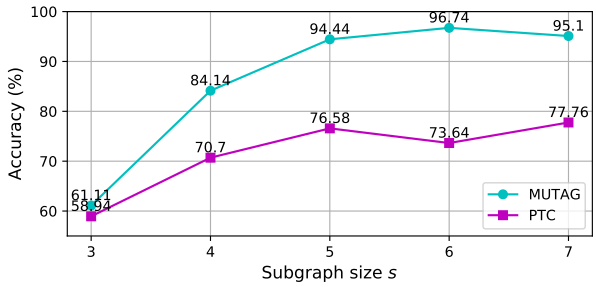


Figure 3: SUGAR with different subgraph size s.

shown in bold. The reported results of the baseline methods come from the initial publications ("–" means not available).

As shown in Table 2, SUGAR consistently outperforms all baselines on all datasets. In particular, SUGAR achieves an average accuracy of 96.74% on the MUTAG dataset, which is a 3.04% improvement over the second-best ranked method MA-GCNN [35]. Compared to node selection pooling baselines (e.g., gPool [15], SAGPool [23]), SUGAR achieves more gains consistently, supporting the intuition behind our subgraph-level denoising approach. Compared to the recent hierarchical method NEST [58] and motif-based method MA-GCNN [35], our method achieves 14.99% and 8.04% improvements in terms of average accuracy on the PTC dataset, respectively. This may be because that both of NEST [58] and MA-GCNN [35] are limited in their ability to enumerated simple motifs, while our method can capture more complex structural information by randomly sampling rather than by pre-defined rules.

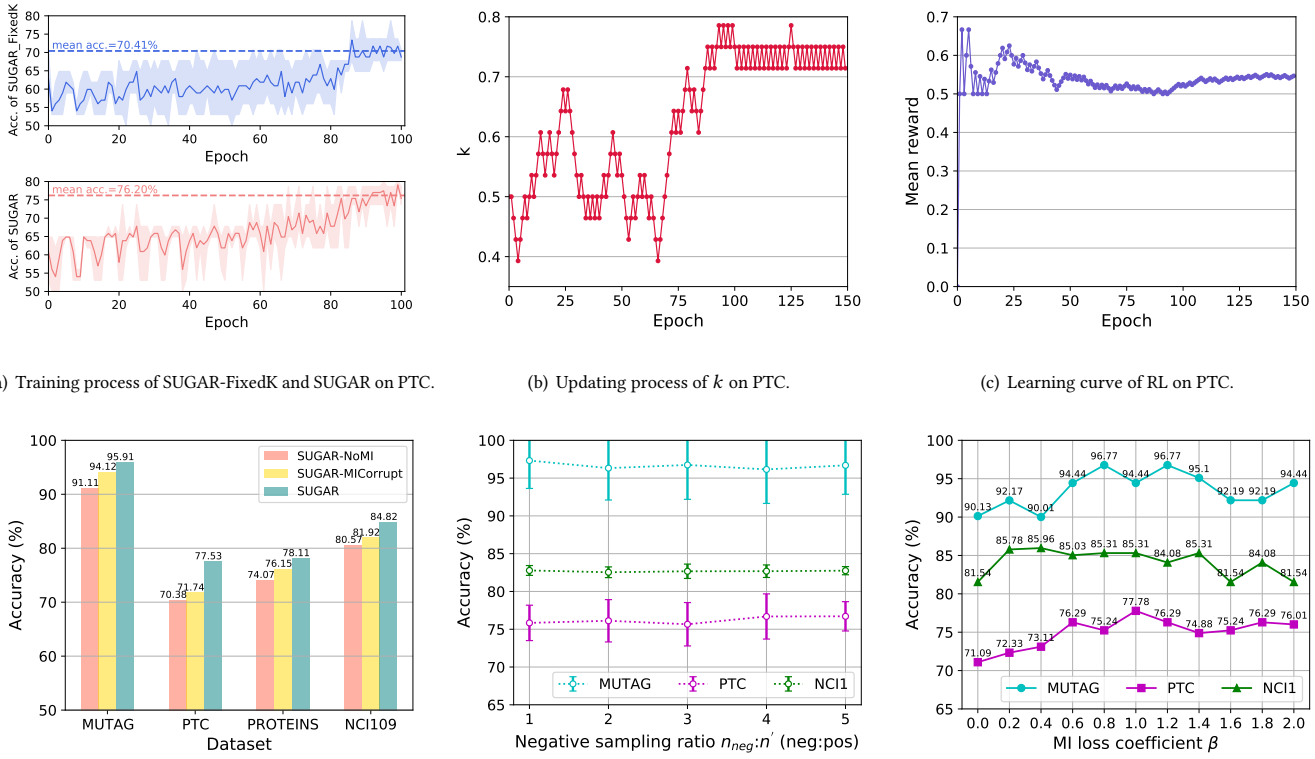
Overall, the proposed SUGAR shows very promising results against recently developed methods.

4.3 Subgraph Encoder and Size Analysis (Q2)

In this subsection, we analyze the impacts of subgraph encoder architecture and subgraph size.

As discussed in Section 3.1, any GNN can be used as the subgraph encoder. In addition to using GCN [21] in the default experiment setting, we also perform experiments with three popular GNN architectures: GAT [48], GraphSAGE[17] (with mean aggregation), and GIN [55]. The results are summarized in Figure 2. We can observe that the performance difference resulting from the different GNNs are marginal. This may be because all the aforementioned GNNs are expressive enough to capture the subgraph properties. *This indicates the proposed SUGAR is robust to the exact encoder architecture.*

Figure 3 shows the performance of SUGAR with different subgraph size s from 3 to 7 on MUTAG and PTC. Although our model does not give satisfactory results with subgraphs of 3 or 4 nodes, it is found that subgraphs of a larger size obviously help to improve the performance. We can also observe that the subgraph size does not significantly improve the performance of SUGAR when it is larger than 5. *This indicates that SUGAR can achieve competitive*



(a) Training process of SUGAR-FixedK and SUGAR on PTC.

(b) Updating process of k on PTC.

(c) Learning curve of RL on PTC.

(d) SUGAR with different negative sampling strategies.

(e) Parameter sensitivity of negative sampling ratio.

(f) Parameter sensitivity of MI loss coefficient β.

Figure 4: The training process and testing performance of SUGAR.

performance when sampled subgraphs can cover most of the basic functional building blocks.

4.4 RL Process Analysis (Q3)

To verify the effectiveness of the reinforcement pooling mechanism, we plot the training process of SUGAR (lower) and the variant SUGAR-FixedK (upper) on PTC in Figure 4(a). SUGAR-FixedK removes the reinforcement pooling mechanism and uses a fixed pooling ratio $k = 1$ (i.e., without subgraph selection). The shadowed area is enclosed by the min and max value of five cross-validation training runs. The solid line in the middle is the mean value of each epoch, and the dashed line is the mean value of the last ten epochs. The mean accuracy of SUGAR with an adaptive pooling ratio achieves a 5.79% improvement over SUGAR-FixedK, supporting the intuition behind our subgraph denoising approach.

Since the RL algorithm and the GNN are trained jointly, the updating and convergence process is indeed important. In Figure 4(b), we visualize the updating process of k in PTC with the initial value $k_0 = 0.5$. Since other modules in the overall framework update with the RL module simultaneously, the RL environment is not very steady at the beginning. As a result, k also does not update steadily during the first 70 epochs. When the framework gradually converges, k bumps for several rounds and meets the terminal condition defined in Eq. (10). Figure 4(c) shows the learning curve in terms of mean reward. We can observe that the RL algorithm converges to the mean reward 0.545 with a stable learning curve.

This suggests that the proposed SUGAR framework can find the most striking subgraphs adaptively.

4.5 Self-Supervised MI Analysis(Q4)

In this subsection, we analyze the impact of the negative sampled strategy, the negative sampling ratio, and the sensitivity of the self-supervised MI loss coefficient.

To evaluate the effectiveness of the self-supervised MI mechanism, we compare SUGAR to its variant SUGAR-NoMI, which removes the self-supervised MI mechanism. To further analyze the impact of the negative sampled strategy, we also compare SUGAR to another variant SUGAR-MICorrupt, which constructs the alternative graph $\widetilde{G}$ by corruption as detailed in Section 3.3. The results are shown in Fig. 4(d). We can observe that models with the self-supervised MI mechanism (i.e., SUGAR and SUGAR-MICorrupt) achieve better performance than SUGAR-NoMI. In addition, SUGAR (i.e., sampling from another graph instance) consistently outperforms than SUGAR-MICorrupt (i.e., sampling from a corrupted graph). A possible reason is that $\widetilde{G}$ loses most of the important structure information during corruption and can only provide weak supervision.

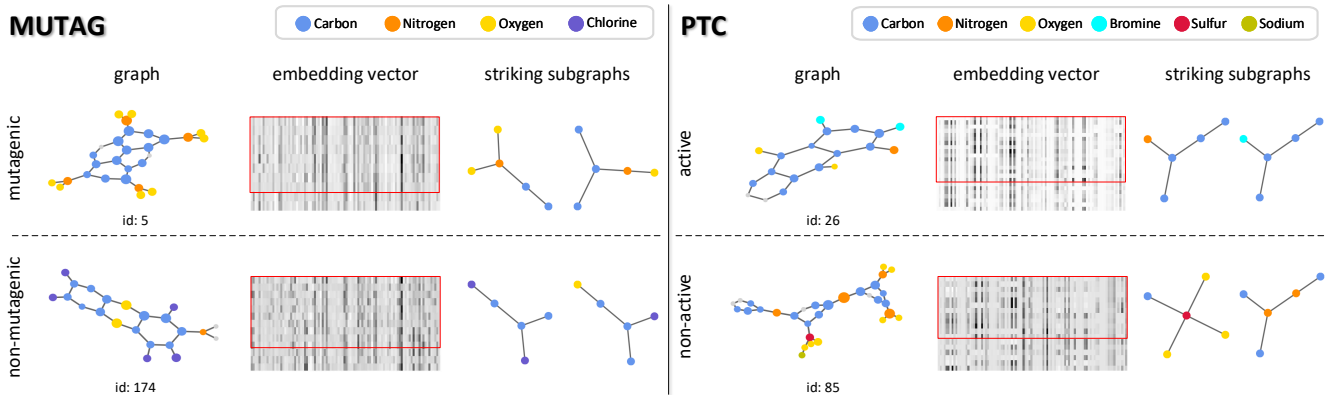


Figure 5: Result visualization of MUTAG (left) and PTC (right) dataset.

We also analyze the sensitivity of two hyper-parameters in the self-supervised MI module, namely negative sampling ratio ($n_{neg} : n'$) and the coefficient of self-supervised MI loss β. The common belief is that contrastive methods require a large number of negative samples to be competitive. Figure 4(e) shows the performance of SUGAR under different negative sampling ratios. The larger negative sampling ratio does not seem to contribute significantly to boosting the performance of SUGAR. This may be because we draw negative samples for every subgraph within the graph. Though the negative sampling ratio is small, it has already provided sufficient self-supervision. As shown in Figure 4(f), when the self-supervised MI loss has more than 0.6 weights compared to graph classification loss, SUGAR achieves better performance. This illustrates that our framework quite benefits from self-supervised training.

This indicates that MI enhancement gives informative self-supervision to SUGAR and negative sampling strategy designs should be considered carefully.

4.6 Visualization (Q5)

In this subsection, we study the power of SUGAR to discover subgraphs with prominent patterns and provide insightful interpretations into the formation of different graphs. Figure 5 illustrates some of the results we obtained on the MUTAG (left) and PTC (right) dataset, where id denotes the graph index in the corresponding dataset. Each row shows our explanations for a specific class in each dataset. Column 1 shows a graph instance after subgraph selection, where the color indicates the atom type of the node (nodes in grey are omitted during subgraph selection) and size indicates the importance of the node in discriminating the two classes. Column 2 shows the $n \times 32$ neuron outputs in descending order of their projection value in the reinforcement pooling module. The first n' rows in the neuron output matrix with the largest projection value are selected as striking subgraphs, roughly indicating their activeness. Column 3 shows striking subgraphs as functional blocks found by SUGAR.

MUTAG consists of 188 molecule graphs labeled according to whether the compound has a mutagenic effect on a bacterium. We observe that the main determinants in the mutagenic class is the nitro group NO_2 connected to a set of carbons. This is the same as the results in [9] which show that the electron-attracting elements conjugated with nitro groups are critical to identifying mutagenic molecules. For the non-mutagenic class, our model takes chlorine connected to carbons as a striking subgraph, which is frequently seen in non-mutagenic molecules.

PTC consists of 344 organic molecules labeled according to whether the compound has carcinogenicity on male rats. The main determinants found by our model are the co-occurrence of carbon rings, nitrogen, sulfur, and oxygen. For instance, one of the striking subgraphs in active compounds is a nitrogen connected to a set of aromatic carbon bonds. This substructure is frequently seen in aromatic amines, nitroaromatics, and azo compounds, which are well-known classes of carcinogens [46]. In addition, our model takes bromine connected to some carbons as a striking subgraph, which is in general agreement with accepted toxicological knowledge. For the non-active class, a striking subgraph found by our model is some oxygen with sulphur bond, which is the same as the knowledge of Leuven2 [9].

This suggests that the proposed SUGAR can find striking subgraphs with discriminative patterns and has great promise to provide sufficient interpretability.

5 CONCLUSION AND FUTURE WORKS

In this paper, we proposed SUGAR, a novel end-to-end graph classification framework by subgraph selection and representation, addressing the challenges of discrimination, prior knowledge, and interpretability. SUGAR preserves both local and global properties hierarchically by reconstructing a sketched graph, selects striking subgraphs adaptively by a reinforcement pooling mechanism, and discriminates subgraph representations by a self-supervised mutual information maximization mechanism. Extensive experiments on graph classification show the effectiveness of our approach. The selected subgraphs and learned weights can provide good interpretability and in-depth insight into structural bioinformatics analysis. Future work includes relating the subgraph sampling strategy to the learned implicit type rules, adopting more advanced reinforcement learning algorithms, and investigating the multi-label problem of subgraphs. Applying SUGAR to other complex datasets and applications such as subgraph classification is another avenue of future research.

ACKNOWLEDGMENTS

The corresponding author is Jianxin Li. The authors of this paper were supported by the NSFC through grants (U20B2053 and 61872022), ARC DECRA Project (No.DE200100964), State Key Laboratory of Software Development Environment (SKLSDE-2020ZX-12), NSF ONR N00014-18-1-2009, and NSF under grants (III-1763325, III-1909323, and SaTC-1930941). This work was also sponsored by CAAI-Huawei MindSpore Open Fund. Thanks for computing infrastructure provided by Huawei MindSpore platform.

REFERENCES

[1] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: a survey. *DMKD* 29, 3 (2015), 626–688.
[2] Emily Alsentzer, Samuel G Finlayson, Michelle M Li, and Marinka Zitnik. 2020. Subgraph Neural Networks. In *Proceedings of the NeurIPS*.
[3] Karsten M Borgwardt and Hans-Peter Kriegel. 2005. Shortest-path kernels on graphs. In *Proceedings of the IEEE ICDM*. 8–pp.
[4] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. 2005. Protein function prediction via graph kernels. *Bioinformatics* 21, suppl_1 (2005), i47–i56.
[5] Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M Bronstein. 2020. Improving Graph Neural Network Expressivity via Subgraph Isomorphism Counting. *arXiv preprint arXiv:2006.09252* (2020).
[6] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and locally connected networks on graphs. In *Proceedings of the ICLR*.
[7] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2018. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE TKDE* 30, 9 (2018), 1616–1637.
[8] Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. 2018. Towards sparse hierarchical graph classifiers. In *NeurIPS 2018 Workshop on Relational Representation Learning*.
[9] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. 1991. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry* 34, 2 (1991), 786–797.
[10] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the NeurIPS*. 3844–3852.
[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the NAACL*. 4171–4186.
[12] Paul D Dobson and Andrew J Doig. 2003. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology* (2003).
[13] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Proceedings of the NeurIPS*. 2224–2232.
[14] Pedro F Felzenszwalb and Daniel P Huttenlocher. 2004. Efficient graph-based image segmentation. *IJCV* 59, 2 (2004), 167–181.
[15] Hongyang Gao and Shuiwang Ji. [n.d.]. Graph U-Nets. In *Proceedings of the ACM ICML*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.).
[16] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. [n.d.]. Neural Message Passing for Quantum Chemistry. In *Proceedings of the ACM ICML*.
[17] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of the NeurIPS*. 1024–1034.
[18] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. 2020. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE CVPR*. 9729–9738.
[19] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. 2019. Learning deep representations by mutual information estimation and maximization. (2019).
[20] Sergey Ivanov and Evgeny Burnaev. 2018. Anonymous Walk Embeddings. In *Proceedings of the ACM ICML*. 2191–2200.
[21] Thomas N Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the ICLR*.
[22] Nils M Kriege, Fredrik D Johansson, and Christopher Morris. 2020. A survey on graph kernels. *Applied Network Science* 5, 1 (2020), 1–42.
[23] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. 2019. Self-Attention Graph Pooling. In *Proceedings of the ACM ICML*, Vol. 97. PMLR, 3734–3743.
[24] John Boaz Lee, Ryan A Rossi, Xiangnan Kong, Sungchul Kim, Eunyee Koh, and Anup Rao. 2019. Graph convolutional networks with motif-based attention. In *Proceedings of the CIKM*. 499–508.
[25] Jia Li, Yu Rong, Hong Cheng, Helen Meng, Wenbing Huang, and Junzhou Huang. 2019. Semi-Supervised Graph Classification: A Hierarchical Graph Perspective. In *Proceedings of the WWW*.
[26] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. [n.d.]. Adaptive graph convolutional neural networks. In *Proceedings of the AAAI*. 3546–3553.
[27] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2016. Gated Graph Sequence Neural Networks. In *Proceedings of the ACM ICML*.
[28] Tengfei Ma, Cao Xiao, Jiayu Zhou, and Fei Wang. 2018. Drug similarity integration through attentive multi-view graph auto-encoders. (2018), 3477–3483.
[29] Yao Ma, Suhang Wang, Charu C Aggarwal, and Jiliang Tang. 2019. Graph convolutional networks with eigenpooling. In *Proceedings of the ACM SIGKDD*. 723–731.
[30] Changping Meng, S Chandra Mouli, Bruno Ribeiro, and Jennifer Neville. 2018. Subgraph Pattern Neural Networks for High-Order Graph Evolution Prediction.. In *Proceedings of the AAAI*. 3778–3787.
[31] Federico Monti, Karl Otness, and Michael M Bronstein. 2018. Motifnet: a motif-based graph convolutional network for directed graphs. In *Proceedings of the IEEE DSW*. IEEE, 225–228.
[32] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *Proceedings of the ACM ICML*. 2014–2023.
[33] Caleb C Noble and Diane J Cook. 2003. Graph-based anomaly detection. In *Proceedings of the ACM SIGKDD*. 631–636.
[34] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. 2016. f-gan: Training generative neural samplers using variational divergence minimization. In *Proceedings of the NeurIPS*. 271–279.
[35] Hao Peng, Jianxin Li, Qiran Gong, Yuanxing Ning, Senzhang Wang, and Lifang He. 2020. Motif-Matching Based Subgraph-Level Attentional Convolutional Network for Graph Classification.. In *Proceedings of the AAAI*. 5387–5394.
[36] Hao Peng, Jianxin Li, Yu He, Yaopeng Liu, Mengjiao Bao, Lihong Wang, Yangqiu Song, and Qiang Yang. 2018. Large-scale hierarchical text classification with recursively regularized deep graph-cnn. In *Proceedings of the WWW*. 1063–1072.
[37] Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang. 2020. Graph Representation Learning via Graphical Mutual Information Maximization. In *Proceedings of the WWW*. 259–270.
[38] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. 2020. GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training. In *Proceedings of the ACM SIGKDD*. 1150–1160.
[39] Ekagra Ranjan, Soumya Sanyal, and Partha P Talukdar. 2020. ASAP: Adaptive Structure Aware Pooling for Learning Hierarchical Graph Representations.. In *Proceedings of the AAAI*. 5470–5477.
[40] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. 2011. Weisfeiler-Lehman Graph Kernels. *JMLR* (2011).
[41] Nino Shervashidze, S. V. N. Vishwanathan, Tobias H. Petri, Kurt Mehlhorn, and Karsten M. Borgwardt. 2009. Efficient graphlet kernels for large graph comparison. *AISTATS* (2009).
[42] Martin Simonovsky and Nikos Komodakis. 2017. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE CVPR*.
[43] Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. 2020. InfoGraph: Unsupervised and Semi-supervised Graph-Level Representation Learning via Mutual Information Maximization. In *Proceedings of the ICLR*.
[44] Qingyun Sun, Hao Peng, Jianxin Li, Senzhang Wang, Xiangyu Dong, Liangxuan Zhao, Philip S Yu, and Lifang He. 2020. Pairwise Learning for Name Disambiguation in Large-Scale Heterogeneous Academic Networks. *arXiv preprint arXiv:2008.13099* (2020).
[45] Aynaz Taheri, Kevin Gimpel, and Tanya Berger-Wolf. 2018. Learning graph representations with recurrent neural network autoencoders. *KDD Deep Learning Day* (2018).
[46] Hannu Toivonen, Ashwin Srinivasan, and Christoph Helma. 2003. Statistical evaluation of the Predictive Toxicology Challenge. *Bioinformatics* (2003).
[47] Julian R Ullmann. 1976. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)* 23, 1 (1976), 31–42.
[48] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph Attention Networks. In *Proceedings of the ICLR*.
[49] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2019. Deep Graph Infomax.. In *Proceedings of the ICLR*.
[50] Saurabh Verma and Zhi Li Zhang. 2018. Graph Capsule Convolutional Neural Networks. *arXiv preprint arXiv:1805.08090* (2018).
[51] Nikil Wale, Ian A Watson, and George Karypis. 2008. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems* (2008).
[52] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.
[53] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE TNNLS* (2020).

[54] Zhang Xinyi and Lihui Chen. 2019. Capsule Graph Neural Network. In *Proceedings of the ICLR*.
[55] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *Proceedings of the ICLR*.
[56] Qi Xuan, Jinhuan Wang, Minghao Zhao, Junkun Yuan, Chenbo Fu, Zhongyuan Ruan, and Guanrong Chen. 2019. Subgraph networks with application to structural feature space expansion. *IEEE TKDE* (2019).
[57] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *Proceedings of the ACM SIGKDD*.
[58] Carl Yang, Mengxiong Liu, Vincent W Zheng, and Jiawei Han. 2018. Node, motif and subgraph: Leveraging network functional blocks through structural convolution. In *Proceedings of the ASONAM*. IEEE, 47–52.
[59] Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. 2018. Graph r-cnn for scene graph generation. In *Proceedings of the ECCV*. 670–685.
[60] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Proceedings of the NeurIPS*. 5753–5763.
[61] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Proceedings of the NeurIPS*.
[62] Fanjin Zhang, Xiao Liu, Jie Tang, Yuxiao Dong, Peiran Yao, Jie Zhang, Xiaotao Gu, Yan Wang, Bin Shao, Rui Li, et al. 2019. Oag: Toward linking large-scale heterogeneous entity graphs. In *Proceedings of the ACM SIGKDD*. 2585–2595.
[63] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI*.