Benchmarking GNNs with GenCAT Workbench

Seiji Maekawa[1] ✉, Yuya Sasaki[1], George Fletcher[2], and Makoto Onizuka[1]

[1] Osaka University, 1-5, Yamadaoka, Suita, Osaka, Japan
maekawa.seiji,sasaki,onizuka@ist.osaka-u.ac.jp
[2] Eindhoven University of Technology, P.O. Box 513, 5600 MB, Eindhoven, Netherlands
g.h.l.fletcher@tue.nl

Abstract. We present GenCAT Workbench, an end-to-end framework with which users can generate synthetic attributed graphs with node labels and evaluate their graph analytic methods, e.g., graph neural networks (GNNs), on the generated graphs. GenCAT Workbench supports various types of graphs with controlled node attributes and graph topology. We demonstrate the GenCAT Workbench and how it clarifies the strong and weak points of GNN models. Our code base is available on Github (https://github.com/seijimaekawa/GenCAT/tree/main/GenCAT_Workbench).

Keywords: attributed graph, graph generator, community, node label

1 Introduction

Graph analytics methods, e.g., graph neural networks (GNNs), have attracted attention from both academia and industry. To clarify their applicability or limitations, many studies address benchmarking GNNs [2, 3]. Though repositories [3] provide collections of real-world graphs with node labels, i.e., an assignment of nodes to groups we call *classes*, the variety of available graphs is still limited.

Because of the large demands for various graphs, synthetic graphs are necessary to mitigate the insufficiency of real-world graphs. Several studies developed benchmarking frameworks with synthetic graphs for evaluating graph analytic methods [2]. However, these frameworks suffer from two drawbacks. First, they use graph generators that cannot generate realistic graphs such as SBM [1]. Second, these frameworks require users to manually set an overwhelming number of parameters of graph generators from scratch when users generate their desired graphs. Hence, the requirements of benchmarking frameworks are 1) the flexibility of controlling the characteristics of generated graphs and 2) the usability for setting parameters of the graph generation.

We present GenCAT Workbench, a framework satisfying both of these desired features. First, our framework allows users to flexibly control the characteristics of generated graphs since we adopt GenCAT [5], an attributed graph generator which supports various characteristics of real-world graphs, such as node degree distributions, attribute distribution, and class structure. The class structure indicates the interplay between classes, attributes and topology. Second, the

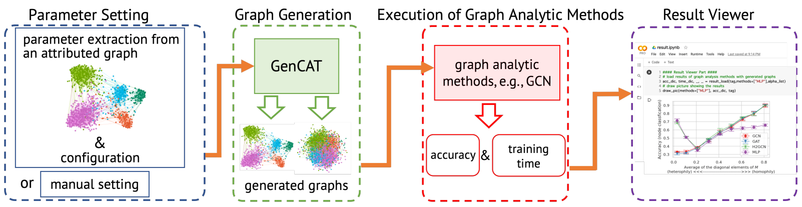


Fig. 1: Overview of the GenCAT Workbench.

GenCAT Workbench can extract the parameters for its graph generation from a given graph and then allows users to configure the parameters, which reduces users' effort compared to fully manual settings. In our demonstration[3], we clarify the pros/cons of each graph analytic method across various topology structures and attribute values. Figure 1 gives an overview of our framework.

Related work. Many studies have addressed benchmarking graph analytic methods [2, 3]. However, there are no frameworks which allow users to generate various graphs and evaluate analytic methods by the graphs. For example, a recent framework uses SBM which generates graphs that are not similar to real-world graphs [2].

2 GenCAT Workbench

GenCAT [5] is the state-of-the-art attributed graph generator which allows users to flexibly control the characteristics of generated graphs. Since it captures the relationships between classes, attributes, and topology, the attributes and topology in generated graphs share the class structure. More specifically, GenCAT can flexibly generate graphs with controlled edge connection proportions between classes, called *class preference mean*. Given as inputs user specified features such as node degrees, attribute distribution, and class features (e.g., class preference mean and class size distribution), GenCAT generates graphs having similar features to these inputs.

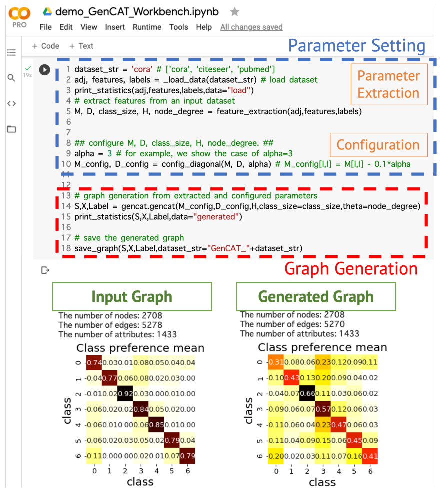
GenCAT is the only method satisfying our requirements; supporting various class structures and extracting parameters from a given graph. Current state of the art methods [1, 7] fail to support one or more features supported by GenCAT. Moreover, it can simulate existing generators in terms of class structures and node degrees. Please see more detailed and precise procedures in [5].

2.1 Features of the GenCAT Workbench

The workflow of the GenCAT Workbench is illustrated in Figure 1. We describe the features of the GenCAT Workbench as follows.

Easy parameter setting. Users can extract statistics from a given graph, and then configure the parameters to obtain their desired graphs. We present

[3] Our demo video is available on `https://www.youtube.com/watch?v=28xV0HRDpCE`.



```
dataset_str = 'cora' # ['cora', 'citeseer', 'pubmed']
adj, features, labels = _load_data(dataset_str) # load dataset
print_statistics(adj,features,labels,data="load")
# extract features from an input dataset
M, D, class_size, H, node_degree = feature_extraction(adj,features,labels)


## configure M, D, class_size, H, node_degree. ##
alpha = 3 # for example, we show the case of alpha=3
M_config, D_config = config_diagonal(M, D, alpha) # M_config[l,l] = M[l,l] - 0.1*alpha


# graph generation from extracted and configured parameters
S,X,Label = gencat.gencat(M_config,D_config,H,class_size=class_size,theta=node_degree)
print_statistics(S,X,Label,data="generated")

# save the generated graph
save_graph(S,X,Label,dataset_str="GenCAT_"+dataset_str)
```

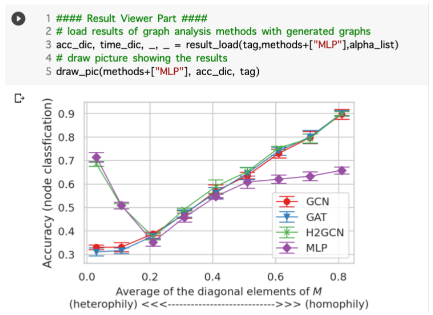
```
The number of nodes: 2708
The number of edges: 5278
The number of attributes: 1433
```

```
The number of nodes: 2708
The number of edges: 5270
The number of attributes: 1433
```

	0	1	2	3	4	5	6
0	0.74	0.03	0.01	0.08	0.05	0.04	0.04
1	0.04	0.77	0.06	0.08	0.02	0.03	0.00
2	0.01	0.02	0.92	0.03	0.00	0.01	0.00
3	0.06	0.02	0.02	0.84	0.05	0.02	0.00
4	0.06	0.01	0.00	0.06	0.85	0.01	0.00
5	0.06	0.02	0.03	0.05	0.02	0.79	0.04
6	0.11	0.00	0.00	0.02	0.01	0.07	0.79

	0	1	2	3	4	5	6
0	0.31	0.08	0.06	0.23	0.12	0.09	0.11
1	0.10	0.43	0.13	0.20	0.09	0.04	0.02
2	0.04	0.07	0.66	0.11	0.03	0.06	0.02
3	0.09	0.06	0.07	0.57	0.12	0.06	0.03
4	0.11	0.06	0.04	0.23	0.47	0.06	0.03
5	0.11	0.04	0.09	0.15	0.06	0.45	0.09
6	0.20	0.02	0.03	0.11	0.07	0.16	0.41

(a) Graph generation usage.



```
#### Result Viewer Part ####
# load results of graph analysis methods with generated graphs
acc_dic, time_dic, _, _ = result_load(tag,methods+["MLP"],alpha_list)
# draw picture showing the results
draw_pic(methods+["MLP"], acc_dic, tag)
```

(b) Use case investigating accuracy on graphs with various class preference means.

Fig. 2: Demonstration.

examples of Cora, Citeseer, and Pubmed, which are commonly used citation networks [4], on the GenCAT Workbench. Also, users can add other new datasets.
Benchmarking graph analytic methods. Users can investigate how each parameter (e.g., class preference mean or the number of edges) affects the performance (e.g., accuracy and training time) of graph analytic methods, while keeping the rest of the parameters the same. This investigation clarifies the advantages and drawbacks of methods on various settings.
End-to-end framework. The GenCAT Workbench provides all necessary components for benchmarking, including parameter setting (i.e., extraction and configuration), graph generation, execution of graph analytic methods, and result viewer. This enables users to easily investigate their methods in various settings.

To enhance the extensibility of the GenCAT Workbench, we implement it on Jupyter Notebook. This allows users to easily add new methods to our framework. This implementation is rather simple yet suitable for GNNs that are changing rapidly.

3 Demonstration Plan

Graph generation usage. We demonstrate and explain how to generate graphs by the GenCAT Workbench in Figure 2a. First, users can choose a dataset from which they extract statistics (see the blue box). In this demonstration, we extract parameters from Cora and configure the class preference mean. As an example, we modify the diagonal elements of the class preference mean such that classes have fewer intra-edges than the original graph, i.e., we simulate a graph with the weaker homophily property than the original graph.

Next, the GenCAT Workbench generates a graph by inputting class features and node degree distribution (see the red box). The GenCAT Workbench presents the heatmaps of class preference means of the original and the generated graphs, which are shown in the bottom part of Figure 2a. Users can observe that the generated graph actually has fewer intra-edges in classes than the original.

Demonstration use case. We demonstrate sample use cases for clarifying the pros and cons of existing graph analytic methods. We pick up three representative GNNs, GCN [4], GAT [6], and H2GCN [8], since graph neural networks are inarguably the hottest topic in graph-based deep learning [2]. The detailed experimental setups are described in our codebase.

In Figure 2b, we demonstrate a sample use case investigating how much class preference means affect the node classification accuracy of the models. First, the GenCAT Workbench extracts parameters from Cora and configures the class preference means to have few intra-edges (i.e., heterophily property) from many intra-edges (i.e., homophily property). Second, the GenCAT Workbench generates graphs with the configured class preference means. Third, the framework executes GNN models on the generated graphs. To compare the models with a graph-agnostic classifier, we execute multi-layer perceptron (MLP).

Next, we discuss observations from this use case. First, GCN, GAT, and H2GCN outperform MLP on graphs with the homophily property since MLP does not use the topology information (see the bottom part in Figure 2b). Then, H2GCN, which considers the heterophily property, performs well on graphs with the heterophily property (the leftmost points). In contrast, GCN and GAT do not perform well since they ignore the heterophily property.

In our demo video, we present two more use cases: 1) accuracy on graphs with various attribute values, and 2) training time per epoch for various numbers of edges. Through the demonstrations, we show how GenCAT Workbench can support investigations of the pros/cons of graph analytics methods on generated graphs with various class preference means, attributes, and graph sizes.

Acknowledgement. This work was supported by JSPS KAKENHI Grant Numbers JP20H00583 and JST PRESTO Grant Number JPMJPR21C5.

References

1. Abbe, E.: Community detection and stochastic block models: recent developments. The Journal of Machine Learning Research (2017)
2. Dwivedi, V.P., Joshi, C.K., Laurent, T., Bengio, Y., Bresson, X.: Benchmarking graph neural networks. arXiv (2020)
3. Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., Leskovec, J.: Open graph benchmark: Datasets for machine learning on graphs. arXiv (2020)
4. Kipf, T.N., Welling, M.: Semi-Supervised Classification with Graph Convolutional Networks. In: ICLR (2017)
5. Maekawa, S., Sasaki, Y., Fletcher, G., Onizuka, M.: GenCAT: Generating Attributed Graphs with Controlled Relationships between Classes, Attributes, and Topology. arXiv (2021)
6. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph Attention Networks. In: ICLR (2018)

7. Wang, B., Wang, C., Feng, H.: Fastsng: The fastest social network dataset generator. In: WWW (2021)
8. Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L., Koutra, D.: Beyond homophily in graph neural networks: Current limitations and effective designs. NeurIPS (2020)