Explaining GNN Biases

Steve Azzolin
University of Trento, DISI
steve.azzolin@studenti.unitn.it
221251

Abstract

Final project for the course *Advanced Topics in Machine Learning and Optimization* supervised by Antonio Longa and Prof. Andrea Passerini. The project description can be found at `https://disi.unitn.it/~passerini/teaching/2021-2022/AdvancedTopicsInMachineLearning/projects/projects.pdf`

1. Introduction

Graph Neural Networks (GNN) have becoming increasingly popular for tasks over graph-structured data, such as node classification (e.g. predicting a property of a user in a social network), graph classification (e.g. predicting whether a molecule is mutagenic or not), and link prediction (e.g. to predict whether a relationship may exist between two entities, like users in a social network). However, despite this widespread use, little is known about why a GNN predicts a certain value. Some recent works try to tackle the problem of explaining GNN predictions, but we are still far from understanding the true behaviour of them. This work represents the first step towards categorizing biases and mining patters in GNN explanations, which will be further explored in my MSc Thesis. My contribution for this project is composed of:

- Developing a modular and scalable codebase to train Graph Neural Networks and to run off-the-shelf explainers;
- Training a GCN and a GAT model over three popular datasets for node classification achieving near state-of-the-art performances and extracting local explanations for each node;
- Devising a method to visualize in a compact way the explanations, and try to extract patterns.

2. Background

2.1. Graph Neural Networks

The standard way of learning over graphs before the Graph Neural Network revolution was following the classical Machine Learning pipeline, i.e. extract hand-crafted features, feature engineering, and finally train a shallow model such as SVMs. In addition, to account for the topological structure of the data in the loss function to be optimized, some techniques use a graph Laplacian regularization term in the loss function (Kipf and Welling, 2017).
The Graph Neural Network model was first introduced and formalized by (Scarselli et al., 2009). Then, (Kipf and Welling, 2017) presented a scalable approach for semi-supervised learning which contributed in making GNNs the preferred paradigm for graph-structured data. Generally, we can describe a GNN model by:

$$h_u^{k+1} = UPDATE^k(h_u^k, m_{N(u)}^k) \tag{1}$$

$$m_{N(u)}^k = AGGREGATE^k(\{h_v^k, \forall v \in N(u)\}) \tag{2}$$

where h_u^0 represents the initial node features of node u, *UPDATE* and *AGGREGATE* are arbitrary differentiable functions, and $N(u)$ represents the neighborhood of node u. Depending on the implementation of *UPDATE* and *AGGREGATE* different variants can be implemented (Kipf and Welling, 2017) (Gilmer et al., 2017) (Veličković et al., 2018). The GCN architecture (Kipf and Welling, 2017), specifically, implements the following transformations:

$$h_u^{k+1} = W^T \sum_{v \in N(u) \cup \{i\}} \frac{e_{v,u}}{\sqrt{d_v d_u}} h_u^k \tag{3}$$

where W are parameters learned via backpropagation, $d_i = 1 + \sum_{v \in N(u)} e_{v,u}$, and $e_{v,u}$ denotes edge weight from node v to node i (by default $e_{v,u}$ equals 1). For performance reasons, it is often convenient to represent the computation in form of matrix multiplications. To account for the diffusion process involving the aggregation of progressively distant nodes, the following formulation can be used instead:

$$H^{k+1} = D^{-1/2} A' D^{-1/2} H^k W \tag{4}$$

where $A' = A + I$ is the adjacency matrix with self-loops, D is the diagonal degree matrix, and $D^{-1/2} A' D^{-1/2}$ represents the symmetric normalized adjacency matrix which normalizes the contribution of nodes based on their connectivity.
GAT (Veličković et al., 2018), on the other hand, can be seen as a generalization of the GCN architecture, in which the edge weight is learned via a set of trainable parameters.

2.2. Explainability in Graph Neural Networks

One of the most cited works on XAI for GNNs is **GNNExplainer** (Ying et al., 2019). GNNExplainer is a post-hoc explanation technique which tries to find a soft mask over the edges and a subset of features of the input nodes by maximizing the mutual information between the predicted labels and the distribution of possible subgraphs and subfeatures. Given a node v, the goal is to identify a subgraph $G_s \subseteq G_c : v \in G_s$ and the associated subset X_s^F of input features that are important for the GNN's prediction Y.

$$\begin{aligned} &max_{G_s} MI(Y, (G_s, X_s)) \\ &\quad = H(Y) - H(Y | G = G_s, X = X_s^F) \end{aligned} \tag{5}$$

MI quantifies the change in the probability of prediction when the input graph is limited to the explanation subgraph

G_s and its node features are limited to X_s^F. Details about the optimization framework can be found in (Ying et al., 2019).
Another popular work is **PGExplainer** (Luo et al., 2020). PGExplainer shares the same settings as GNNExplainer, presented in Equation (5), but instead of optimizing a soft mask over the features and the edges of the input graph, it learns the parameters of a MLP which takes as input the representations of two nodes and returns the likelihood that the edge connecting them will be present in the final explanation. By training this MLP over all input nodes, PGExplainer is able to explain each individual node with a global view of the model to be explained, a property that GNNExplainer lacks since the soft mask is learned independently for each node to explain. Another major different with respect to GNNExplainer is that PGExplainer cannot learn the node-level features relevant for the explanation.
A broader overview of XAI approaches for graphs is (Yuan et al., 2021). Another useful resource for staying up-to-date is the following[1]. In this work, in accordance with my supervisor, PGExplainer for node classification was used.

3. Codebase

In this section, the organization of the codebase will be briefly described. The final goal was to develop a modular and a scalable codebase for running experiments on different architectures and datasets. In order to accomplish this, Python modules encapsulating common parts into reusable functions were developed. Figure 1 presents a compact UML diagram of the classes. The different implementations of the functions *train(), train_epoch(),* and *predict()* reflect the different characteristics of the task, i.e. node vs graph classification, but also tackle performance issues, for example for big graphs like REDDIT in which an ad-hoc *NeighborSampler* is needed. In case one wants to add the support for a new dataset, for example for semi-supervised node classification, and the relative architectures:

- define a *FrameworkNEWDATASET*, inheriting from the *SemiSupervisedFramework* class, specifying only how to load the new dataset;
- define the structure of the model in a class named like *GCN_NEWDATASET*;
- run the code and save the resulting model.

The full codebase can be found here[2].

4. Node Classification Experiments

Once the codebase has been developed, a GCN and a GAT model were trained for the datasets present in Table 1, which shows also the final performances of the models. Every model was trained independently and the hyper-parameters were found via manual random search. Details about hyper-parameters can be found in the code. In order to track the performances of different configurations and different architectures, the tool Weights & Biases[3] was used. Finally, Figure 2 shows both the accuracy and the loss for training and validation splits in the CiteSeer dataset.

Dataset	GCN	GAT
CORA	0.81	0.81
REDDIT	0.92	0.92
CiteSeer	0.70	0.71
BAShapes	0.99	0.99

Table 1: Test accuracy of the trained models. The results are overall in line with the published relative performances

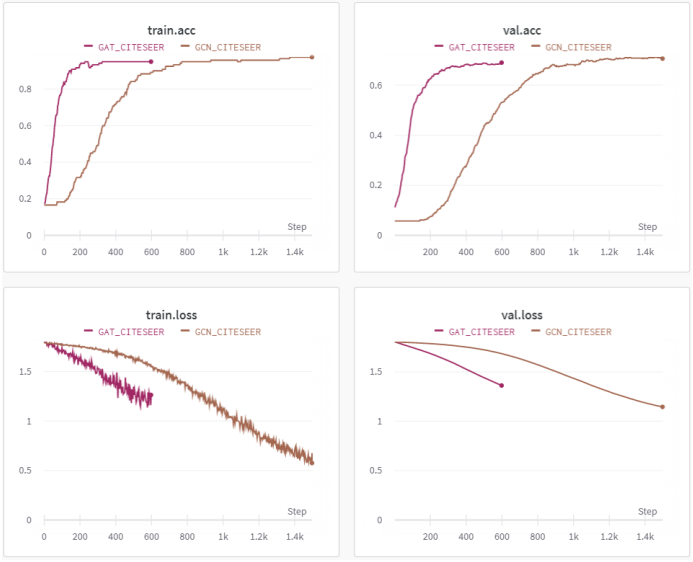


Figure 2: Training and validation curves for the CITESEER dataset. The plots are exported from Weights & Biases

5. Explaining the predictions

PGExplainer was run over the previously trained models in order to get a local explanation for each correctly classified node in each dataset. Unfortunately, due to computational complexity issues and in accordance with my supervisor, the explanations for the REDDIT dataset (which is usually used for large scale GNN benchmarks) were not extracted. PGExplainer comes with several hyper-parameters to be tuned, for example:

- `num_epochs`: Number of epochs to train the explainer
- `num_hops`: Number of hops to extract neighborhood of target node
- `top_k`: Number of edges to include in the final explanation

In my experiments, `num_hops` was kept fixed at `3`, `top_k` was not used since the entire raw soft mask over the edges was saved, and `num_epochs` was hand-tuned for each

[1] `https://github.com/flyingdoog/awesome-graph-explainability-papers`
[2] `https://github.com/steveazzolin/advanced_machine_learning`
[3] `https://wandb.ai/site`

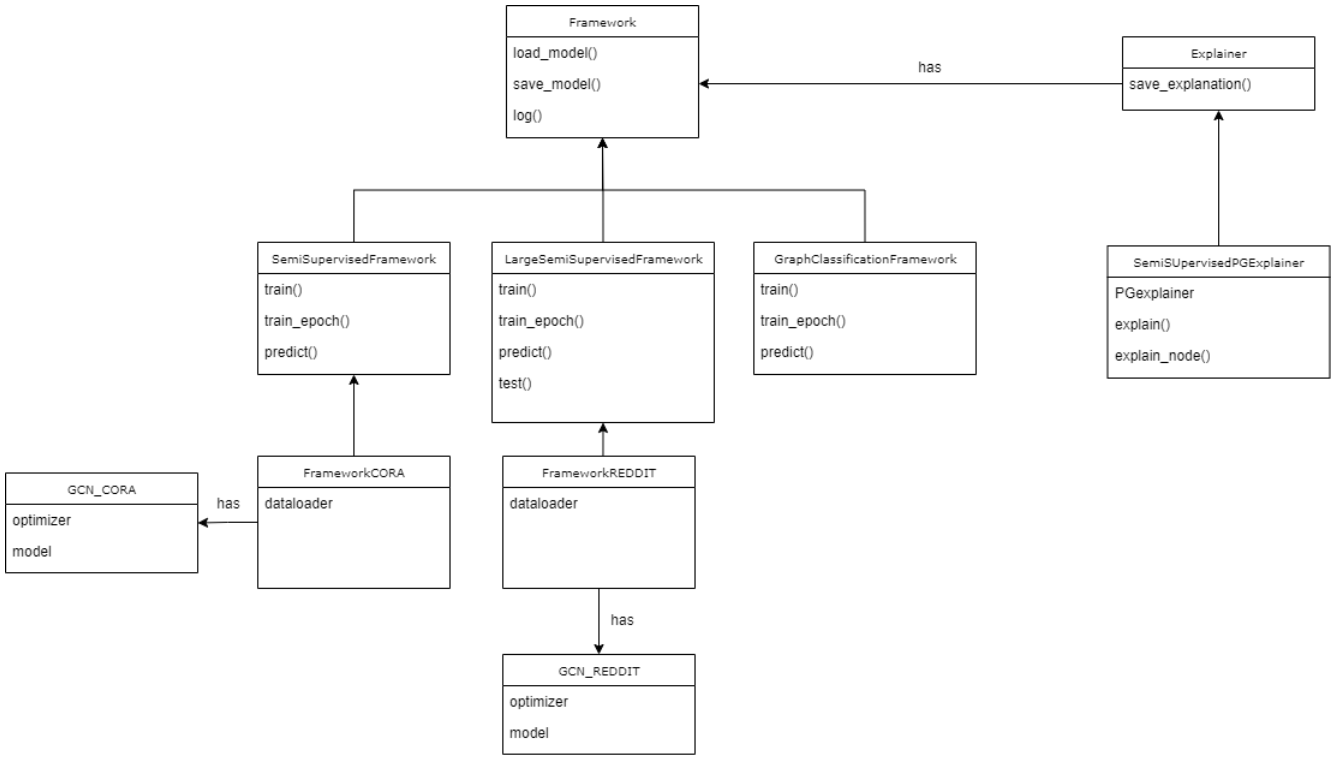


Figure 1: Simplified UML diagram of the codebase

model and for each dataset in order to reach a plateau in the training loss (overall, between 20 and 40 epochs). The resulting local explanations are converted into the *NetworkX*[4] format and saved with the *Joblib* library[5]. Note that for real world datasets, tuning the hyper-parameters of the explainer is non-obvious, since no reference explanation pattern is available.

6. Mining explanation patterns

6.1. Visualizing explanations

One of the first steps to process the raw local explanations was to prune from the returned `num_hops` neighborhood the irrelevant edges. Since PGExplainer returns a confidence score for each edge in this neighborhood, this translates into selecting an appropriate threshold for cutting irrelevant edges. The most naive way of finding such threshold is to select it manually by inspecting the score distribution over the entire dataset. This assumes that the regions of irrelevant and relevant edges is well separated. To validate (or not) this hypothesis, figure 4 shows the overall distribution of edge scores for the different datasets, and as can be noted it is not possible to select a hard threshold for all explanations. The alternative approach used in this work resembles the elbow method, i.e. find a threshold for each local explanation, which is selected as the first ordered value which is different enough with respect to the previous values. Specifically, in these experiments, the threshold was set as $weights_i$ when:

$$d_j \geq \frac{K}{100} * d_{j-1} + d_{j-1} \tag{6}$$

where $d_j = weights_{i-1} - weights_i$, $weights$ being the vector of edge weights ordered in descending order, and K being a parameter set to 10 after a manual inspection of the results over the synthetic dataset BAShapes. A visual representation of such approach is shown in Figure 3.

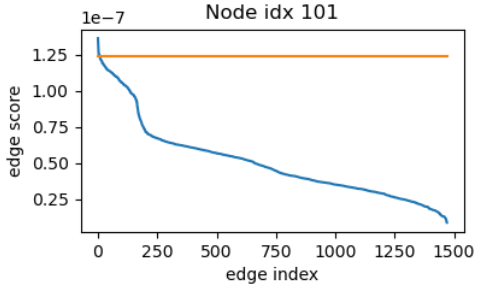


Figure 3: An example of the resulting threshold for the elbow-like approach. The plot is referred to one sample from the training set of BAShapes, where the weights have been sorted in descending order. The orange horizontal line corresponds to the selected threshold for this specific sample.

The next step was to plot such thresholded explanations. Figure 5a shows the original raw explanations, Figure 5b shows the same explanations after cutting the irrelevant edges with the method presented above, while Figure 5c presents the final post-processed explanations, in which all nodes and edges not belonging to the same connected component of the target node to be explained were cut. From now on, the explanations after applying the threshold and after cutting the irrelevant connected components will be referred as **post-processed explanations**. As can be seen

[4] `https://networkx.org/`
[5] `https://joblib.readthedocs.io/en/latest/`

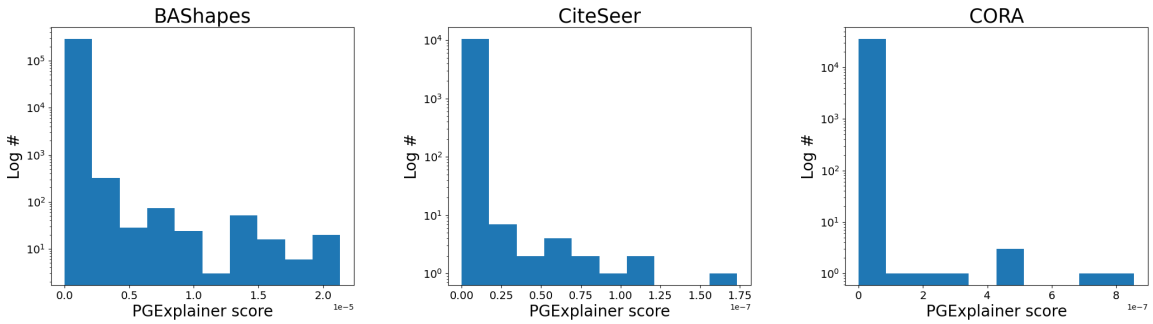


Figure 4: Histograms of edge scores for BAShapes, CORA, and CiteSeer obtained by PGExplainer over the relative GCN model. Results for GAT share the same behaviour

from the figures above, especially for synthetic datasets in which the prediction pattern is known, many local explanations share the very same topology. Thus, in order to simplify the visualization and to reduce the complexity of successive experiments, an isomorphism test is run for each ground truth class, in order to only keep the unique explanations per class. Table 2 compares the number of explanations before and after this isomorphism filtering, along with the execution time. Indeed, isomorphism is well known for being a *NP-complete* problem, but the low cardinality of the post-processed explanations make the full isomorphism test feasible, even with my standard commercial laptop as shown in the table. Figure 5d shows the first six non-isomorphic explanations for the class 1 of the BAShapes dataset, while Figures 7-12 present the plot of the first five non-isomorphic post-processed explanations divided by class for all datasets.

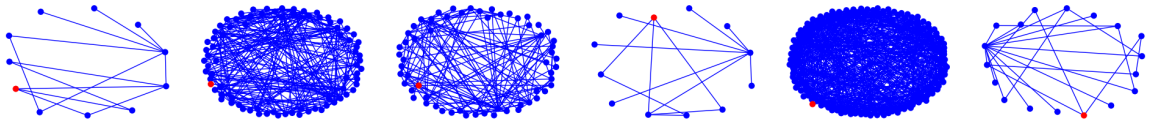
One caveat of the isomorphism test is that it is not noise tolerant. For example, two graph differing by only one edge are considered non-isomorphic even if the differing edge does not impact the overall topology of the two graphs, which is shared. An example of such case can be found in the first two graphs of Figure 5d. To avoid such cases, and to create a more noise-tolerant filtering, a greedy recursive algorithm based on graph Edit distance was devised. In particular, first all pairwise Edit distances are computed for the explanations, then for each iteration the graph which is most similar to all other graphs is selected to be moved in the final set F of explanations (in practical terms, select the graph with the minimum average Edit distance between all other graphs). Then, in order to exclude graphs that share a very closed topology with the selected graph, all explanations with an Edit distance of maximum K (K set to 4 for these experiments) are removed. This algorithm is iterated until no more explanations are left to be chosen, or until a minimum number of explanations are moved to F. However, with the standard setting of the algorithm it is difficult to have control over the selected graphs, which may result in a final set not readily interpretable. For example, the first resulting explanation of Figure 5e, despite having a low Edit distance w.r.t. the expected house pattern, at first glance it does not really look like a house. To gain more control over the selected graphs, it is possible to hard include one or more explanations in F, in this way all the successive explanations will be later included following the same procedure, but with an initial fixed starting set F. This specific extension is left as future work.

	CiteSeer	**CORA**	**BAShapes**
# train samples	120	140	448
# explanations	117	138	444
# post processed	67	43	278
# non isomorphic	39	32	17
Isom. filter time (s)	2.0	2.2	2.8

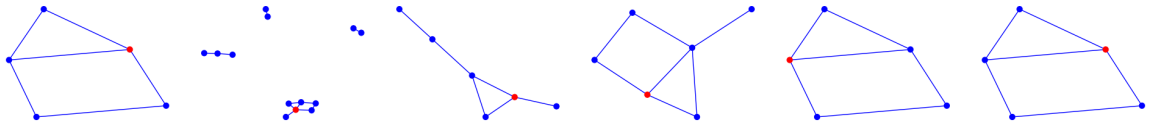
Table 2: Statistics on the number of examples and number of explanations before and after the post-processing operations described in Section 6.1, along with the total time for running the isomorphism filtering. Results are referred just to the GCN model and to the train split of the relative dataset. The reason for the mismatch between number of examples and number of explanations is due to the removal of incorrectly predicted samples, whereas the mismatch between the second and the third row is due to empty post-processed graphs, which for BAShapes affects mostly the class 0

6.2. Pattern mining

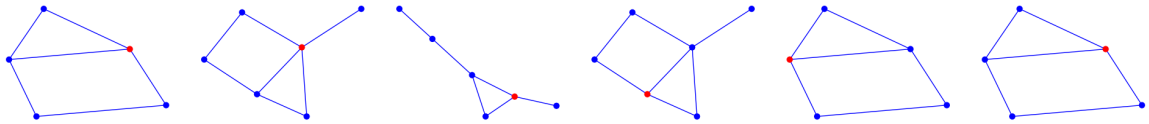
Another line of work consisted in trying to extract patterns from the local explanations, for example to check for similarities in the topology of explanations belonging to a specific class. The first approach was oriented towards extracting graph level hand-crafted features (like average clustering coefficient, average betweenness centrality, average degree connectivity) and to check whether the feature space shows some structure. However, as can be seen from the 2D PCA-reduced embedding presented in Figure 6 for the GCN model, no clear clusters seem to appear. The GAT's embedding space is different, but does not present any structure as well. This might be either caused by dimensionality reduction, which preserves around the $60 - 70\%$ of the total variance, by the features being not discriminative enough, or by the fact that there could not be an underlying class-specific structure in the explanations. In order to exclude the first two scenarios, a specific **graph explanation classification task** was devised with the goal of learning via backpropagation a meaningful 2D latent representation that can classify correctly each explanation as be-



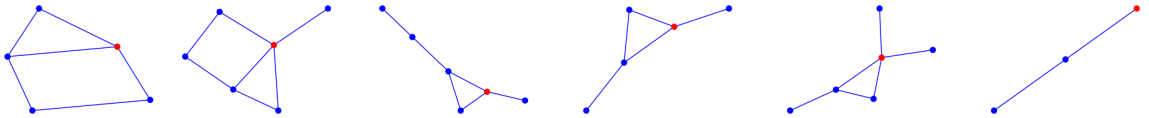
(a) Raw explanations



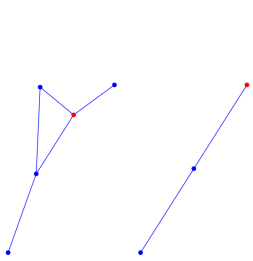
(b) Explanations filtered with the elbow-like threshold method



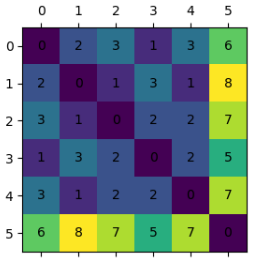
(c) Post-processed explanations



(d) Post-processed non-isomorphic explanations filtered by the isomorphism test



(e) Non-isomorphic explanations after Edit filtering

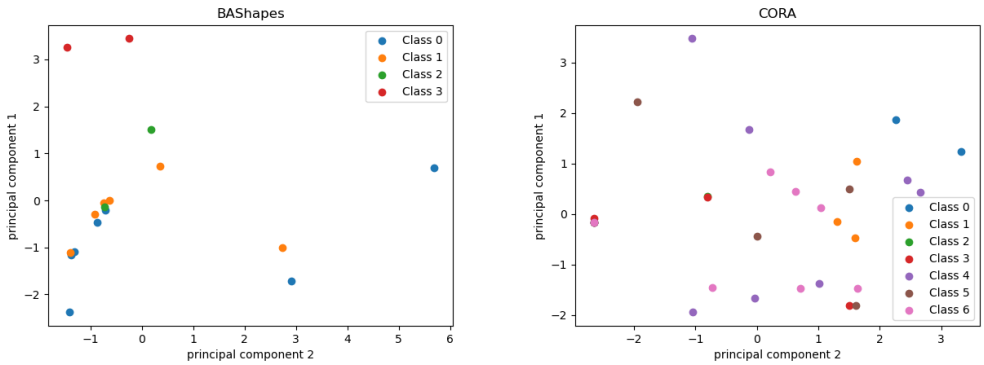


(f) Pairwise edit distances referred to Figure d

Figure 5: All explanations are referred to the same train samples belonging to class 1 of BAShapes, extracted for the GCN model. Figure a, b, and c are referred to the first six explanations, while Figure d is referred to all non-isomorphic explanations

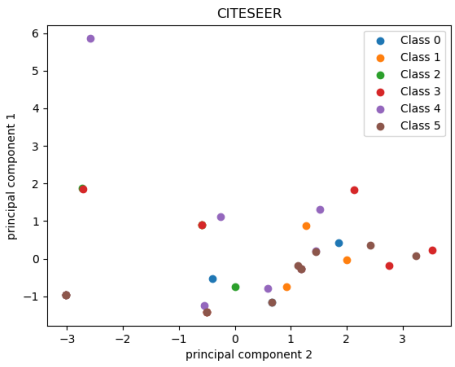
longing to its respective class. To do so, a 2-layers GIN (Xu et al., 2019) network was trained using as dataset the non-isomorphic post-processed explanations for each dataset independently. In addition, a 2-layer GINE (Hu* et al., 2020) was trained with the same setting of the above GIN, with the only difference of using the normalized PGExplainer scores as edge attributes. Isomorphic explanations were removed since they do not carry any additional information and to simplify the training. Each example thus corresponds to a local explanation, and its ground truth corresponds to the ground truth of the target node for which the explanation was computed. The hyper-parameters of the network were hand-tuned[6] in order to favor overfitting. The resulting Macro-F1 scores are shown in Table 3. For the experiment considering only the topology of the explanation, the Macro-F1 scores are quite low, meaning that GIN is not able to extract a meaningful embedding space. This behaviour is either caused by the absence of class specific patterns in the explanations (which was actually quite expected by manually inspecting the few explanation patterns in Figures 7-12), or by the training data scarcity as reported in the fourth row of Table 2, which might have limited the learning of the model. However, since the network is not even able to classify well the explanations in the overfitted training set, I believe that the reason is solely due to the absence of class-specific explanation patterns in the tested datasets. On the other hand, when considering the PGExplainer's scores as edge attributes, the network is better at extracting patterns, meaning that these scores can play an important role in discriminating the activation patterns of different architecture types. Indeed, as a final experiment, instead of classifying each explanation into the class of the node for which the explanation was extracted, all explanations are collected together and classified as belonging to the model they were extracted from, namely GCN or GAT. The results reported in Table 4 show that edge attributes play an important role for correctly predicting the origin of an explanation, and that for certain dataset the explanation patterns are really typical of certain architectures.

[6] hidden_layers_dimensionality=[20, 2], dropout=0, lr=0.01, weight_decay=0, num_epochs=700



(a)

(b)



(c)

Figure 6: 2D hand-crafted embedding space obtained via PCA

Net.	Model Expl.	Dataset	Train Macro-F1
GIN	GCN	BAShapes	0.29
		CORA	0.12
		CiteSeer	0.11
	GAT	BAShapes	0.26
		CORA	0.14
		CiteSeer	0.10
GINE	GCN	BAShapes	0.63
		CORA	0.10
		CiteSeer	0.11
	GAT	BAShapes	0.96
		CORA	0.18
		CiteSeer	0.33

Table 3: Macro-F1 scores of a GIN and a GINE network trained on the graph explanation classification task. GINE uses the PGExplainer's scores as edge attributes

Net.	Dataset	Test Macro-F1
GIN	BAShapes	0.71
	CORA	0.52
	CiteSeer	0.55
GINE	BAShapes	1.0
	CORA	0.50
	CiteSeer	0.93

Table 4: Macro-F1 scores of a GIN and a GINE network trained on the model-level graph explanation classification task, i.e classify each explanation into the model they were extracted from. GINE uses the PGExplainer's scores as edge attributes

7. Conclusions

This work focused on developing a modular codebase for running experiments on multiple datasets with different GNN architectures. In addition, the local explanations for each node were extracted. On top of these explanations, different experiments aimed at mining both the inter-architecture patterns and intra-architecture patterns were conducted. The results showed that the topology alone does not present any explanation pattern, whereas the incorporation of the explainer's edge scores seems to reveal stronger patterns. However, the setting of these experiments was limited, so further improvements are needed to validate these preliminary results over many more architectures, datasets, and explainers. Some possible improvement are listed in the Section below.

8. Future works & proposals

- The modular nature of the codebase would allow the incorporation of an Auto-ML tool for automatic hyper-parameter search. An example of such tool is Optuna(Akiba et al., 2019). This can automatize the procedure of adding new networks, since it would remove the need of manually looking for hyper-parameters, which is something slow and tedious;

- Taking inspiration from several papers about Reinforcement Learning-based explainer (Shan et al., 2021) and subgraph sampling (Zeng et al., 2019) (Zeng et al., 2021), this proposal is about devising a combination of both that could give rise to a novel explainer which founds on both techniques. Specifically, (Shan et al., 2021) trains a RL agent to select the relevant nodes/edges in a graph for a trained GNN, by maximizing the mutual information like in Equation 5. GraphSAINT (Zeng et al., 2019), instead, aims at scaling GNNs to large graphs by sampling the training graphs in such a way to loose little information. The samplers introduced by the paper are shallow and simple, like random sampler or sampler based on random walks. Another interesting work on subgraph sampling is (Zeng et al., 2021), which shows that by sampling a connected subgraph around the node to be classified, it is possible to use much deeper networks without incurring in over-smoothing. Again, the subgraph sampler presented in the paper is shallow, based on heuristics. The proposal is to formalize the explanation problem as finding, during training, the minimum subgraph around the node to be classified which allows for a correct prediction. In practical terms, we would have the classical GNN architecture which is trained over the subgraph sampled by a RL agent, which in turn is trained to sample a connected subgraph around the target node, so to minimize the cross entropy of the node prediction w.r.t. the ground truth. Since the training of the model is coupled with the explanation component, it would give rise to a gray-box model;

- The graph classification tasks of Section 6.2 help in shedding some light over the biases of different architectures, but little is known about the biases of different explainers. Maybe an adaptation of the graph classification task, in which for a single model and a single dataset we train a GNN for predicting the explainer that generated the explanation, could give some additional insights.

- In order to operate over real-world datasets for which we have a reference ground truth of the expected explanation, I was wondering to use image datasets annotated both with the class of the subject and its semantic segmentation mask. In this way, by decomposing the image into connected superpixels, we could classify the graph into the respective class, using as reference pattern for the explanation the superpixels which are in the semantic segmentation of the subject;

- The recursive algorithm based on graph Edit distance for finding prototypical explanations presented in Section 6.1 is inherently greedy. Indeed, at each iteration, the explanation with the lowest average distance to all other explanations is selected. However, it is not guaranteed that this results in an optimal solution. An alternative approach would be to define an algorithm based on backtrack and a cost function based on the sum of the minimum Edit distances between prototypes and original explanations. This would guarantee to find the set F minimizing the residual;

- The methods for cutting irrelevant edges presented in Section 6.1 are both suboptimal. Indeed, they do not consider the topology of the explanation. For example, they might preserve a disconnected edge far apart from the target node if PGExplainer assigned to it an high score erroneously. The issue was partially tackled by filtering out all connected components not containing the target node. However, a single algorithm for cutting the relevant edges around the target node may be beneficial.

9. Bibliographical References

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry.

Hu*, W., Liu*, B., Gomes, J., Zitnik, M., Liang, P., Pande, V., and Leskovec, J. (2020). Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks.

Luo, D., Cheng, W., Xu, D., Yu, W., Zong, B., Chen, H., and Zhang, X. (2020). Parameterized explainer for graph neural network.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80.

Shan, C., Shen, Y., Zhang, Y., Li, X., and Li, D. (2021). Reinforcement learning enhanced explainer for graph neural networks. In A. Beygelzimer, et al., editors, *Advances in Neural Information Processing Systems*.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019). How powerful are graph neural networks? In *International Conference on Learning Representations*.

Ying, R., Bourgeois, D., You, J., Zitnik, M., and Leskovec, J. (2019). Gnnexplainer: Generating explanations for graph neural networks.

Yuan, H., Yu, H., Gui, S., and Ji, S. (2021). Explainability in graph neural networks: A taxonomic survey.

Zeng, H., Zhou, H., Srivastava, A., Kannan, R., and Prasanna, V. (2019). Graphsaint: Graph sampling based inductive learning method. 07.

Zeng, H., Zhang, M., Xia, Y., Srivastava, A., Malevich, A., Kannan, R., Prasanna, V., Jin, L., and Chen, R. (2021). Decoupling the depth and scope of graph neural networks. In A. Beygelzimer, et al., editors, *Advances in Neural Information Processing Systems*.

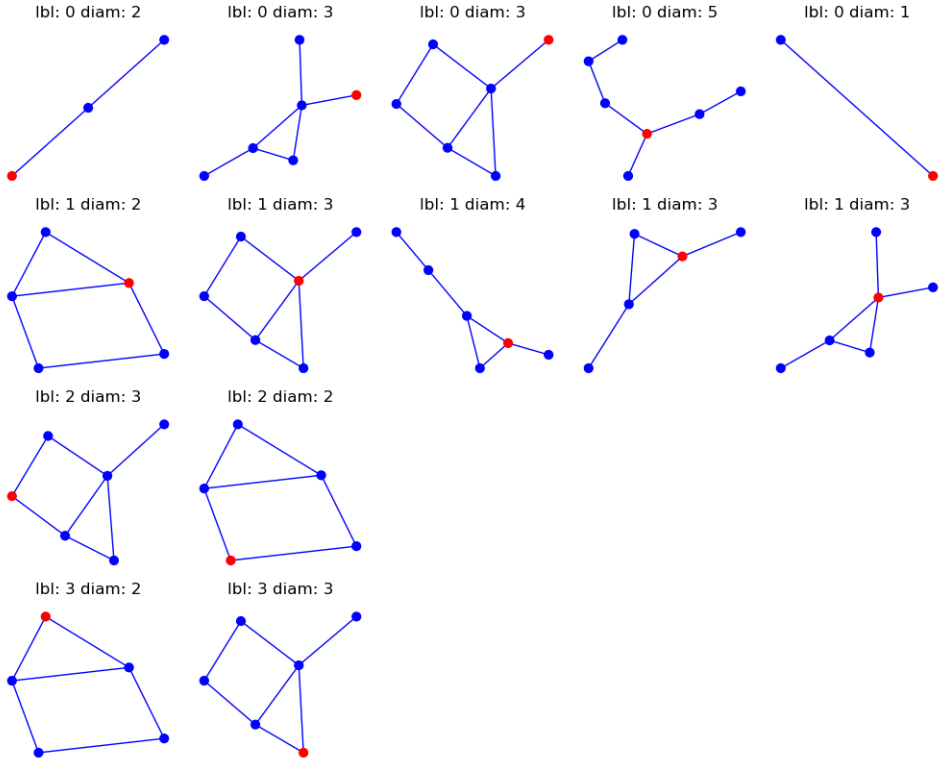


Figure 7: First five non isomorphic post-processed GCN explanations for BAShapes. The plot is annotated with the ground truth label and with the diameter of the explanatory graph

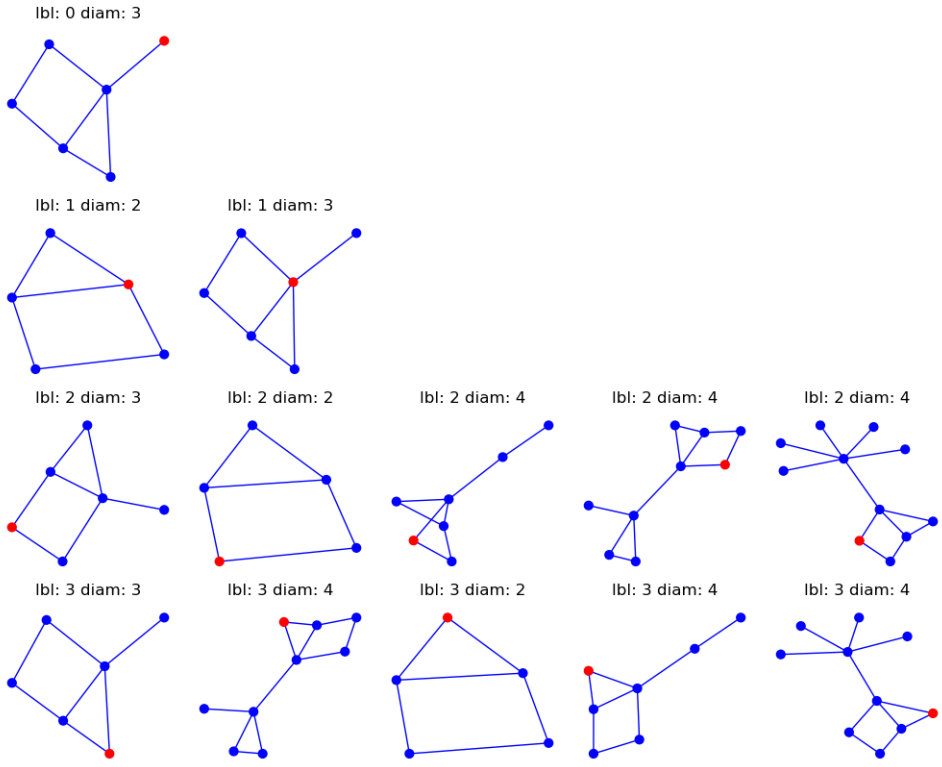


Figure 8: First five non isomorphic post-processed GAT explanations for BAShapes. Note that for class 0, only one pattern survived from the connected component filtering. The reason is that after cutting irrelevant edges, a plethora of small connected components is created, which do not contain the target node

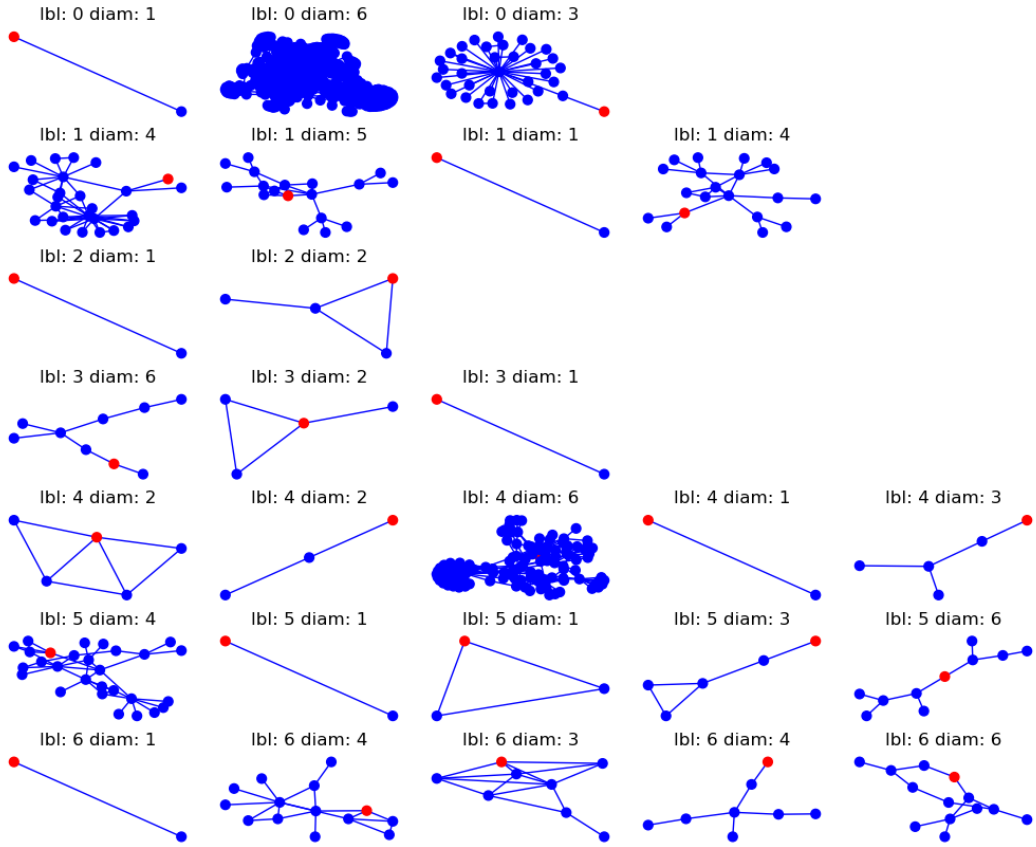


Figure 9: First five non isomorphic post-processed GCN explanations for CORA. The plot is annotated with the ground truth label and with the diameter of the explanatory graph

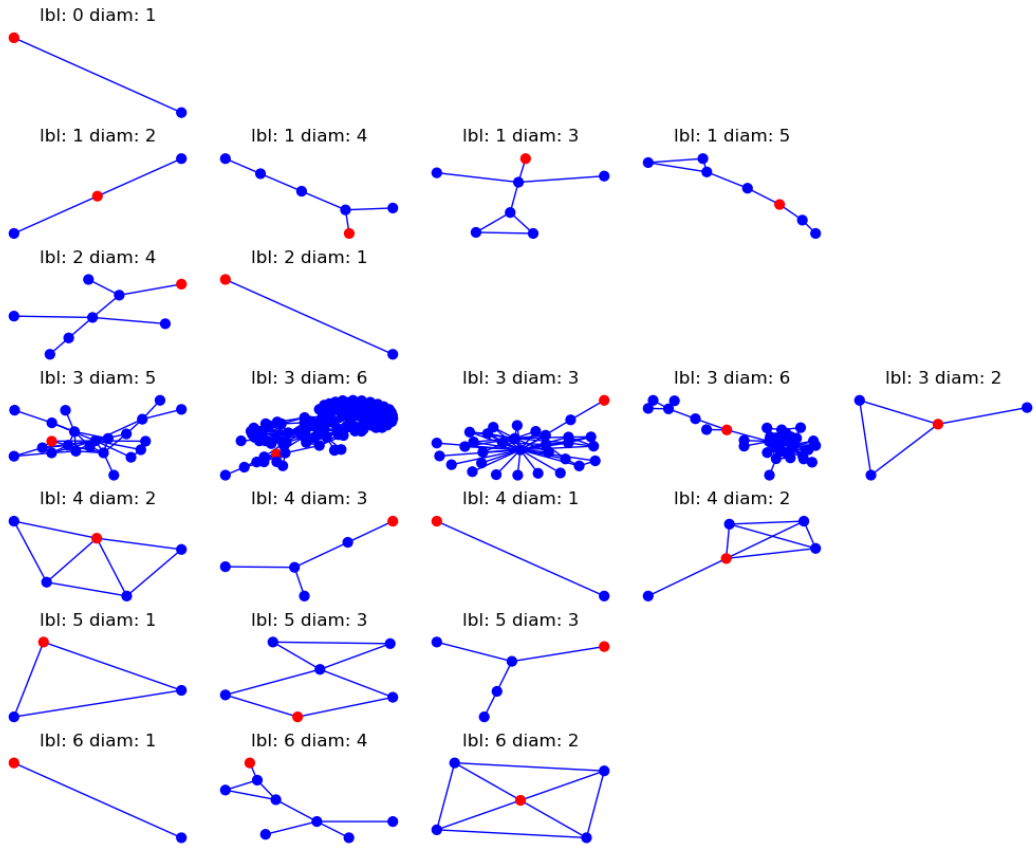


Figure 10: First five non isomorphic post-processed GAT explanations for CORA. The plot is annotated with the ground truth label and with the diameter of the explanatory graph

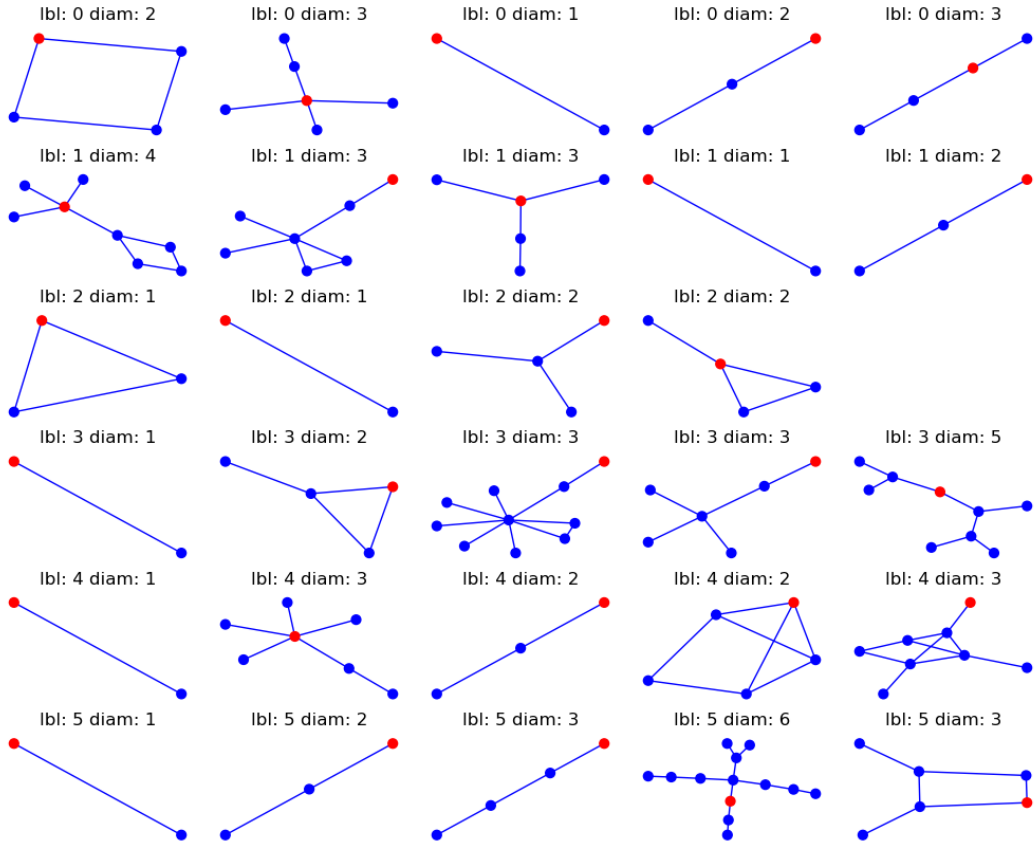


Figure 11: First five non isomorphic post-processed GCN explanations for CiteSeer. The plot is annotated with the ground truth label and with the diameter of the explanatory graph

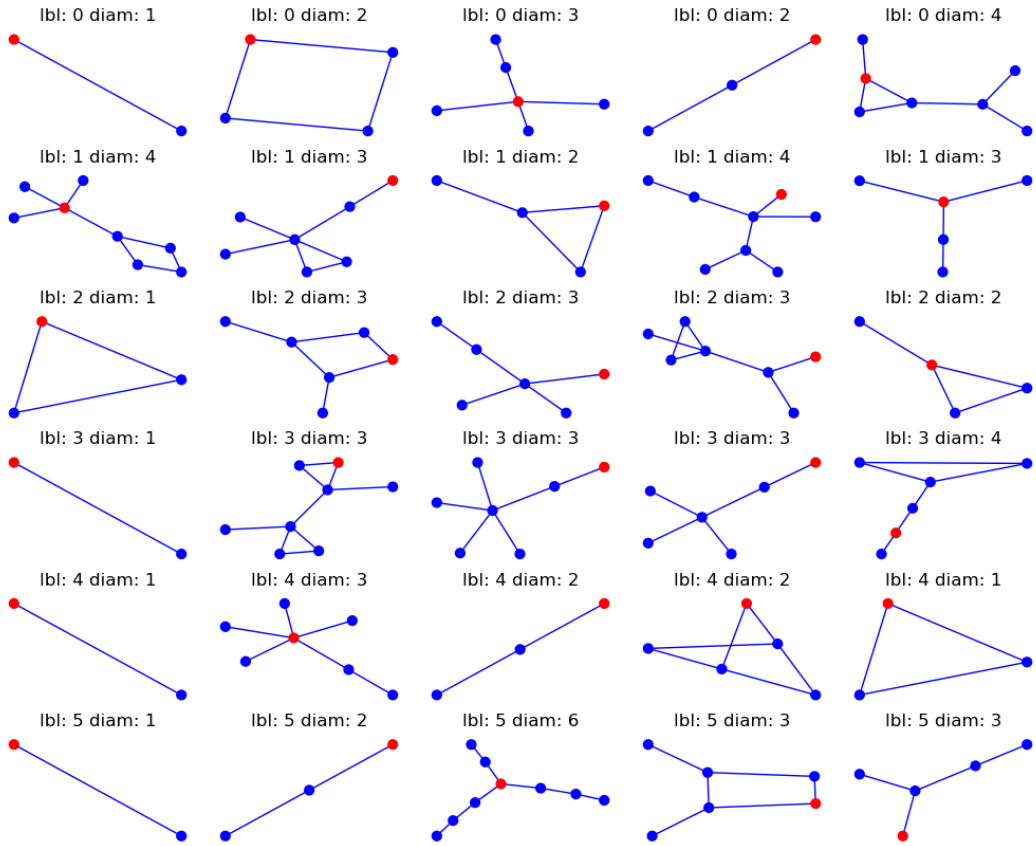


Figure 12: First five non isomorphic post-processed GAT explanations for CiteSeer. The plot is annotated with the ground truth label and with the diameter of the explanatory graph