UNDER REVIEW

RESCIENCE C

Replication / ML Reproducibility Challenge 2022

[Re] On Explainability of Graph Neural Networks via Subgraph Explorations

Yannik Mahlau[1], Leo Kayser[1], and Lukas Berg[1]
[1]Leibniz Universität Hannover, Hannover, Germany

Edited by
(Editor)

Received
01 November 2018

Published
–

DOI
–

Reproducibility Summary

Scope of Reproducibility – Yuan et al. claim their proposed method SubgraphX achieves (i) higher fidelity in explaining models for graph- and node classification tasks compared to other explanation techniques, namely GNNExplainer. Additionally, (ii) the computational effort of SubgraphX is at a "reasonable level", which is not further specified by the original authors. We define this as at most ten times slower than GNNExplainer.

Methodology – We reimplemented the proposed algorithm in PyTorch. Then, we replicated the experiments performed by the authors on a smaller scale due to resource constraints. Additionally, we checked the performance on a new dataset and investigated the influence of hyperparameters. Lastly, we improved SubgraphX using greedy initialization and utilizing fidelity as a score function.

Results – We were able to reproduce the main claims on the MUTAG dataset, where SubgraphX has a better performance than GNNExplainer. Furthermore, SubgraphX has a reasonable runtime of about seven times longer than GNNExplainer. We successfully employed SubgraphX on the Karate Club dataset, where it outperforms GNNExplainer as well. The hyperparameter study revealed that the number of Monte-Carlo Tree search iterations and Monte-Carlo sampling steps are the most important hyperparameters and directly trade performance for runtime. Lastly, we show that our proposed improvements to SubgraphX significantly enhance fidelity and runtime.

What was easy – The authors' description of the algorithm was clear and concise. The original implementation is available in the DIG-library as a reference.

What was difficult – The authors performed extensive experiments, which we could not replicate in their full scale due to resource constraints. However, we were able to achieve similar results on a subset of the datasets used. Another issue was that despite the original code of the authors and datasets being publicly available, there were many compatibility issues.

Communication with original authors – The original authors briefly reviewed our work and agreed with the findings.

Copyright © 2023 Y. Mahlau, L. Kayser and L. Berg, released under a Creative Commons Attribution 4.0 International license.
Correspondence should be addressed to Yannik Mahlau (yannik.mahlau@stud.uni-hannover.de)
The authors have declared that no competing interests exist.
Code is available at https://github.com/ymahlau/subgraphx – DOI https://zenodo.org/badge/latestdoi/637344138. – SWH swh:1:dir:439719e0ad99cbd3d980619c24dec1744b408dd0.
Open peer review is available at https://openreview.net/forum?id=zKBJw4Ht8s.

1 Introduction

Graph Neural Networks (GNN) achieve increasingly better results in the three main tasks regarding graphs: node classification, graph classification and link prediction. However, the lack of interpretability of machine learning models is a barrier hindering their adoption [1]. Therefore, a post-hoc explanation algorithm for GNNs would be desirable. There already exist some techniques for explaining black-box GNNs, which mainly focus on explanations by edge- or feature masks, for example GNNExplainer [2].
Empirical studies already showed that an explanation by a subgraph is more interpretable to humans than a soft edge- or feature mask [3]. Therefore, Yuan et al. [4] proposed an algorithm named SubgraphX for explaining a GNN by the most important subgraph of the graph instance. To briefly outline the SubgraphX algorithm:

1. Start with the full graph instance if the GNN performs graph classification. In the case of node/edge classification, one can prune all nodes and edges outside of the receptive field of the GNN for efficiency reasons. Therefore, start with the subgraph containing the k-hop neighborhood around the target node/edge. k denotes the number of layers of the respective GNN model.

2. Perform Monte-Carlo Tree Search (MCTS) [5] with the subgraph identified in step one as root node. Children nodes are subgraphs where a single node was removed from the parent subgraph. Leaf nodes are subgraphs whose sizes are smaller than a constant threshold N_{min}. The score of a node during search is determined by the Upper Confidence Bound (UCB) regarding the Shapley Value [6] of its subgraph. More details about the Upper Confidence Bound are discussed in subsection 7.1. The Shapley Value is approximated by Monte-Carlo sampling. Repeat this step for M iterations.

3. Output the best leaf node found by MCTS.

During MCTS, if the removal of a node splits the subgraph in multiple connected components, only the component including target node/edge is kept. In graph classification, there exists no target node/edge and therefore only the largest component is kept. The Shapley Value of a subgraph $E \subseteq G$ (approximated by Monte-Carlo sampling) is computed as:

$$\phi(E) = \frac{1}{T} \sum_{t=1}^{T} f(S_t \cup E)_y - f(S_t)_y, \quad S_t \subseteq (G \setminus E) \tag{1}$$

where $f(\cdot)_y$ is the model output for class y and T is the number of Monte-Carlo sampling steps. The coalitions S_t are sampled randomly. All nodes, which are not considered in the current sampling step, are excluded from the model input by setting their attributes to a baseline of zero.
The authors also propose MCTS_GNN, a faster variant of SubgraphX. MCTS_GNN determines the score of a subgraph E by the model prediction for that subgraph. This is equivalent to an approximated Shapley Value with just the empty coalition $S_t = \emptyset$ (sampling size $T = 1$):

$$\text{MCTS_GNN}(E) = f(E) - f(\emptyset) \approx f(E). \tag{2}$$

The term $f(\emptyset)$ is the same for all subgraphs and therefore can be omitted in the computation of Monte-Carlo Tree Search.

2 Scope of Reproducibility

As part of the ML Reproducibility Challenge we reimplemented SubgraphX using the PyTorch-Geometric library [7]. Then, we replicated the experiments performed by the authors on a smaller scale, using a subset of the original datasets. The central claims of Yuan et al. [4] regarding SubgraphX are the following:

- SubgraphX achieves higher fidelity scores than the GNNExplainer for graph- and node classification tasks.
- The computation time of SubgraphX is at a "reasonable level" in comparison to GNNExplainer. To specify this exactly, we consider a "reasonable level" to be up to a factor of ten. In other words, the computation of SubgraphX should be at most ten times slower than GNNExplainer.

Note that in the original paper the authors also consider PGExplainer [8] for comparison. We excluded PGExplainer in our experiments as its implementation would exceed any reasonable effort for a student project. In addition to the two main claims, we tested the sensitivity of SubgraphX to its hyperparameters. Next, we tested the transferability of the algorithm to a new datasets. Lastly, we empirically tested two novel optimizations to the SubgraphX algorithm. These novel optimizations improve the runtime drastically while scoring a higher fidelity than the original algorithm.

3 Methodology

Even though Yuan et al. [4] performed experiments on five different datasets, we focused only on the MUTAG dataset [9] due to our limited resources. We chose the MUTAG dataset for two reasons:

- The number of graphs (188) and the size of the graphs (≈18 nodes) is small enough for training a GNN model and computing the explanation with limited resources.
- The paper considered two different GNN models for this dataset, namely a Graph Convolutional Network (GCN) and a Graph Isomorphism Network (GIN), hence in using this dataset we 'kill two birds with one stone'.

We trained both models on an 80% training split (i.e. 150 graphs) and used the remaining 38 graphs as the test set. The structure of the models is the same as described in the appendix of the original paper [4]. Both models achieved a test accuracy of around 92%. Note that in the original paper the authors tested the the explanation performance on all 188 graphs, but this would take too long on our hardware. Therefore, we decided to use the 38 graphs of the test set. Unless specified otherwise, our calculations were performed on a computer with an AMD Ryzen 9 3900 12-Core processor and a NVIDIA GeForce RTX 2080 Super GPU.
Like Yuan et al. [4], we measure the performance of an explanation by the fidelity metric, i.e. the decrease in model confidence when removing the nodes in the explanation. As this is highly dependent on the number of nodes in the explanation, the results are plotted as fidelity versus sparsity curves. Fidelity and sparsity are computed as:

$$Fidelity = \frac{1}{N}\sum_{i=1}^{N} f(G_i)_{y_i} - f(G_i \setminus E_i)_{y_i}, \tag{3}$$

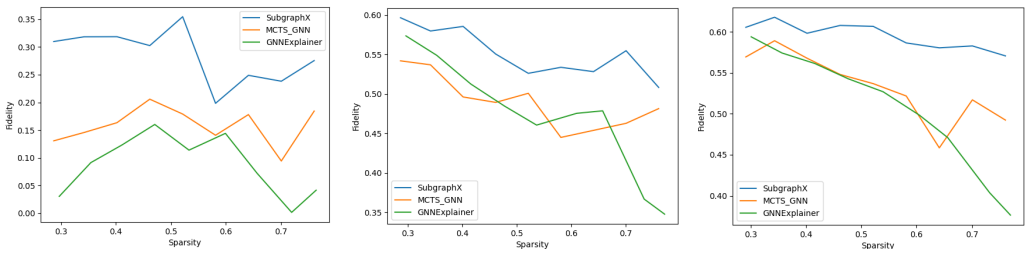
$$Sparsity = \frac{1}{N}\sum_{i=1}^{N}(1 - \frac{|E_i|}{|G_i|}). \tag{4}$$

where N is the number of instances (graphs, nodes or edges depending on the task at hand). G_i is the graph corresponding to a single of those instances. $G_i \setminus E_i$ denotes the graph, where the features of nodes contained in the explanation set E_i are set to zero. The class y_i is the prediction of the model on the original graph G_i.
For GNNExplainer, we used the implementation available in the PyTorch-Geometric library [7]. Even though SubgraphX is available as part of the DIG library, we chose to reimplement it using the PyTorch-Framework. That is, because the DIG-library is currently not compatible with some of the newer version of PyTorch and PyTorch-Geometric. Additionally, this gave us the opportunity to extend the original implementation with further algorithmic optimizations (see Section 7). Our implementation is publicly available at https://anonymous.4open.science/r/subgraphX_reproducibility-8FC8.

4 Replication

4.1 Performance Study

For each explanation method, each graph in the test set and each explanation size $N_{min} \in \{4, \ldots, 12\}$ we collected sparsity, fidelity and execution time for the respective explanation node set. Then for each size N_{min} we calculated the average sparsity and fidelity of all explanations. The results are reported in Figure 1. It is clearly visible that SubgraphX outperforms both GNNExplainer and MCTS_GNN in both experiments. The difference is most prominent for GCN.



(a) GCN on test set (b) GIN on test set (c) GIN on whole dataset

Figure 1. Fidelity vs Sparsity of the explanation algorithms for GCN (a) and GIN (b)/(c) model on the MUTAG dataset.

To address the concern that the small number of graphs may distort the experimental results, we repeated the experiment for the GIN model (which is the faster one of the two) for all 188 graphs in the MUTAG dataset. Even though the individual fidelity scores are different between the two setups, the general trends are the same. Therefore, we conclude that it is valid to restrict our experiments to a subset of the dataset.

4.2 Efficiency Study

To compare the runtime of these algorithms we measured the runtime for explanations of the GIN model. The average runtime is displayed in Table 1.

METHOD	GNNExplainer	MCTS_GNN[1]	SubgraphX
TIME	6.6s	2.9s	45.8s

Table 1. Average execution time of explanation algorithms for GIN model on MUTAG dataset.

[1]Notice that the "MCTS" values in the efficiency study of the original paper [4, Table 2] have nothing to do with MCTS_GNN. This is just an unfortunate naming issue.

MCTS_GNN has an even lower runtime than GNNExplainer with similar or slightly better performance (see Figure 1) Therefore, MCTS_GNN appears to be the best choice if time constraints are important. If performance is more important than time constraints, SubgraphX is suited best. These results adhere to the findings of the original paper.

5 Transfer Study

Yuan et al. [4] already extensively studied the performance of SubgraphX on the graph classification task, which we validated in Section 4. However, they only tested a single node classification dataset. Therefore, we tried to verify the transferability of their results to another dataset, which is the Karate Club Network [10] (See Figure 2). This is a very small dataset consisting of only a single graph, which is perfect for our limited compute capabilities. The experiments in this section were performed on a consumer grade laptop with Intel Core i7-1065G7 (1.30GHz) CPU and NVIDIA GeForce MX330 GPU. The training set is already given and consists of a total of four nodes; One node of each class. Furthermore, we randomly picked 16 validation and 14 test nodes with class distributions as equal as possible.
To generate node features we first trained an unsupervised Node2Vec-Embedding [11]. The model to be analyzed was a two layer Graph Convolutional Network (GCN) [12] with a hidden dimension of 20 and ReLU activation function. The prediction scores were normalized into probabilities by a softmax layer. The model scored 85.7% accuracy on the test set.
We compared the performance of SubgraphX and GNNExplainer. Both explanation methods were evaluated for $N_{min} \in \{4, .., 12\}$ on all nodes in the test set. Figure 3 displays the results of our experiments. It is clearly visible that SubgraphX outperforms GNNExplainer at any sparsity. Especially at high sparsity SubgraphX performs much better than GNNExplainer. This validates the claim of the original authors that SubgraphX can be used in node classification tasks. The average runtime per node (of a single explanation) is 80.5 seconds for SubgraphX and 9 seconds for GNNExplainer. Therefore, according to our own definition, the runtime is still within a "reasonable limit".

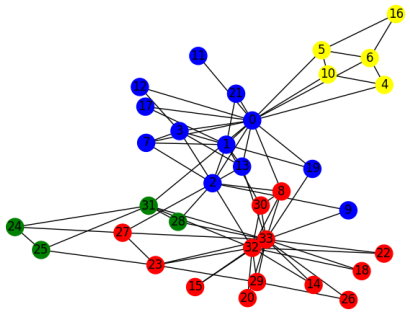


Figure 2. The Karate Club Network.

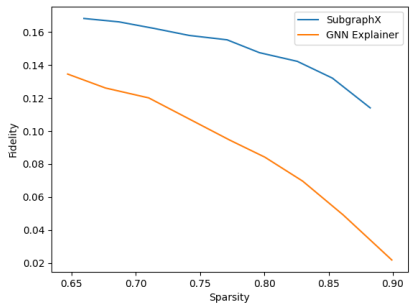


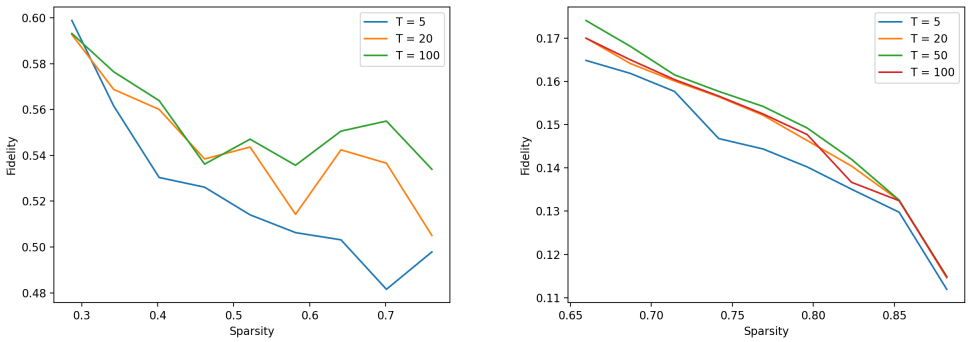
Figure 3. Fidelity vs Sparsity of the explanation algorithms for GCN on the Karate Club Network.

6 Hyperparameter Study

6.1 MCTS iterations and MC sampling steps

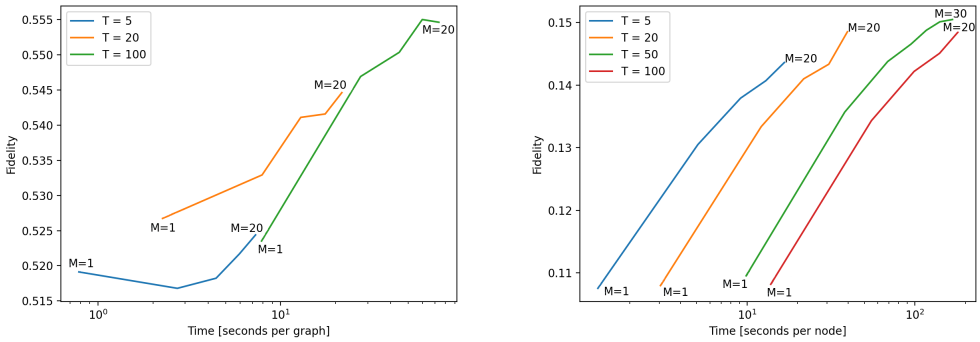
There are two central hyperparameters which directly trade performance for runtime: The number of iteration of MCTS M and the number of Monte-Carlo sampling steps T

(for the shapley value). More iterations and more sampling steps yield better results at the cost of higher runtime.



(a) SubgraphX for GIN model on MUTAG dataset **(b)** SubgraphX for GCN model on Karate Club Network

Figure 4. Effect of Monte-Carlo sampling steps T on fidelity. The number of MCTS iterations M is kept at 20.



(a) SubgraphX for GIN model on MUTAG dataset **(b)** SubgraphX for GCN model on Karate Club Network

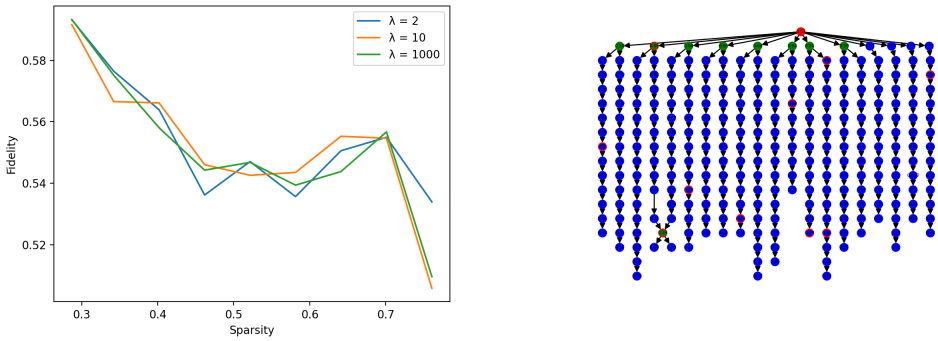
Figure 5. Effect of MCTS iterations M and Monte-Carlo sampling steps T on runtime and fidelity.

The authors chose values of $M = 20$ and $T = 100$ for all of their experiments. Figure 4 shows the effect of T on the fidelity, which is higher for the MUTAG dataset than the Karate Club network. It seems that $T = 20$ is a sufficient value, because the performance is similar at lower runtime. The difference in runtime as well as performance is displayed in Figure 5. The number of MCTS iterations M has a strong effect on fidelity and therefore should not be lower than $M = 20$ as chosen by the authors.

6.2 MCTS Exploration vs Exploitation

There is a key difference between the original Monte-Carlo Tree Search algorithm, e.g. as employed by AlphaGo [13], and MCTS employed by the authors. In the original version unexplored nodes in the search tree are treated as leaf nodes until they are explored. Then, from leaf node to terminal node a game would be simulated, for example by playing randomly. In contrast, in SubgraphX only terminal nodes are considered leaf nodes, meaning there is no simulation phase in SubgraphX and there is no benefit in exploiting existing leaf nodes (visiting multiple times). Therefore, the MCTS algorithm should focus on exploration, which the authors enforce by choosing a sufficiently high exploration rate λ.

Looking at a representative MCTS search tree (see Figure 6), we indeed observe this behavior as most subgraphs are visited only once. In our experiment, a broad range of exploration rates resulted in the desired wide exploration with little effect on the fidelity.



(a) SubgraphX for GIN model on MUTAG dataset

(b) Example of MCTS search Tree

Figure 6. Effect of exploration rate λ on fidelity and an example search tree. Blue nodes are visited once, green nodes twice and red nodes more than two times. The red border indicates the best nodes found for different thresholds N_{min}.

7 Improvements beyond original Paper

7.1 Greedy Initialization

During MCTS, the path to the terminal node is taken by choosing the best children according to the upper confidence bound for trees. For node N_i the best pruning action a^* in SubgraphX is chosen as [4]:

$$a^* = arg\,max_{a_j} \frac{W(N_i, a_j)}{C(N_i, a_j)} + \lambda R(N_i, a_j) \frac{\sqrt{\sum_k C(N_i, a_k)}}{1 + C(N_i, a_j)} \tag{5}$$

where $C(N_i, a_j)$ is the visit count, $W(N_i, a_j)$ the total reward of all visits, and $R(N_i, a_j)$ the score function, i.e. the Shapley Value. However, while exploring a new part of the search tree, the visit counts of all nodes are inherently zero. Therefore, the upper confidence bound for all these nodes is not defined due to division by zero. A tie-break has to be used to resolve this conflict. Note that this situation would never occur in the original version of Monte-Carlo Tree Search because it would be in the random simulation phase. However, this situation occurs very often in SubgraphX, as most nodes in the search tree are visited only once (recall Figure 6b). In fact, the tie-break has to be used more often than the upper confidence bound.
In the implementation of the authors, children are chosen by a tie break according to the pruning strategy, which depends only on the node degree. Since the node degree is completely independent of the model to explain, we propose a tie-break based on the Shapley Values of the children. In other words, we greedily choose the child with the highest immediate reward. This strategy is equivalent to initializing the visit counts of all nodes to one. That is, because the only term of equation 5 differing between children in an unexplored subtree would be the immediate reward $R(N_i, a_j)$, i.e. the shapley value. The effects of a greedy strategy are shown in Figure 7.

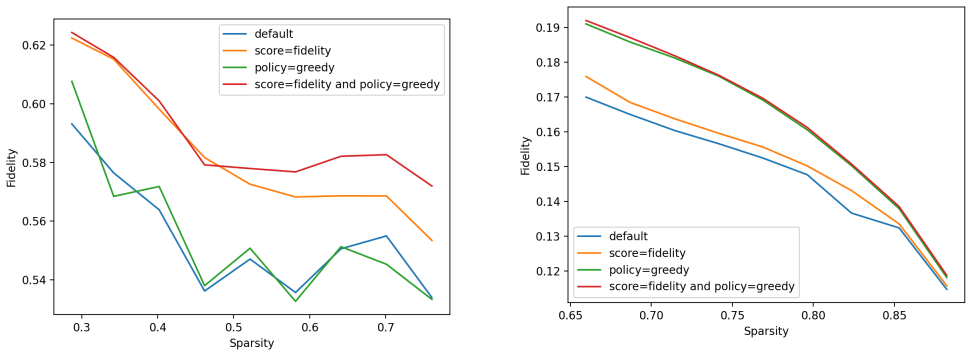
7.2 Fidelity Score Function

Internally, the SubgraphX algorithm uses a metric derived from Shapley Values, denoted as ϕ, to guide the MCTS and select the subgraph for the final explanation. In the end, Yuan et al. use the fidelity of explanations to quantitatively compare their method against others like GNNExplainer [2]. For a single explanation E of predicted class y on graph G, they are computed as [4]:

$$Fidelity = f(G)_y - f(G \setminus E)_y \tag{6}$$

$$\phi(E) = \frac{1}{T}\sum_{t=1}^{T} f(S_t \cup E)_y - f(S_t)_y, \quad S_t \subseteq (G \setminus E) \tag{7}$$

where $f(\cdot)_y$ is the models output for class y, T is the number of Monte-Carlo sampling steps and S_t is sampled randomly. The features of nodes which are not passed to the model are set to a baseline of zero. Comparing these formulas, fidelity is a special case of the Shapley Value (with $S_t = G \setminus E$). Similarly to MCTS_GNN, we can directly optimize for fidelity by using it as the score function. Optimizing for fidelity ($S_t = G \setminus E$) can be seen as the antipodal method to MCTS_GNN ($S_t = \emptyset$).
In Figure 7, a comparison between the scoring functions is shown. In addition to superior performance, using fidelity as a scoring function has the advantage of requiring much less computational effort. That is, because only one step of Monte-Carlo sampling has to be performed. Unfortunately, we did not record the runtime in this experiment. Yet, a comparison can be made because the computation of fidelity as a score function is similar to MCTS_GNN. Thus, it is reasonable to assume that the runtime difference is similar to the values reported in Table 1.



(a) Improved SubgraphX for GIN on MUTAG dataset **(b)** Improved SubgraphX for GCN on Karate Club Network

Figure 7. Effect of greedy initialization and fidelity as a score function on the fidelity of SubgraphX.

7.3 Edge Case Behavior

During our experiments, we observed that SubgraphX explored the same leaf node multiple times for some graph instances. In the usual setting of MCTS, for example in the game of Go [13], this is desirable as it updates the estimated Q-value of all nodes on the path in the search tree. However, in this case we are only interested in the best leaf node. Exploring a leaf node multiple times only repeats the same computation. Therefore, reaching the same leaf node multiple times is wasted computational effort. This problem becomes especially apparent when using a small exploration rate λ. As a solution, we propose to mark nodes in the search tree whose subgraphs are already

completely explored. These nodes should be ignored during search. Future work could include empirical investigation of this effect.

8 Discussion

We were able to reproduce the main claims made by Yuan et al. [4] regarding SubgraphX using our own implementation. SubgraphX has a higher fidelity score than GNNExplainer, while being slower by a factor of seven. This is, according to our own definition, a "reasonable runtime". Additionally, we were able to transfer the results to the Karate Club dataset, where SubgraphX outperforms GNNExplainer as well. During our hyperparameter study, we discovered that SubgraphX is sensitive to number of MCTS iterations M and Monte-Carlo sampling steps T. However, the exploration rate λ has little effect on the performance.
Lastly, we propose to improve SubgraphX by using a greedy initialization and optimizing directly for fidelity. Using these improvements, we were able to achieve higher fidelity at lower runtime.

8.1 What was easy

SubgraphX is explained very concisely in the original paper. It was quite easy to understand and implement the method by ourselves. It was also helpful to check the original implementation in the DIG repository to compare against our own. Furthermore, the datasets were readily available and easy to download and use. The appendix of the paper contained most of the information necessary for the reproduction process.

8.2 What was difficult

Even though the original code of the authors is available as part of the DIG-library, we had to spent considerable time to resolve version issues in order to use the datasets of the original experiments. That is, because at the start of this project the DIG-library did not support PyTorch versions above 1.6.
Another challenge was the computational effort required to perform the experiments. The authors performed extensive experiments, which we could not repeat with our limited resources. Therefore, we had to run the experiments on fewer instances and with fewer iterations than in the original experiments.

8.3 Communication with original authors

We contacted the original authors after finishing the first draft of this report. They briefly reviewed our work and agreed that our experiments prove a successful reproduction of their paper. Additionally, they gave helpful feedback regarding the presentation of our results.

References

1. C. Molnar. **Interpretable Machine Learning. A Guide for Making Black Box Models Explainable**. Bookdown, 2019.
2. Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec. "GNNExplainer: Generating Explanations for Graph Neural Networks." In: **Advances in Neural Information Processing Systems**. Vol. 32. Curran Associates, Inc., 2019.
3. H. Yuan, H. Yu, S. Gui, and S. Ji. **Explainability in Graph Neural Networks: A Taxonomic Survey**. 2021. arXiv:2012.15445 [cs.LG].

4. H. Yuan, H. Yu, J. Wang, K. Li, and S. Ji. "On Explainability of Graph Neural Networks via Subgraph Explorations." In: **Proceedings of the 38th International Conference on Machine Learning**. Vol. 139. Proceedings of Machine Learning Research. PMLR, July 2021, pp. 12241–12252. URL: https://proceedings.mlr.press/v139/yuan21c.html.
5. R. Coulom. "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search." In: **Computers and Games**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 72–83.
6. L. S. Shapley. **Contributions to the Theory of Games**. Vol. 2. Princeton University Press, 1953, pp. 307–317.
7. M. Fey and J. E. Lenssen. **Fast Graph Representation Learning with PyTorch Geometric**. 2019. DOI: 10.48550/ARXIV.1903.02428. URL: https://arxiv.org/abs/1903.02428.
8. D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang. "Parameterized Explainer for Graph Neural Network." In: **Advances in Neural Information Processing Systems**. Vol. 33. Curran Associates, Inc., 2020, pp. 19620–19631. URL: https://proceedings.neurips.cc/paper/2020/file/e37b08dd3015330dcbb5d6663667b8b8-Paper.pdf.
9. A. Debnath, R. Lopez de Compadre, G. Debnath, A. Shusterman, and C. Hansch. "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity." In: **Journal of medicinal chemistry** 34.2 (Feb. 1991), pp. 786–797. DOI: 10.1021/jm00106a046. URL: https://doi.org/10.1021/jm00106a046.
10. W. W. Zachary. "An Information Flow Model for Conflict and Fission in Small Groups." In: **Journal of Anthropological Research** 33.4 (Dec. 1977), pp. 452–473. DOI: 10.1086/jar.33.4.3629752. URL: http://dx.doi.org/10.1086/jar.33.4.3629752.
11. A. Grover and J. Leskovec. **Node2Vec: Scalable Feature Learning for Networks**. 2016. eprint: 1607.00653.
12. T. N. Kipf and M. Welling. **Semi-Supervised Classification with Graph Convolutional Networks**. 2017. eprint: 1609.02907.
13. D. Silver et al. "Mastering the game of Go with deep neural networks and tree search." In: **Nature** 529.7587 (2016), pp. 484–489. DOI: 10.1038/nature16961.