Improving the quality of rule-based GNN explanations

Ataollah Kamal[1], Elouan Vincent[2], Marc Plantevit[2], and Céline Robardet[1]

[1]Univ Lyon, INSA Lyon, CNRS, UCBL, LIRIS, UMR5205, FR-69621 Villeurbanne
[2]EPITA LRE, FR-94276 Le Kremlin-Bicêtre

Abstract. Recent works have proposed to explain GNNs using activation rules. Activation rules allow to capture specific configurations in the embedding space of a given layer that is discriminant for the GNN decision. These rules also catch hidden features of input graphs. This requires to associate these rules to representative graphs. In this paper, we propose on the one hand an analysis of heuristic-based algorithms to extract the activation rules, and on the other hand the use of transport-based optimal graph distances to associate each rule with the most specific graph that triggers them.

1 Introduction

One of the purposes of artificial intelligence is to help human beings to perform cognitive tasks, especially categorization which is among the most important ones. Supporting human beings in this process can be considered in two ways: either by carrying out the process for them or by just helping them so that they keep control of the ongoing process. In this paper, we adopt the second point of view and consider the use of machine learning tools to automatically associate objects with classes in a very efficient way (generally using numerical models with many learned parameters) to then seek to interpret the classification mechanisms to understand how the classification has been made. By making the models explicit, we hope to increase their scope of application in areas with high societal challenges (medicine, justice) but also for the discovery of knowledge (scientific impact). The effectiveness of many recent learning algorithms is at the price of their interpretability, as they rely on the learning of latent variables. This is particularly the case for Graph Neural Networks (GNNs) [22] that classify graphs by learning embedding vectors to represent each of the graph nodes in a metric space so that the classification task based on these vectors is optimized. These vectors encode a lot of information that is unreadable to humans and need to be "interpreted".

Interpretation is an ill-defined concept that has been specified in [5] as covering three distinct aspects: the *comprehensibility*, i.e. the ability for the user to understand the model well enough to be able to apply it manually to new data, the *justifiability*, which specifies whether the model is in line with existing knowledge, and the *plausibility*, i.e. the pragmatic value of the model for the user. In this article, we mainly address the first two aspects by identifying the main activation rules as well as the subgraph that they characterize. The first step relies on pattern mining techniques that have been shown to be valuable for interpreting machine learning black box models [19], especially by providing comprehensible interpretations of a latent space. The second step leverages techniques of Optimal Transport on graphs [17] to transform comprehensible interpretations into justifiable models that makes it possible to evaluate whether the model is in line with existing knowledge expressed in a graph language.

2 Related work

GNNs are generating considerable interest thanks to their performance in several tasks such as node classification [14], link prediction [27] and graph classification [22, 23]. Many cutting-edge techniques improve the performance of models. However, there are few studies that address the explainability of GNNs in comparison to the areas of image and text where an abundance of methods have been proposed [2, 11]. As established by [26], the existing methods for the explanation of convolutional neural networks for the classification of images cannot be directly used on data which is not grid-like such as graphs. For example, the methods that computes an abstract images via back-propagation [16] provide non-exploitable results when they are applied to discrete adjacency matrices. Those that learn soft masks to find important regions of images [13] do not apply to discrete data as well. Though, some methods have been proposed to explain GNNs over the past four years. One can identify three types of explanation methods: (i) instance-level and (ii) model-level explanation methods, that both explain the output of the model, and (iii) rule-based approaches that in addition consider the latent space built by the GNN.

2.1 Instance-level methods

Given an input graph, *instance-level* methods aim to provide input-dependent explanations by identifying important input characteristics on which the model builds its prediction. The gradient/feature-based methods [1] use the gradients or hidden feature map values to compute the importance of the input features. Perturbation-based methods [9, 24] learn a graph mask by studying the prediction changes when perturbing the input graphs. GNNExplainer [24] learns a soft mask by maximizing the mutual information between the original prediction and the predictions of the perturbed graphs. PGExplainer [9] uses a generative probabilistic model to learn succinct underlying structures from the input graph data as explanations. Surrogate methods [6, 20] explain an input graph by sampling its neighborhood and learning an interpretable model. GrapheLime [6] uses a Hilbert-Schmidt Independence Criterion Lasso as a surrogate model. PGM-Explainer [20] builds a probabilistic graphical model for explaining node or graph classification models. These surrogate models can be misleading because the user tends to generalize beyond its neighborhood an explanation related to a local model. GraphSVX [4] falls into these 4 categories by learning a surrogate explanation model on a perturbed dataset, the explained prediction is decomposed among input nodes and features based on their respective contribution.

2.2 Model-level methods

The only existing model-level method is XGNN [25]. It consists in training a graph generator to maximize the predicted probability for a certain class and uses such graph patterns to explain this class. However, it is based on the strong assumption that each class can be explained by a single graph, which is unrealistic when considering complex phenomena.

2.3 Rule-based methods

INSIDE-GNN [18] does not only consider the output of the model when building its explanations: it also considers the intern weight matrix and derive rules that associate a set of activated components to a class. This work is rooted in the FORSIED framework [3] which allows to address the problem of pattern flooding by identifying a set of non redundant and informative patterns. As our work heavily relies on it, we detail below its main characteristics.

Activation matrix. Considering a set of graphs $\mathcal{G}$ where each graph $G = (V, E, L)$ has labels L on vertices. A Graph Neural Network classifies each graph of $\mathcal{G}$ into two categories $\{0, 1\}$: $\text{GNN} : \mathcal{G} \rightarrow \{0, 1\}$. We use a Graph Convolutional Networks (GCN) [7] that computes vectors $\mathbf{h}_v^\ell$ associated to the ego-graph centered in vertex v with radius ℓ, recursively. Such an ego-graph is the sub-graph of G induced by v and all its neighbors at distance ℓ. Each vector is of size K and ℓ varies from 0 up to L (the maximum number of layers in the GNN), two hyperparameters of the GNN. The vectors $\mathbf{h}_v^\ell$ capture the key characteristics of the graphs for the classification task, especially vector components of high value. We therefore consider the activation matrix that has to be interpreted:

$$\widehat{H^\ell}[v, k] = \begin{cases} 1 \text{ if } (\mathbf{h}_v^\ell)_k > 0, \text{ with } k = 1 \dots K, \text{ the dimension of the embeddings} \\ 0 \text{ otherwise} \end{cases}$$

Activation rules. Activation rules group vector components that are mostly activated together in graphs having the same GNN decision. $\mathbf{A}^\ell \rightarrow c$ is composed of a binary vector $\mathbf{A}^\ell$ of size K and $c \in \{0, 1\}$ a decision class of the GNN. A graph $g_i = (V_i, E_i, L_i) \in \mathcal{G}$ activates the rule if there is a node v in V_i such that $\widehat{H^\ell}[v, k] = (\mathbf{A}^\ell)_k$, $\forall k = 1 \cdots K$. The activated graphs with GNN decision c form the support of the rule. Activated rules are more interesting if their supports are largely homogeneous in term of GNN decisions, i.e. the graphs of the support are mainly classified either in class 0 or in class 1.

Measuring the interest of an activation rule. As theorized in the FORSIED framework [3], the knowledge extracted from the activation matrix is modeled by a background model that is used to evaluate the interest of a rule. Considering the discrete random variable $H^\ell[v,k]$ associated to the activation matrix $\widehat{H^\ell}[v,k]$[1], the background knowledge is defined by the probabilities $P(H^\ell[v,k]=1)$. Considering the assumption that all $H^\ell[v,k]$ are independent of each other, the interest of a rule is evaluated by the negative log-probability of the product of $P(H^\ell[v,k]=1)$, for v activated by the rule and k such that $(A^\ell)_k = 1$:

$$IC(R,\mathcal{G}) = \sum_{g_i \in \mathbf{Supp}(R,\mathcal{G})} \min_{\substack{v \in V_i,\\ \mathbf{Act}(R,v)}} \sum_{(A^\ell)_k=1} \log(P(H^\ell[v,k]=1))$$

with $R = A^\ell \to c$, **Supp** the supporting graphs and **Act** the nodes that activate the rule. A pattern with a large IC is more informative but is more difficult to assimilate. Thus, IC value is contrasted by its description length which measures the complexity of communicating the pattern to the user:

$$SI(A^\ell \to c,\mathcal{G}) = \frac{IC(A^\ell \to c,\mathcal{G})}{\alpha.|A^\ell| + \nu}$$

with α the cost for the user to assimilate each component and ν a fixed cost for the pattern[2]. However, in order to identify rules specific to a GNN decision, we consider the difference of subjective interestingness of the measure evaluated on the two groups of graphs. We denote by $\mathcal{G}^0$ (resp. $\mathcal{G}^1$) the graphs $g_i \in \mathcal{G}$ such that $\text{GNN}(g_i)=0$ (resp. $\text{GNN}(g_i)=1$). The subjective interest of the rule $A^\ell \to c$ with respect to the classes is evaluated by

$$SI\_SG(A^\ell \to c) = \omega_c\, SI(A^\ell \to c,\mathcal{G}^c) - \omega_{1-c}\, SI(A^\ell \to c,\mathcal{G}^{1-c}).$$

The weights ω_0 and ω_1 are used to counterbalance the measure in unbalanced decision problems. The rational is to reduce the SI values of the majority class. We set $\omega_0 = \max(1, \frac{|\mathcal{G}^1|}{|\mathcal{G}^0|})$ and $\omega_1 = \max(1, \frac{|\mathcal{G}^0|}{|\mathcal{G}^1|})$.

Computing the background model. The background model is initialized with basic assumptions about the activation matrix:

$$\sum_v P(H^\ell[v,k]=1) = \sum_v P(\widehat{H^\ell}[v,k]=1),\ \sum_k P(H^\ell[v,k]=1) = \sum_k P(\widehat{H^\ell}[v,k]=1).$$

However, these constraints do not completely specify the probability matrix. and we choose the probability distribution with the maximum entropy.

Once a rule $A^\ell \to c$ has been extracted, it brings some information about the activation matrix that can be integrated into P: $P(H^\ell[v,k]=1)$ is set to 1, $\forall k$ such that $(A^\ell)_k = 1$ and v such that $\widehat{H^\ell}[v,k] = (\mathbf{A}^\ell)_k$, $\forall k = 1 \cdots K$.

[1] We use hats to signify the empirical values.

[2] We set $\nu = 1$ and $\alpha = 0.6$, as the constant parameter ν does not influence the relative ranking of the patterns, and with a value of 1, it ensures that the DL value is greater than 1. With $\alpha = 0.6$, we express a slight preference toward shorter patterns.

2.4 Limitations and desiderata

Most of the introduced methods attempt to explain a GNN model from its final decision. INSIDE_GNN [18], is the only one to analyze the internal structure of the network and to build an explication on the different layers of GNN. However, due to the exhaustive search employed to construct the activation rules, this method is time-consuming, which makes it difficult to use for large sets of graphs. Moreover, these rules are not intelligible in themselves and it is important to know which parts of the graphs they capture. These are the two limits that we address in the following.

3 Computing activation rules

We propose and study three approaches to compute iteratively the activation rule $R = A^{\ell} \rightarrow c$ with the largest SI_SG value and to integrate it in the background distribution P to take into account the knowledge provided by the rule. The first method is an exact algorithm, the two others are approximation methods. In the two last approaches, we are able to consider activated components (indices k such that $(\mathbf{h}_v^{\ell})_k > 0$) and non-activated components (when $(\mathbf{h}_v^{\ell})_k \leq 0$).

3.1 Using an exhaustive search

This enumerate-and-rank approach starts with the empty rule $\emptyset \rightarrow c$ and recursively add components to A. We use a branch and bound approach, updating the current best SI_SG value found so far, using the following upper bound:

$$UB\_SI(R) = \frac{w_c}{\alpha(|A|) + \nu} \times \sum_{g_i \in \mathbf{Supp}(R,\mathcal{G}^c)} \min_{\substack{v \in V_i,\\ \mathbf{Act}(R,v)}} \sum_{(A\&D)_k=1} \log(P(H^{\ell}[v,k]=1))$$
$$-\frac{w_{1-c}}{\alpha(|A\&D|) + \nu} \times \sum_{g \in \mathbf{Supp}(A\&D,\mathcal{G}^{1-c})} \min_{\substack{v \in V_i,\\ \mathbf{Act}(R,v)}} \sum_{(A)_k=1} \log(P(H^{\ell}[v,k]=1))$$

with D a vector whose one's values represent the activated components that can be further added to A during the enumeration process, and $A\&D$ the bitwise **and** operation between vectors A and D. $|A|$ is the L1 norm of A. UB_SI makes the recursion stop if its value is less that the one of the current best rule found.

3.2 Using Beam Search

This algorithm is a tree search algorithm pretty similar to the breadth-first search with the difference that in each stage it only keeps a fixed number of descendants. A selector, an atomic proposition of the form $X == Y$, where X is a component and $Y \in \{True, False\}$, describes the status of a component. A conjunction D of selectors forms a description. For a description D, the length of the description is the number of its selectors. A graph g is in the support of D, if D is true for at least one node of g from logical point of view. Therefore, we can have a mapping between a rule and a description and thanks to this mapping we can define subjective interestingness (SI_SG) for a description. To this ends, the SI_SG of a description is the subjective interestingness SI_SG of its mapped rule. Each node of the beam search tree corresponds to a description and its children are those descriptions by adding one new selector to the corresponding description. The root of the tree is the description with length 0. Thus, nodes in the $depth = 1$ are selectors. At each stage of the algorithm we use a beam-width (bw) parameter that indicates the number of nodes that at the end of the stage would be kept. Those with the highest SI_SG values are kept. Besides discovering the new nodes, we save the node with the best SI_SG that so far we have found. As the depth of the tree can be too high and regarding the fact that we are interested in simple rules, we limit the exploration up to a certain depth. In our task, $bw = 20$ and the maximum depth is 9. After each run, we get one rule in return. Then we update the model with respect to the rule. We use PySubgroup framework [8] for this task.

3.3 Using Monte Carlo Tree Search

MCTS partially explores the tree of possible rules where each node v represents a partial rule as a tuple $(free, fixed)$: the components of the embedding vector are either in the $free$ or the $fixed$ set of the tuple. The $free$ set contains the components that have not been treated yet, and $fixed$ is a set of couples (x, y) that indicates that component x has the state y, y being either $activated$, $non - activated$ or $loose$ meaning that x is activated, non-activated or there is no constraint on it. A partial rule with $free = \emptyset$ is called a rule.

MCTS focuses on analyzing the most promising partial rules, expanding the search tree based on random sampling of the search space. Monte Carlo tree search is based on many roll-outs. In each playout, a rule is constructed by selecting component values at random until $free = \emptyset$. The value of SI_SG from the obtained rule is then used to weight the nodes in the tree so that the best nodes are more likely to be chosen in future roll-outs. To that end, v_1 and v_2 are two numerical values also associated to each node v, with v_1 is the subjective interestingness value of the rule $fixed \cup \{(x, loose) : x \in free\}$, and v_2 value is defined in the roll-out and propagation step of the algorithm. Each round of Monte Carlo tree search consists of four steps:

- **Selection**: Starting from the root node, it selects successive child nodes until a leaf node is reached. A leaf is any node that has a potential child and from which no simulation (roll-out) has yet been done. The section of child nodes is biased so that the tree expand towards the most promising rules, which is the essence of Monte Carlo tree search. A child v is selected if it satisfies $SI\_UB(v) \geq SI\_SG(best\_rule)$ and maximizes the value: $v_1 + \frac{v_2}{n_v} + \kappa\sqrt{\frac{n * \log(N_v)}{n_v}}$. N_v is the number of times the parent of v has been visited,

n_v is the number of times v has been visited, and $n = |fixed|$. Note that in case $n_v = 0$, this function equals to ∞. κ is set to 100.

- **Roll-out and Propagation**: From a leaf-node, if this node is not terminal (i.e. $free \neq \emptyset$), we randomly assign values to the components in $free$ to reach a rule. Then the subjective interestingness of this rule is computed and added in v_2 variables of all the nodes in the path from this node to the root (propagation). In case that SI_SG of this node is the best value found so far, we store it as the *best_rule*. To avoid visiting already visited terminal nodes, we add to each i^{th} node of the path the value $(-1)^i \frac{x}{2^i}$ to v_2, where $x = SI\_SG$.
- **Expand Children**: Once a node has been visited, we expand all its children and we pursue with the first child u such that $SI\_UB(u) \geq SI\_SG(best\_rule)$. The expand consists in building $2 \times |free|$ children by taking a component in $free$ and assigning values *activated*, *desactivated*, or *loose*.

Each run of the algorithm finds one rule and consists of 100,000 iterations of the above steps. There is another termination condition for a run: if there is a node v, with $n_v > 500$, the run terminates. After finding a rule, we update the model the same as the exact method. We run the MCTS until either we reach 10 rules, or reach a rule with $SI\_SG < 10$ or there would be at least one rule r for each component c in which c has a non-free state.

4 Transforming rules into subgraphs

The activation rules make it possible to isolate the characteristics of the graphs useful to the task of classification. However, although we know that the graphs supporting the rule have common characteristics, we do not know which ones it is. We then propose to search for these properties that the graphs supporting the rule have in common by searching for the median graph of this set. This approach makes it possible to summarize the whole set of supporting graphs by a single realistic graph. As we would like to calculate a median for a set of graphs, we need to define a distance between two graphs. Being able to leverage both features and structural information from graphs to calculate their distance can be time consuming, requiring the combination of these two pieces of information in a way that makes it possible to capture the similarity between graphs. We opt for the use of a distance based on Optimal Transport known to unveil the geometric nature of attributed graphs. Unlike Wasserstein or Gromov-Wasserstein metrics, that focus solely and respectively on features or structure, the distance Fused Gromov-Wasserstein (FGW) [17] exploits jointly both information.

4.1 Optimal Transport

Optimal Transport (OT) defines a distance between two probability distributions. It already prove its utility in a lot of fields, this is not yet very developed for graphs. While features can be compared using a standard metric, such as l_2, comparing structures requires a notion of similarity which can be found via the concept of isometry, since the nodes of the graph are not ordered. But the natural formulation of OT cannot exploit the structural information of objects since it only relies on a cost function that compares their feature representations. A way to compare two distance matrices, seen as representations of object structures, is introduced in [10] where an OT metric, called Gromov-Wasserstein distance, is able to compare two distributions even if they are not in the same ground space. In [17], authors introduce the Fused Gromov-Wasserstein (FGW) using OT. It uses both Gromov-Wasserstein distance on structure and Wasserstein distance on features. Graphs are transformed into probability measures via histograms which are used to identify the relative importance of vertices in the graph. On a graph of n vertices, the vertices are associated with weights $(h_i)_i \in \sum_n$. When all the weights are equal, it means that all the vertices have the same relative importance, the structured data contains exactly the same information as their graph. FGW uses a compromise parameter $\alpha \in [0, 1]$. When α tends to zero, the FGW distance retrieves the Wasserstein distance between features, whereas when α tends to one, it retrieves the Gromov-Wasserstein distance between structure.

The FGW distance looks for the coupling π between the vertices of the graph that minimizes the cost function which is a linear combination of a cost $d(a_i, b_j)$ of transporting one feature a_i to a feature b_j and a cost $|C_1(i, k) - C_2(j, l)|$ of transporting pairs of nodes in each structure, where C_1 and C_2 are the structure matrix of the two graphs which are compared. FGW is null iff graphs have the same number of vertices and if there exists a one to one mapping between the vertices of the graphs which respect both shortest-paths and the features. The complexity is in $O(n^2m + nm^2)$ and FGW defines a semi-metric.

4.2 Barycenter

A notion of barycenter is also introduced in [17] based on FGW distance. It looks for the graph that minimizes the sum of (weighted) FGW distances within a given set of structured data associated with structure matrices, features and base histograms. This is the first formulation of barycenter of a set of graphs that can leverage both structural and feature information. We cannot use directly the barycenter in our method for the following reasons: (1) To compute the barycenter of a set of graphs, we need to specify the parameter n that defines the number of vertices in the generated graph; (2) Graphs that are generated are not guaranteed to be realistic; (3) It cannot work on graphs labeled with discrete values. This justifies our following proposal for computing median graphs.

4.3 Associating a graph to a rule

To generate completely realistic graphs with an embedding close to an activation rule, we propose to calculate the median graph of all the graphs of the support (those that activate the rule), and then to perform a best first enumeration to find the subgraph with the highest score on the activation rule.

Median graph. Computing a median graph guaranties that the graph is realist as it is an element of the set of graphs. Also, we are sure that this graph activates the rule. The median graph of a set G of graphs is the graph of G whose average FGW distance to other graphs of G is minimal. It requires to compute all distances between every pair of graphs which can be time expansive. Therefore, we propose to compute an approximation of the median. It makes it possible to avoid considering graphs that are close to other ones.

In Algorithm 1, the approximate median of a set of graphs G starts with and empty set of selected graphs S. It first draws a graph g uniformly at random in G and adds it to S. Then, all the distances between g and the graphs of G are computed. A loop starts that consists in drawing at random a new graph g from $G \setminus S$, but this time according to the distances $dist(g, S)$. This graph is added to S. The loop stops when $dist(g, S)$ is small enough.

The further the graph is from the set, the higher its probability of being drawn and added to the set S. When the loop stops, the median graph on the set S is computed and returned.

Algorithm 1 Approximate median graph

```
Require: $G$ a set of graphs, $t$ a threshold
Ensure: Median graph
S ← {}
g ← drawn from $G$
S ← $S \cup \{g\}$
repeat
    draw $g \sim dist(g, S)$
    S ← S ∪ $\{g\}$;
until ($dist(g, S) < t$)
return median(S)
```

The approximate median graph procedure only computes $\sum_{i=i}^{q}(n-i)i = q(q+1)\left(\frac{n}{2} - \frac{2q+1}{6}\right)$ FGW distances, instead of n^2, with n the size of G and q the number of iterations[3]. In practice q is between 10% to 40% of n.

Improving the median graph to better describe the rule. The median graph supports the rule but it is potentially not specific to it and may contain additional information not related to the rule. Starting from the median graph, or its approximation, we search for a subgraph whose embedding vector is the closest to the activation rule. This proximity between the embedding vector and the rule is evaluated by the Cosine metric between the vectors as in [19]. To maximize the Cosine value from a graph g, we first compute the Cosine value for g. Then, we enumerate all the subgraphs of g that are obtained by removing a single vertex. The subgraph with the largest Cosine value is taken, and the process iterates until no better graph is found.

[3] $\sum_{i=1}^{q}(n-i)\,i = n\sum_{i=1}^{q} i - \sum_{i=1}^{q} i^2 = n\frac{q}{2}(1+q) - \frac{q(q+1)(2q+1)}{6} = q(q+1)\left(\frac{n}{2} - \frac{2q+1}{6}\right)$.

5 Experiments

The purpose of the experiments is twofold: the comparative study of the algorithms to extract activation rules in terms of computation time and rule quality, and the study of distances based on the optimal transport to associate the most specific graph to each of the rules. For these experiments, we trained a GNN with 3 layers of dimension $K = 20$ for each dataset. We mined at most 10 rules for each layer and class with $SI\_SG > 10$. All the experiments have been written in python and done on a machine with 8 Intel(R) Xeon(R) W-2125 CPU 4.00GHz cores 128GB RAM, and Debian GNU/Linux operating system.

5.1 Datasets

We have used four datasets Aids [12], BBBP [21], Mutagen [12] and BA2 [24]. BA2 is a synthetic dataset in which graphs with the label 0 have a cycle of length 5 and the graphs of opposite class, have "house" motifs. Graphs in the rest of the datasets represent real molecules. Main characteristics of these datasets are given in Table 1 (left).

Dataset Name	#Graphs	(#neg, #pos)	Avg. Nodes	Avg. Edges
Aids	2000	(400,1600)	15.69	322
BBBP	1640	(389,1251)	24.08	51.96
Mutagen	4337	(2401, 1936)	30.32	61.54
BA2	1000	(500, 500)	25	50.92

Dataset	MCTS	Exhaustive search
Aids	14:24	11:14
BBBP	05:15	13:29
Mutagen	35:10	69:16
BA2	00:37	02:22

Table 1. Dataset description: number of graphs, number of graphs with positive and negative labels, and average number of nodes and edges in each dataset (left). Time comparison for MCTS and exhaustive search. Times are in the format of hh:mm (right).

5.2 Computing rules

We evaluate two approximation methods, beam search and MCTS, in comparison with the exhaustive search method. The main goal of our work is to reduce the running time. However, we should be careful what we lose in price of the time. Therefore, we measure the total interestingness of patterns obtained by each method in comparison to the exhaustive search. To assess how explainable our patterns are, we use fidelity, infidelity, and sparsity measures.

Time Comparison. Among all the methods, beam search has the best time. All the experiments have been done under twenty minutes. In the second place, MCTS has a better time in three datasets (BA2, BBBP and Mutagen) than exhaustive search. However, in the Aids dataset, the process did not complete in less time than the exhaustive search. This problem is due to the computation of rules for the last layer of the GNN. In the Table 1 (right), the time needed by MCTS and exhaustive search methods are compared.

Cumulative Subjective Interestingness (CSI). To evaluate how good is the quality of the rules mined by beam search and MCTS, one factor is cumulative subjective interestingness of them. Figure 1 shows that exhaustive search has the best CSI in non-synthetic datasets, MCTS works better than beam search despite its early termination in the Aids due to the runtime exceeding. Another interesting point is in the BA2 dataset with MCTS. For the last layer and class 0, the first and second rules discovered by exhaustive search have SI_SG of 700 and 261 respectively and for the same class and layer, by MCTS, the first two rules have SI_SG of 674 and 433 respectively which resulted to have a better CSI in MCTS, which is an approximate method than exhaustive search as an exact algorithm. Therefore, although that MCTS in some places can be time-consuming, it can have interesting features to study.

When we consider non-activated components, although we have more general space, we cannot get better results than when we have only activated components, except for BA2 and Aids. In BA2 we obtain results even better than the exhaustive search and in Aids they are better than the approximation methods but not better than the exact one. The main drawback of the MCTS with the mode consisting activated and non-activated components, is the running time. We could not obtain results for the Mutagen dataset due to this problem.

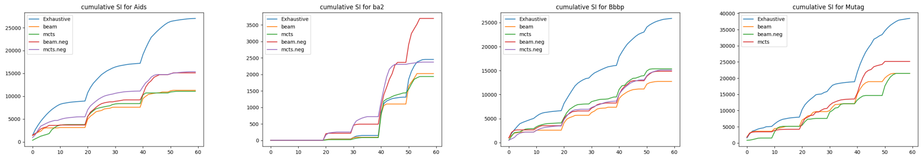

Fig. 1. Cumulative subjective interestingness comparison between the three methods (exhaustive search, MCTS and beam search). In each chart, the horizontal axis is the number of the rules and the vertical axis is SI_SG. The suffix ".neg", represent methods while considering non-activated patterns.

Fidelity, Infidelity, and Sparsity. So far we have rules that are not still human interpretable. To have some human-friendly explanations, in each graph that activates a pattern we build a mask for that pattern. Considering a graph g in the support of the considered rule and $s \subseteq V_g$ is the set of its vertices that activate all the components of the rule. Then the mask for graph g is the induced subgraph by $s \cup N(s)$ where $N(s)$ is the neighbors of s. We expect the mask to be the reason for the decision of the GNN for graph G. To measure how well these masks capture the decision of the GNN, we use three measures fidelity, infidelity, and sparsity [15]. Fidelity measures how the GNN decision changes when removing the mask from the graph. It should be maximized. The infidelity measures the difference in the GNN decision when considering the whole graph and only the mask. It should be minimized. These metrics are not enough to

	(a) Fidelity				(b) Infidelity				(c) Sparsity			
Model	Aids	BBBP	Mutagen	BA2	Aids	BBBP	Mutagen	BA2	Aids	BBBP	Mutagen	BA2
Exhaustive	0.179	0.312	0.499	0.343	0.767	0.420	0.305	0.003	0.884	0.916	0.962	0.032
MCTS	0.178	0.624	0.526	0.343	0.767	0.131	0.344	0.004	0.877	0.265	0.939	0.041
BS	0.792	0.522	0.514	0.343	0.074	0.170	0.309	0.002	0.270	0.452	0.938	0.028
MCTS neg.	0.172	0.322	N/A	0.341	0.767	0.385	N/A	0.029	0.901	0.899	N/A	0.105
BS neg.	0.808	0.304	0.417	0.343	0.036	0.352	0.341	0.006	0.132	0.804	0.989	0.058
GnnEx	0.036	0.100	0.177	0.093	0.036	0.099	0.140	0.223	0.501	0.501	0.505	0.804
PGEx	0.032	0.098	0.157	0.004	0.038	0.098	0.157	0.353	0.547	0.534	0.515	0.955
PGM-Ex	0.080	0.212	0.123	0.222	0.766	0.482	0.347	0.296	0.862	0.884	0.900	0.746
SVXEx	0.003	0.008	0.039	0.004	0.771	0.489	0.356	0.341	0.988	0.940	0.931	0.943

Table 2. Assessing the explanations with several metrics. A better explainer achieves higher fidelity, lower infidelity while keeping a sparsity close to 1. The suffix (neg) represent methods while considering activated and non-activated components.

assess a set of masks. As an illustration, assume that $m_i = g_i$ for $1 \leq i \leq n$. In this case, infidelity can be 0. Therefore, we need another metric that is sensitive to the proportion of a graph used as its mask: $Sparsity(M) = \frac{1}{n} \sum i = 1^n (1 - \frac{|m_i|}{|g_i|})$. So in the case that we have masks identical to their corresponding graphs, which minimizes the fidelity, sparsity will be zero too. Thus, the greater sparsity means the better masks.

Table 2 shows the values of these metrics compared to state of the art methods for explaining GNNs. As it can be seen, on the Aids dataset, MCTS has comparable results in all of the metrics to the exhaustive search. Although beam search has better fidelity and infidelity than MCTS and exhaustive search, it has lower sparsity. It can be interpreted that activation rules obtained by this method cover too many nodes. On the BBBP, both methods in terms of fidelity and infidelity have outperformed the exhaustive search. However, sparsity for both of them is lower than the one of exhaustive search. On the Mutagen and BA2 datasets, metrics are pretty close which means that rules captured by the two approximation methods are as explainable as those captured by the exact method.

These preliminary experiments do not make it possible to conclude on the added-value of the non-activated components. Other rules evaluation measures would be necessary.

5.3 Finding a representative graph for a rule

Our goal is to generate a representative graph for each rule with median approximation and best first enumeration[4]. This experimental study aims to answer the following questions: Is the median approximation good? Is the reduction of the execution time of the approximation significant? How close the median approximation is to the embedding of a targeted rules? How good the best first

[4] All algorithms are implemented in Python, using the FGW code given by the author https://github.com/ElouanV/optimal_transpor_for_gnn

enumeration improves the score? We compare the generated graph to those generated by DISCERN [19]. For FGW, we set $\alpha = 0.9$, giving more importance to the structural information, but in molecule, features and structure information are correlated. We use shortest path as a method for structure matrix of graphs and *sqeuclidean* to compute the cost matrix between the features. In median approximation, we set the threshold $t = 10^{-10}$. In score computation, we use *Cosine* metrics to compute the similarity between an ego-graph and a rule. Most of the experiments are done on two rules of Mutagen datasets (rule 23 and 28) as we know that they are highly correlated to the mutagenicity.

Median approximation quality. To study the median approximation quality, the distance between the median of a set and its approximation, we compute at each iteration the distance between the real median and the median of the set S (see Fig. 2). But, in this set of graphs, there are a lot of graphs that are really close to each others, and even some graphs are identical. For example, the real median graph of this set exists in nine copies. The median approximation function uses a threshold to stop when the distance between the newly selected graph and the set S is too small. This distance is monitored over iterations on the same rule and shown in Fig. 2.

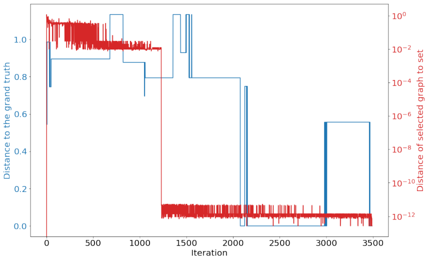

Fig. 2. Distance between the selected graph and the set S at each iteration on a logarithmic scale in red, distance between real median of a set of 3490 graphs supporting rule 23 of Mutagen and the approximation of median over iterations using FGW distance in blue.

Here, the algorithm stops at iteration 1239 over 3490. It means that we only compute the median on 35% of the graphs of the set. The distance between the median of this set and its approximation is 0.8 which seems to far comparing to all distances between graphs of this set, but it shows that there is a lot of duplicated graphs in the set, and the approximation eliminates them and find a median graph that may better represent the set. On other rules, like the rule 28 of Mutagen, the approximation converges quickly to the same graph as the real median. On the set of graph that are not big enough i.e. less than 200 graphs, the approximation methods is useless since.

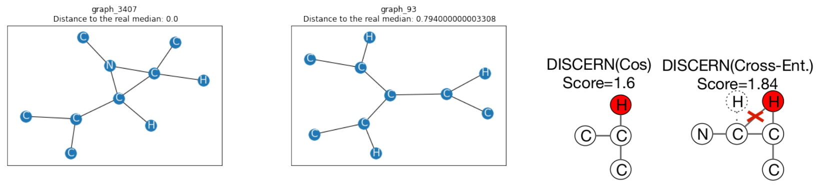
graph_3407
Distance to the real median: 0.0
graph_93
Distance to the real median: 0.794000000003308
DISCERN(Cos)
Score=1.6
DISCERN(Cross-Ent.)
Score=1.84

Fig. 3. The real median of the graphs from rule 23 of Mutagen *(left)* and the approximation with $t = 10^{-10}$ *(middle)*. Graphs generated by DISCERN on rule 23, red cross highlights unrealistic bonds or molecules (right).

On Fig. 3, we compare the median graph of all graphs of the rule 28 of Mutagen and the median approximation of the same set. First of all, they have the same number of vertices. The difference between them comes from the nitrogen atom, which is not present in the median approximation, and the structure also is a bit different, but in both cases, we can identify three part link by an atom of carbon in the middle. Moreover, both graphs have the same number of atoms of carbon. When we compare it to the graph generated by the DISCERN method on the same rule Fig. 3 (right), we find in both of our graphs the three carbon chains, but the nitrogen atom is only present on the real median.

Median approximation execution time. We also want to study the execution time we can win to balance with the loss of accuracy. The execution time reduction depends on each set. On the same rule of Mutagen in Fig. 4 (left), we can observe that the execution time over iterations is almost linear, so by selecting only 35% of the graphs, we highly reduce the execution time by almost 60%. In Fig. 4 (right), we can see the percentage of graphs use to compute the median from a set thanks to the approximation. Among these 60 rules, some of them contain more than 10 000 graphs, which is a lot more than what we have seen in rule 23 of Mutagen. We can see that the percentage of graphs sectioned for the approximation decreases when the number of graphs in the set increases. When there is more than 10 000 graphs, we only select less than 10% of them which allows us to reduce the computation time significantly. When there is less than 1000 graphs, the approximation use almost all the graphs to compute the median, but it is not an issue.

Are the result good? We compute the median approximation on the 60 rules of AIDS dataset, and use the computed median as starting seed for the best first enumeration. In Fig. 5, we focus on the rule 54, and we compare the median approximation to the output of the best enumeration first. The score is increasing from 0.48 to 0.65 thanks to the exploration, and the result is a cycle of 5 atoms.

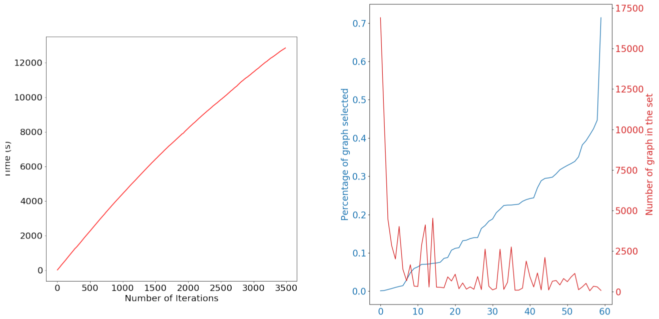

Fig. 4. Computation time over iterations of median approximation (rule 23 of Mutagen) in second (left). Proportion of graphs use from a set in for an approximation for the 60 rules of AIDS dataset in blue (sorted) and number of graphs in each set of graphs in red (right).

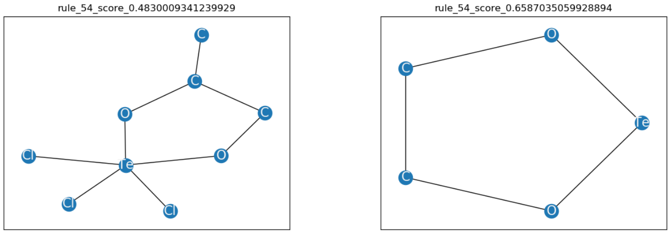
rule_54_score_0.4830009341239929
C
C
O
C
O
Cl
Te
Cl
Cl
rule_54_score_0.6587035059928894
O
C
Te
C
O

Fig. 5. Median approximation of the rule 54 of AIDS dataset (left) and the subgraph generated by the best first enumeration (right).

6 Conclusion

We have proposed two alternative algorithms for computing activation rules. Experiments showed that beam search reduces the computation time significantly and in terms of fidelity and infidelity has acceptable results. We have also introduced a novel method for explaining internal representations of GNNs. With a median graph computation and a better first enumeration, we associate each rule with a realistic graph that fully embeds in the subspace defined by the activation rule. The study shows that the median approximation makes it possible to reduce the computation time without losing the quality of the generated graphs. In future work, we might try adding vertices and edges to the median graphs to see if this can improve the generated graph, giving the median graph as an exploration seed for MCTS as the method DISCERN did.

References

1. Baldassarre, F., Azizpour, H.: Explainability for GCNs. arXiv:1905.13686 (2019)

2. Burkart, N., Huber, M.F.: A survey on the explainability of supervised machine learning. JAIR **70**, 245–317 (2021)
3. De Bie, T.: An information theoretic framework for data mining. In: SIGKDD. pp. 564–572 (2011)
4. Duval, A., Malliaros, F.D.: Graphsvx: Shapley value explanations for graph neural networks. In: ECMLPKDD'21. pp. 302–318 (2021)
5. Fürnkranz, J., Kliegr, T., Paulheim, H.: On cognitive preferences and the plausibility of rule-based models. Machine Learning **109**(4), 853–898 (dec 2019)
6. Huang, Q., Yamada, M., Tian, Y., et al.: GraphLIME: Local Interpretable Model Explanations for Graph Neural Networks. arXiv:2001.06216 (2020)
7. Kipf, T., Welling, M.: Semi-supervised classification with GCN. In: ICLR (2017)
8. Lemmerich, F., Becker, M.: pysubgroup: Easy-to-use subgroup discovery in python. In: ECMLPKDD. pp. 658–662 (2018)
9. Luo, D., Cheng, W., Xu, D., Yu, W., Zong, B., Chen, H., Zhang, X.: Parameterized explainer for GNN. In: NeurIPS 2020 (2020)
10. Mémoli, F.: Gromov-wasserstein distances and the metric approach to object matching. Found. Comput. Math. **11**(4), 417–487 (2011)
11. Molnar, C.: Interpretable machine learning. Lulu. com (2020)
12. Morris, C., Kriege, N.M., Bause, F., Kersting, K., Mutzel, P., Neumann, M.: Tudataset: A collection of benchmark datasets for learning with graphs (2020)
13. Olah, C., Mordvintsev, A., Schubert, L.: Feature visualization. Distill **2**(11) (2017)
14. Park, H., Neville, J.: Exploiting interaction links for node classification with deep graph neural networks. In: IJCAI 2019. pp. 3223–3230 (2019)
15. Pope, P.E., Kolouri, S., Rostami, M., Martin, C.E., Hoffmann, H.: Explainability methods for GCN. In: IEEE CVPR. pp. 10772–10781 (2019)
16. Simonyan, K., Vedaldi, A., Zisserman, A.: Deep inside convolutional networks. In: ICLR 2014 (2014)
17. Vayer, T., Courty, N., Tavenard, R., Chapel, L., Flamary, R.: Optimal transport for structured data with application on graphs. In: ICML. pp. 6275–6284 (2019)
18. Veyrin-Forrer, L., Kamal, A., Duffner, S., Plantevit, M., Robardet, C.: On GNN explanability with activation patterns (Oct 2021), working paper or preprint
19. Veyrin-Forrer, L., Kamal, A., Duffner, S., Plantevit, M., Robardet, C.: What does my GNN really capture? On the exploration of GNN internal representations. In: IJCAI-ECAI (2022)
20. Vu, M.N., Thai, M.T.: PGM-Explainer: Probabilistic Graphical Model Explanations for Graph Neural Networks . In: NeurIPS 2020 (2020)
21. Wu, Z., Ramsundar, B., Feinberg, E., Gomes, J., Geniesse, C., Pappu, A.S., Leswing, K., Pande, V.: Moleculenet: a benchmark for molecular machine learning. Chem. Sci. **9**, 513–530 (2018)
22. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Philip, S.Y.: A comprehensive survey on GNNs. IEEE trans. on NN and learning systems **32**(1), 4–24 (2020)
23. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are GNN? In: ICLR (2019)
24. Ying, Z., Bourgeois, D., You, J., Zitnik, M., Leskovec, J.: GNNExplainer: Generating Explanations for Graph Neural Networks. In: NeurIPS. pp. 9240–9251 (2019)
25. Yuan, H., Tang, J., Hu, X., Ji, S.: XGNN: Towards Model-Level Explanations of Graph Neural Networks . In: KDD'20. pp. 430–438 (2020)
26. Yuan, H., Yu, H., Gui, S., Ji, S.: Explainability in graph neural networks: A taxonomic survey. arXiv:2012.15445 (2020)
27. Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An end-to-end deep learning architecture for graph classification. In: AAAI-2018. pp. 4438–4445 (2018)