Available online at www.sciencedirect.com
ScienceDirect
Data & Knowledge Engineering 00 (2022) 1–23
Data & Knowledge Engineering

In pursuit of the hidden features of GNN's internal representations

Luca Veyrin-Forrer[a], Ataollah Kamal[a], Stefan Duffner[a], Marc Plantevit[b], Céline Robardet[a]

[a] *Univ Lyon, INSA Lyon, CNRS, UCBL, Centrale Lyon, Univ Lyon 2, LIRIS, UMR5205, F-69621 Villeurbanne, France*
[b] *Laboratoire de Recherche de l'EPITA (LRE), Le Kremlin-Bicêtre, 94276, France*

Abstract

We consider the problem of explaining Graph Neural Networks (GNNs). While most attempts aim at explaining the final decision of the model, we focus on the hidden layers to examine what the GNN actually captures and shed light on the hidden features built by the GNN. To that end, we first extract activation rules that identify sets of exceptionally co-activated neurons when classifying graphs in the same category. These rules define internal representations having a strong impact in the classification process. Then – this is the goal of the current paper – we interpret these rules by identifying a graph that is fully embedded in the related subspace identified by the rule. The graph search is based on a Monte Carlo Tree Search directed by a proximity measure between the graph embedding and the internal representation of the rule, as well as a realism factor that constrains the distribution of the labels of the graph to be similar to that observed on the dataset. Experiments including 6 real-world datasets and 3 baselines demonstrate that our method DISCERN generates realistic graphs of high quality which allows providing new insights into the respective GNN models.

Keywords: Graph Neural Networks, Explainable Artificial Intelligence, Monte Carlo Tree Search

1. Introduction

Graphs are a powerful and widespread data structure used to represent relational data. One of their specificity is that their underlying structure is not in Euclidean space and does not have a grid-like structure, characteristics that facilitate the direct use of generic machine learning techniques. Indeed, each node of a graph is characterized by its features, its neighboring nodes and recursively their properties. Such intrinsically discrete information cannot be easily used by standard machine learning methods to either predict a label associated with the graph or a label associated with each node of the graph. To overcome this difficulty, Graph Neural Networks (GNNs) learn embedding vectors $\mathbf{h}_v \in \mathbb{R}^K$ to represent each node v as a continuous vector that eases comparison between similar nodes. GNN methods employ a message propagation strategy that recursively aggregates information from nodes to neighboring nodes. This method produces vector representations of ego-graph centered in each node v (with radii equal to the recursion index) in such a way that the classification task, based on these vectors, is optimized.

Although GNNs have achieved exceptional performance in many tasks, a major drawback is their lack of interpretability. The last five years have witnessed a huge growth in the definition of techniques for explaining deep neural networks [3, 17], especially for image and text data. However, the explainability of GNNs has been much less explored. Two types of approaches have recently been proposed and have gained a certain visibility. On the one hand, instance-level explanation algorithms [1, 8, 16, 21, 24, 38] aim at learning a mask seen as an explanation of the model decision for a graph instance. They obtain the best performances for instance explanation. It appears that such masks

can lead to unreliable explanations, and most importantly, can lead to misleading interpretations for the end-user. One can be tempted to interpret all the nodes or features of the mask as responsible for the prediction leading to wrong assumptions. An example of misleading interpretations is when a node feature is perceived as important for the GNN prediction, whereas there is no difference between its distribution within and outside the mask. On the other hand, the XGNN method [39] aims at providing model-level explanations by generating a graph pattern that maximizes a GNN output label. Yet, this assumes that there is a single pattern for each target which is not the case in practice when dealing with complex phenomena. Most of these methods query the GNN with perturbed input graphs to evaluate their impact on the GNN decision and build their masks from the model output. They do not study the internal mechanisms of the GNNs, especially the different embedding spaces produced by the graph convolutions, although we are convinced that the study of GNN activation vectors may provide new insights on how GNNs perceive the world.

In this paper, we consider GNNs for graph classification. We introduce a new method, called DISCERN (DISClosing the IntERnal workings of gNns with graphs), that aims at characterizing interesting internal representations of the GNN with graphs as illustrated in Fig. 1. In each hidden layer of the GNN, we identify sets of neurons that are differently activated according to the output variable. Such activation rules capture specific configurations in the embedding space of a given layer that is discriminant for the GNN decision. We believe that such activation rules also catch hidden features of input graphs. They can be directly used to support instance-level model explanation. However, these activation rules cannot be easily interpreted by human beings. Our goal is then to explain each activation rule by generating a graph that is fully embedded in the related subspace identified by the rule. To that end, we define a proximity measure to assess how close a graph is to an activation rule and optimize this measure using a Monte Carlo Tree Search (MCTS). Eventually, the generated graphs can then provide insights about the model, especially highlighting what the model actually capture through the GNN. These graphs can also be used to explain single instance decisions of the related model.

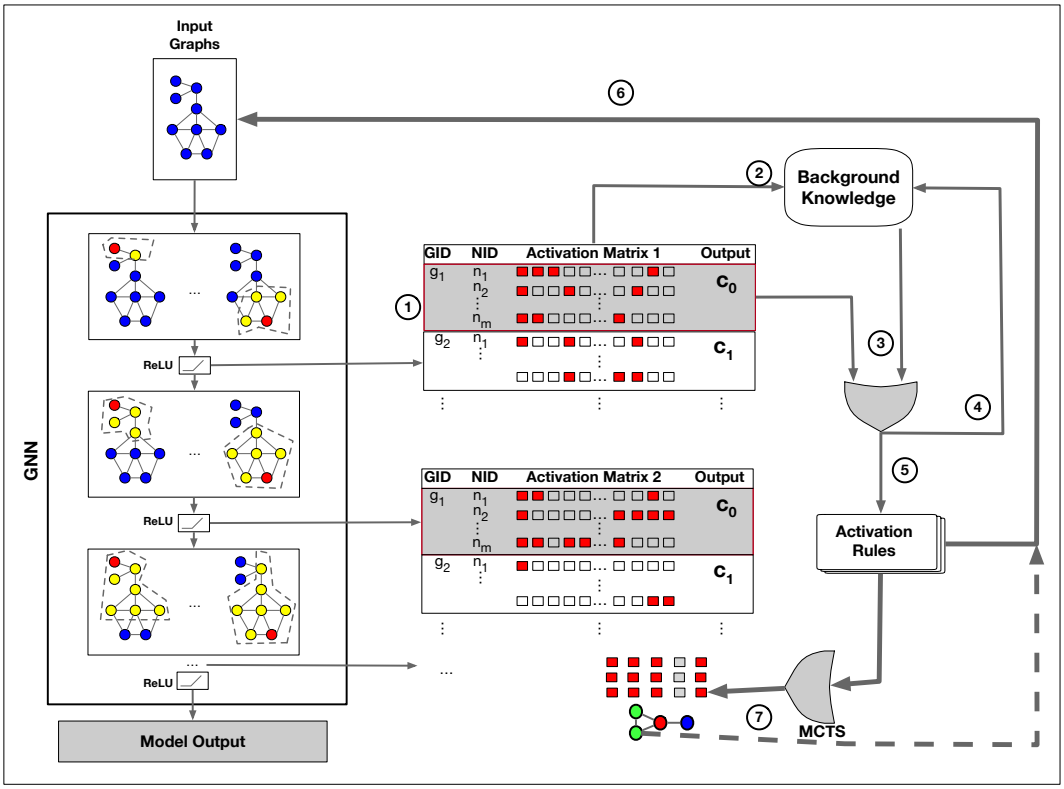

Figure 1. Overview of DISCERN. For each layer (1), a background model captures the activation distribution (2) used to assess the interest of activation rules (3). The most relevant rule is added to the pattern set (5) and used to update the background model. Steps (2-5) are repeated until no more informative activation rule is retrieved. Activation rules directly support instance level explanations (6) or are transformed into graph thanks to MCTS-based graph generation (7) to be easily interpreted by humans. Such graphs can also be used to explain single instance model decision.

A preliminary version of this work was published in [30]. This paper significantly extends our first attempt. The novel method we introduced is further described and discussed. We provide an extensive empirical study considering more datasets and state of the art methods. The main contributions of this paper are manifolds:

- We introduce the novel problem of characterizing internal representations of GNNs as well as our method DISCERN to generate realistic graphs that are representative of the activation rules. This method relies on a proximity measure between a graph and an activation rule. There are different ways to construct such measures and we propose three different ones.
- We report an empirical evaluation on several real-world datasets in Section 4 where we study the ability of DISCERN to provide good explanations with realistic graphs and compare the three metrics we introduce. DISCERN is also compared to six state of the art baselines. These experiments demonstrate that our method provides better and more realistic explanations.

The structure of this paper is as follows: First, we discuss the most important related work in Section 2. We formally define the problem of characterizing internal representations of GNNs and introduce our method DISCERN in Section 3. We report a thourough and extensive empirical study in Section 4. Finally, Section 5 concludes this paper and offers an outlook for future research.

2. Related work

GNNs are arousing wide interest thanks to their performance in several tasks such as node classification [20], link prediction [41] and graph classification [36, 34] Many cutting-edge techniques improve the performance of models as graph convolution [14], graph attention [28], and graph pooling [33]. 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 [3, 17]. As established by [40], 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 [26] provide non-exploitable results when they are applied to discrete adjacency matrices. Those that learn soft masks to find important regions of images [19] do not apply to discrete data as well. Though, some methods have been proposed to explain GNNs over the past four years. In [40], the authors propose a taxonomy to classify all the methods from the literature. Based on this taxonomy, one can identify two types of explanation methods: (i) the instance-level explanation methods and (ii) the model-level explanation methods.

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 explanation is provided through a *mask* which is a subgraph of the input graph one wants to explain the decision. This mask allows to highlight the part of the input graph that plays an important role in the final decision of the model. We can identify four different families of instance-level explanation methods.

- The *gradient/feature-based methods* [1, 21] – directly adapted from dedicated image and text solutions – use the gradients or hidden feature map values to compute the importance of the input features.
- The *perturbation-based methods* [38, 16, 10, 23, 32] aim at learning a graph mask by studying the prediction changes when perturbing the input graphs. GNNExplainer [38] is the seminal perturbation based method for GNNs. It learns a soft mask by maximizing the mutual information between the original prediction and the predictions of the perturbed graphs. Similarly, PGExplainer [16] uses a generative probabilistic model to learn succinct underlying structures from the input graph data as explanations. GraphMask [23] also trains a classifier to build an edge mask. Edges that do not affect the GNN prediction are removed till obtaining the final mask. Zorro [10] uses a greedy method to build discrete masks and identify important input nodes and node features. Causal Screening [32] aims at providing edge masks by selecting the edges with the highest causal effect, i.e., the edges that impact the prediction when adding.

- The *surrogate methods* [12, 31] explain an input graph by sampling its neighborhood and learning an interpretable model. GrapheLime [12] thus extends the LIME algorithm [22] to GNN in the context of node classification. It uses a Hilbert-Schmidt Independence Criterion Lasso as a surrogate model. However, it does not take into account the graph structure and cannot be applied to graph classification models. Relex [42] uses a GCN – which is not it interpretable – as the surrogate model. Thererfore, it uses a pertubation-based method to generate a soft mask. PGM-Explainer [31] builds a probabilistic graphical model for explaining node or graph classification models. Yet, it does not allow taking into consideration edges in its explanations. These surrogate models can be misleading because the user tends to generalize beyond its neighbourhood an explanation related to a local model. Furthermore, the identification of relevant neighborhood in graphs remains challenging.
- The *decomposition-based methods* [21, 24] start by decomposing the prediction score to the neurons in the last hidden layer. Then, they back-propagate these scores layer by layer until reaching the input space. Recently, [25] proposes GNN-LRP to assess the importance of different graph walks.

On top of that, GraphSVX [8] falls into these 4 categories by learning a surrogate explanation model on a pertubed 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 [39]. 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. Model introspection methods

There exists in the literature some rule extraction methods for DNNs [27], but not for GNNs. For example, Tran and d'Avila Garcez [27] mine association rules from Deep Belief Networks. Still, their approach suffers from an explosion of the number of patterns, which makes the results of frequency-based rule mining mostly unusable in practice. Also, with its focus on DBNs, the method is not directly applicable to standard GNNs. More recently, the authors of [9] define a method to identify the best set of rules in a CNN. These rules are selected thanks to the Minimum Description Length principle. Although promising, this method is limited to CNNs.

2.4. Limitations and desiderata

Most of the aforementioned methods aim at either explaining the final decision of a GNN or generating a representative graph for a given decision. We believe that focusing only on the model decision does not allow to fully understand how the model behaves and builds its decision. One can provide additional insights about the GNN by not only looking at the output of the model, but also by trying to characterize some representation subspaces that the model has built in the different layers. Several state of the art methods [16, 31, 38, 39] will be considered as baselines in our empirical study in Section 4.

3. Method

We propose an introspective method to explain GNN decisions in a post-hoc manner. The method identifies groups of neurons that work together in the decision-making process and associate a subgraph that triggers the corresponding classification rule. This problem is formalized below.

3.1. Problem definition

We consider a set of graphs $\mathcal{G}$ with labels: $G = (V, E, L)$ with V a set of vertices, E a set of edges in $V \times V$, and L a mapping between vertices and labels: $L \subseteq V \times T$, with T the set of labels. A GNN classifies each graph of $\mathcal{G}$ into two categories $\{0, 1\}$: GNN : $\mathcal{G} \rightarrow \{0, 1\}$. The GNN takes decisions at the level of each graph on the basis of vectors, the neurons, computed at the level of the nodes of each graph. For each node, ego-graphs of increasing radii are embedded in the Euclidean space in such a way that similar ego-graphs are associated to similar vectors. More precisely, we consider Graph Convolutional Networks (GCN) [14] that compute vectors $\mathbf{h}_v^\ell$ associated to the ego-graph centered in vertex v with radius ℓ, recursively by the following formula:

$$\mathbf{h}_v^\ell = ReLU\left(\mathbf{W}_\ell \cdot \sum_{w \in \mathcal{N}(v)} \frac{e_{w,v}}{\sqrt{d_v d_w}} \mathbf{h}_v^{\ell-1}\right).$$

$e_{v,w}$ is the weight of the edge between nodes v and w, $\mathcal{N}(v)$ is the set of neighboring nodes of v including v, ReLU is the rectified linear activation function, and $\mathbf{W}_\ell$ are the parameters learnt during the training phase of the model. We also have $d_v = \sum_{w \in \mathcal{N}(v)} e_{v,w}$. Finally, $\mathbf{h}_v^0$ is the initial feature vector for node v with the one-hot encoding of its label from T. 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.

Once the GNN learnt, the vectors $\mathbf{h}_v^\ell$ capture the key characteristics of the corresponding graphs on which the classification decision is made. When one of the vector components is of high value, it plays a role in the decision process. More precisely, activated components of the vectors (the indices k such that $(\mathbf{h}_v^\ell)_k > 0$) are combined by the neural network in a path leading to the decision, either 0 or 1. For a given layer ℓ, the activated components of the embedding $\mathbf{h}_v^\ell$ correspond to the part of the ego-graph centered in v and of radius ℓ that trigger the decision. We therefore construct the activation matrix that corresponds to the activated vector components and that is defined by

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

$\widehat{H^\ell}$ has dimensions $(n \times K)$, with $n = \sum_{g_i \in \mathcal{G}} |V_i|$.

The matrix $\widehat{H^\ell}$ constitutes the trace of the process performed by the GNN to classify the graphs. If it is able to correctly classify the graphs, it is because it identifies their key characteristics to assign them in one class or the other. However, these features are not clearly and directly accessible from $\widehat{H^\ell}$. This is why we propose in the following a method to identify these features and to represent them in an intelligible way for the user. This method has two steps. The first one identifies activation rules, i.e. the components of the matrix which are co-activated together for the graphs assigned to the same class. This method was published in [29] and we recall its principle in Section 3.2. The second step, and this is the main contribution of this paper, expresses these rules into a space of representation intelligible by the user: the vector of co-activated components is represented by a sub-graph making it possible to discriminate the graphs of the two classes. This step is detailed in Section 3.3.

3.2. Computing activation rules with INSIDE-GNN

3.2.1. Activation rules

We consider activation rules that are groups of vector components that are mostly activated together in graphs having the same GNN decision. Let us first define the rules and their supports.

Definition 1 (Activation rule and support). *An activation rule $\boldsymbol{A}^\ell \rightarrow c$ is composed of a binary vector $\boldsymbol{A}^\ell$ 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 $\widehat{H^\ell}[v, k] = (\boldsymbol{A}^\ell)_k$, $\forall k = 1 \cdots K$. It is denoted* **Activate**$(\boldsymbol{A}^\ell \rightarrow c, v)$.

The activated graphs with GNN decision c form the support of the rule:

$$\textbf{Supp}(\boldsymbol{A}^\ell \rightarrow c, \mathcal{G}) \quad = \quad \{g_i \in \mathcal{G} \mid \exists v \in V_i, \textbf{Activate}(\boldsymbol{A}^\ell \rightarrow c, v) \text{ and } GNN(g_i) = c\}.$$

Hence, 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. We propose to measure the interestingness of these patterns in a subjective manner. It makes possible to take into account *a priori* knowledge on activation components, but also to perform an iterative extraction of the rules and thus limiting the redundancy between them. These notions are explained below.

3.2.2. Measuring the interest of an activation rule

The question now is how to evaluate the interest of the activation rules so as to obtain a set of non-redundant rules. One way to achieve this is to model the knowledge extracted from the activation matrix into a background model and to evaluate the interest of a rule by the knowledge it brings in relation to it. This is what the FORSIED framework [5] does. It proposes an operational way to define the background model and to evaluate the subjective[1] interest of a pattern by using information theory to quantify both its informativeness and complexity.

We consider the discrete random variable $H^\ell[v,k]$ associated to the activation matrix $\widehat{H^\ell}[v,k]$[2], and we model the background knowledge by the probability $P(H^\ell[v,k]=1)$. Intuitively, the information content (IC) of an activation rule should increase when its components are unusually activated for the nodes in the graphs of its support (it is unlikely that these components are activated when considering a random node, while this probability increases when considering graphs supporting the pattern).

Thus, given the probabilities $P(H^\ell[v,k]=1)$ and with the assumption that all $H^\ell[v,k]$ are independent of each other, we can evaluate the interest of a rule by the product of $P(H^\ell[v,k]=1)$ for v activated by the rule and k such that $(A^\ell)_k=1$. Equivalently, we use the negative log-probability. The more probable the pattern – and therefore the less interesting – the smaller this value. As there may exist several nodes activated in a single graph, we choose the one maximizing the measure. This is formalized in the definition below.

Definition 2 (Rule information content). *Given a probabilistic background model P, the information provided by a rule $A^\ell \to c$ to characterize a set of graphs $\mathcal{G}$ is measured by*

$$IC(A^\ell \to c, \mathcal{G}) = \sum_{g_i \in \mathbf{Supp}(A^\ell \to c, \mathcal{G})} \max_{v \in V_i} - \sum_{k \text{ s.t. } (A^\ell)_k=1} \log(P(H^\ell[v,k]=1)) \,.$$

A pattern with a large IC is more informative, but it may be more difficult for the user to assimilate it, especially when its description is complex. To avoid this drawback, the IC value is contrasted by its description length which measures the complexity of communicating the pattern to the user. The higher the number of components in A^ℓ, the more difficult to communicate it to the user.

Definition 3 (Description length of a rule). *The description length of a rule is evaluated by*

$$DL(A^\ell \to c) = \alpha(|A^\ell|) + \nu$$

with α the cost for the user to assimilate each component and ν a fixed cost for the pattern. 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. In [6], Deng recommended to tune α to biase the results toward either shorter or longer patterns. Accordingly, α is set to 0.6 to express a slight preference toward shorter patterns.

The subjective interestingness measure is defined as the trade-off between IC and DL. 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.

Definition 4 (Subjective interestingness of a rule). *The subjective interestingness of a rule on the whole set of graphs $\mathcal{G}$ is defined by*

$$SI(A^\ell \to c, \mathcal{G}) = \frac{IC(A^\ell \to c, \mathcal{G})}{DL(A^\ell \to c)}$$

[1] Subjective means relative to the background knowledge model.
[2] We use hats to signify the empirical values.

If we denote by $\mathcal{G}^0$ (resp. $\mathcal{G}^1$) the graphs $g_i \in \mathcal{G}$ such that $GNN(g_i) = 0$ (resp. $GNN(g_i) = 1$), 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 $\omega_1 = \max(1, \frac{|\mathcal{G}^0|}{|\mathcal{G}^1|})$.

3.2.3. Computing the background model

The background model is initialized with basic assumptions about the activation matrix and updated as rules are extracted.

Definition 5 (Initial background model). *Some components can be activated more than others on all the graphs, or some nodes can activate a variable number of components. We assume that this information is known and use it to constrain the initial background distribution P:*

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

However, these constraints do not completely specify the probability matrix. Among all the probability distributions satisfying these constraints, we choose the one with the maximum entropy. Indeed, any distribution P with an entropy lower than the maximum entropy distribution effectively injects additional knowledge, reducing uncertainty unduly. The explicit mathematical MaxEnt model solution can be found in [4].

Once a rule $A^\ell \to c$ has been extracted, it brings some information about the activation matrix that can be integrated into P. The model must integrate the knowledge carried by this rule, that is to say that all the components with value 1 of A^ℓ are activated by the vertices activating the rule.

Definition 6 (Updating the background model). *Given a rule $A^\ell \to c$, the model P integrates the rule as follows*

$$P(H^\ell[v,k] = 1) \text{ is set to } 1$$

$\forall k$ such that $(A^\ell)_k = 1$ and v such that $\widehat{H^\ell}[v,k] = (\boldsymbol{A}^\ell)_k$, $\forall k = 1 \cdots K$.

3.3. Characterizing activation rules with subgraphs

Activation rules correspond to a part of the GNN (i.e. part of the matrix $\widehat{H^\ell}$) specifically activated for a given decision (for the graphs g such that $GNN(g) = c$). However, these rules are not intelligible, and do not allow highlighting the parts of the graphs which are used for the classification. To make the rule humanly understandable, we propose to associate to each rule a subgraph. To that end, we are looking for a subgraph whose GNN embedding in layer ℓ is as close as possible to a given activation rule. This requires defining a measure of proximity between embedding and activation rule (see section 3.3.1), ensuring that the graphs are realistic (see section 3.3.2) and defining a procedure to determine the subgraph that maximizes the proximity measure while being realistic (see section 3.3.3).

3.3.1. Measuring the proximity between a graph and an activation rule

To measure the proximity between a graph and a rule, we compute the embedding of the graph by the GNN and then we compare the embedding with the rule. We propose three different measures to evaluate the proximity between a graph embedding, centered at node v, $\mathbf{h}_v^\ell$ and an activation rule $\mathbf{A}^\ell \to c$, with $\mathbf{A}^\ell = \{a_1, \ldots, a_K\}$, $a_i \in \{0, 1\}$. These measures use $\mathbf{E}_g = \{\epsilon_1, \ldots, \epsilon_K\}$ the ego-graph embedding truncated to fit the interval $[0, 1]$: $\epsilon_k = \min\left(\max(0, (\mathbf{h}_v^\ell)_k), 1\right)$. The truncation avoids distorting our measurements by extreme values.

The first measure is the cosine similarity measure:

$$\text{Cosine}(\mathbf{E}_g, \mathbf{A}^\ell) = \frac{\mathbf{E}_g \,.\, \mathbf{A}^\ell}{\| \mathbf{E}_g \| \| \mathbf{A}^\ell \|} = \frac{\sum a_i \epsilon_i}{\sqrt{\sum a_i^2} \sqrt{\sum \epsilon_i^2}} \tag{1}$$

It is defined to equal the cosine of the angle between the two vectors, or identically be the inner product of the vectors normalized to length 1.

We can also consider to use the cross-entropy measure (equivalently the log-likelihood):

$$\text{Cross-entropy}(\mathbf{E}_g, \mathbf{A}^\ell) \quad = \quad \sum a_i \log(\epsilon_i) \tag{2}$$

that increases with the number of components that have large values in the embedding of the ego-graph for the components activated in the activation rule.

We can also consider a set of activation rules R and search for an ego-graph that specifically activates a rule $\mathbf{A}^\ell$ but not the other rules, i.e. this ego-graph activates components outside the rule only if it does not trigger another rule. We propose the following expression to measure this:

$$\text{Relative-CE}(\mathbf{E}_g, \mathbf{A}^\ell, R) = \text{Cross-entropy}(\mathbf{E}_g, \mathbf{A}^\ell) - \max_{\mathbf{r} \in R} \text{Cross-entropy}(\mathbf{E}_g, \mathbf{r}) \tag{3}$$

3.3.2. Realism factor

Graphs can have an embedding close to an activation rule without being realistic and be very different from graphs in the dataset. To avoid this, we associate a realism score with each graph. This factor depends both on the probability that two vertices are connected according to their type, and on the degree distribution for each vertex type.

Let $P_{i,j}$ be the probability of having an edge with endpoints of type i and j ($i, j \in T$). Let $P_{deg}(k|i)$ be the probability for a node of type i of being of degree k. As we are considering subgraphs of the graphs of the dataset, we do not want to penalize graphs whose degree is smaller than expected. Thus, we propose to use the value $d_{k|i} = P_{deg}(k|i) + \sum_{x>k} \frac{P_{deg}(x|i)}{2k}$ to increase the value with the probabilities associated with higher degrees. We transform this value in a probability measure with $D_{k|i} = \frac{d_{k|i}}{\sum_x d_{x|i}}$. Thus, the realism score is calculated by

$$Realism(g = (V, E, L)) = \frac{\sum_{(u,v)\in E} \log\left(P_{L(u),L(v)}\right)}{\#E} + \frac{\sum_{u \in V} \log(D_{d_u|L(u)})}{\#V} \tag{4}$$

with $L(u)$ the type of node u and d_u its degree. This realism value is added to the similarity measure between the activation rule and the graph embedding to form the score used to evaluate a graph quality:

$$\textbf{Score}(g, A^\ell, R) = \beta \times m(\mathbf{E}_g, \mathbf{A}^\ell, R) + (1 - \beta) \times Realism(g) \tag{5}$$

with β a hyperparameter whose value is fixed empirically (see section 4) and m one of the tree measures defined by equations (1)–(3).

3.3.3. DISCERN method

We propose to use Monte Carlo Tree Search (MCTS) to find a realistic subgraph whose embedding is similar to an activation rule. The objective is to generate an ego-graph g_t that maximizes $\textbf{Score}(\mathbf{E}_t, \mathbf{A}^\ell, R)$. Each node t of the search tree represents an ego-graph g_t centered at v_t whose GNN embedding is $\mathbf{E}_t$. A value M_t is also associated to t: it is the sum of $\textbf{Score}(\mathbf{E}_t, \mathbf{A}^\ell, R)$ values evaluated on the terminal nodes of the subtree rooted in t.

Each node of the tree is obtained by adding an edge to the graph of its parent node. This process stops when a terminal condition is satisfied. In our algorithm, we consider three terminal conditions:

1. The diameter of the graph is greater than ℓ,
2. The number of edges is greater than *min-edges*,

3. The number of vertices is greater than *min-vertices*.

If one of the three conditions mentioned above is satisfied, the graph is considered terminal (see method **isTerminal** in function **findChild** line 3, and in Function **rollout** line 2).

The tree is partially explored favoring the parts most likely to lead to high-scoring graphs. From an intern node t, terminal nodes (satisfying one of the terminal conditions) are randomly generated in order to be able to evaluate their score. This process is called rollout. The obtained score is then back-propagated to all ancestors of t in their variable M_t. The rational behind the value M_t is to estimate the quality of the descendants of t explored so far. This value is then adjusted to trade-off exploitation of the current examined graphs, and exploration of new ones. We use the classical the Upper Confidence Bound UCB1 to guide the selection of the tree node to be expanded: We select the one that maximizes UCB1.

$$UCB1(t) = \frac{M_t}{n_t} + c\sqrt{\frac{\log(N(t))}{n_t}},$$

where c is a constant number, M_t is the sum of the values **Score** for all terminal nodes descendant of t that have been explored so far, n_t is the number of terminal nodes expanded from node t during previous iterations of the algorithm, and $N(t)$ is the number of times the parent of t has been visited so far. It should be mentioned that the constant c in UCB1 plays an important role to make a balance between exploration and exploitation. If c is too large, the algorithm acts like a pure random algorithm (i.e. more exploration) and if c is too small, then exploitation rate will be increased and may get stuck in local maxima. In our experiments, c is set to 0.5.

Let's consider how DISCERN works in detail. Algorithm 1 starts by building the tree root node, that is a leaf, its number of visits n_t equals 0 as well as its value M_t. DISCERN consists of some iterations (epochs) whose number is an input parameter N (line 6 to 16 of **DISCERN**). Each iteration starts by calling the method **ExploreChild** (line 5 of **DISCERN**) on the root node. This method returns the next tree node that had to be explored, that is to say the one 1) whose path from root is made of nodes with maximal UCB1 values among their brothers, 2) that is a leaf and 3) that has still to be explored (it is not a terminal node). If the current node is not a leaf (line 1 of **exploreChild**), **findChild** is called to take among the children that are not terminal, the one with the best UCB1 value. If such a node exists, **exploreChild** is recursively called on it (line 4). Otherwise, the subree has completely been explored and the exploration goes up to the parent node to examine another branch (line 6).

Then, **DISCERN** explores the identified node. If the node has never being expanded (t.visit = 0 at line 8), then a rollout is performed and the obtained value is back-propagated to the tree root. Otherwise, the node is expanded with children and a rollout is performed on the first whose value is back-propagated to the root. These three functions (**generateChildren**, **rollout**, **backPropagate**) are as follows. **generateChildren** creates as many children then there are possible graphs with one edge in addition to the current graph. Those edges can be between two nodes of the graph ($v \in V$ line 6) or between a node of the graph and a new node with on e of the possible labels from T ($v \in W$ line 6). These edges have to be valid (see Figure 2 and explanation below). The (arbitrary) first child is return by the function. The function **rollout** simulates a new graph created by taking repeatedly uniformly at random a valid edges and adding to the current graph until the graph is terminal. It returns the graph and its score. If the score is better than the one encountered so far, the *best_graph* is updated (see lines 17 to 20 of **DISCERN**). Finally, **backPropagate** updates the M_t and n_t values of the tree nodes ($t.value$ and $t.visit$) until reaching the tree root.

Only valid edges can be added to a current graph g centered in node v with radius r. An edge is valid (see **isValid** method in **rollout** and **generateChildren**) if adding it does not change the graph embedding in layer $r-1$. That is to say, for each node $x \in g$, $dist(v, x)$ does not change when adding the edge. To that end, three conditions have to be met:

- Adding an edge cannot reduce the shortest distance of a node to v. The counter example (a) in Figure 2 illustrates this condition.
- An edge can be added between two nodes of g, if their distance to v is greater or equal to $radius - 1$. The counter example (b) in Figure 2 illustrates this condition as well as examples (2) and (3).
- An edge between a node x of the graph and a new node can be added only if $dist(v, x) \geq radius - 1$. The counter example (c) in Figure 2 illustrates this condition, as well as examples (1) and (4).

Algorithm 1 DISCERN

Require: N: Number of epochs, **Score** (.,$\mathbf{A}^\ell$, R): the measure to maximize, T: the set of node labels.
Ensure: *best_graph* the graph that maximizes **Score**($\mathbf{E}_g$, $\mathbf{A}^\ell$, R) among all explored graphs.

```
root.leaf ← True
root.visit ← 0
root.value ← 0
best_value ← −∞
for epoch = 1 to N do
    t ← exploreChild(root)
    if (t.leaf = True) then
        if (t.visit = 0) then
            [value, explored_graph] ← rollout(t)
            backPropagate(t,value)
        else
            first_child ← generateChildren(t)
            [value, explored_graph] ← rollout(first_child)
            backPropagate(first_child, value)
        end if
    end if
    if (value > best_value) then
        best_value ← value
        best_graph ← explored_graph
    end if
end for
return best_graph
```

exploreChild(t)

```
if (t.leaf = False) then
    best_child ← findChild(t)
    if (best_child ≠ None) then
        exploreChild(best_child)
    else
        exploreChild(t.parent)        ▷ the subtree rooted in t has been completely explored
    end if
else
    return t
end if
```

findChild(t)

```
best_child ← None
for s ∈ children(t) do
    if (isTerminal(s.g) = False) then
        if UCB1(best_child) ≤ UCB1(s) then
            best_child ← s
        end if
    end if
end for
return best_child
```

Algorithm 2 Additional sub-functions

generateChildren(t):

first_child ← **None**
$g \leftarrow$ the graph associated to t
$V \leftarrow$ the vertices of the graph g
$W \leftarrow$ a set of new vertices with label in T
for $u \in V$ **do**
    for $v \in V \cup W$ **do**
        if (isValid(u,v)) **then**
            g' is the graph g with edge (u, v)
            $t' \leftarrow$ t.add_child(g')
            $t'.leaf \leftarrow$ **True**, $t.value \leftarrow 0$, $t.visit \leftarrow 0$
            if (first_child = **None**) **then**
                first_child ← t'
            end if
        end if
    end for
end for
return first_child

rollout(t):

$g, g_2 \leftarrow$ the graph associated to t
while (isTerminal(g) = **False**) **do**
    $g_2 \leftarrow g$
    Take a random edge e among the valid edges that can be added to g
    $g \leftarrow$ Simulate a new graph g with edge e added
end while
return [**Score**($\mathbf{E}_g, \mathbf{A}^\ell, R$), g_2]

backPropagate(t,value):

while (t ≠ **None**) **do**
    $t.visit \leftarrow t.visit + 1$
    $t.value \leftarrow t.value + value$
    $t \leftarrow t.parent$
end while

4. Experiments

In this section, we evaluate DISCERN through several experiments. We first describe synthetic and real-world datasets and the experimental setup. Then, we discuss some examples of subgraphs generated by our method and the baselines. Eventually, we present a quantitative study of our method as well as some comparisons against several baselines. DISCERN has been implemented in Python and the experiments have been done on a machine equipped with 8 Intel(R) Xeon(R) W-2125 CPU @ 4.00GHz cores 126GB main memory, running Debian GNU/Linux. The code and the data are available [3].

4.1. Datasets and experimental setup

Experiments are performed on six graph classification datasets whose main characteristics are given in Table 1. BA2 [38] is a synthetic dataset generated with Barabasi-Albert graphs and hiding either a 5-cycle (negative class)

[3] `https://doi.org/10.5281/zenodo.7208320`

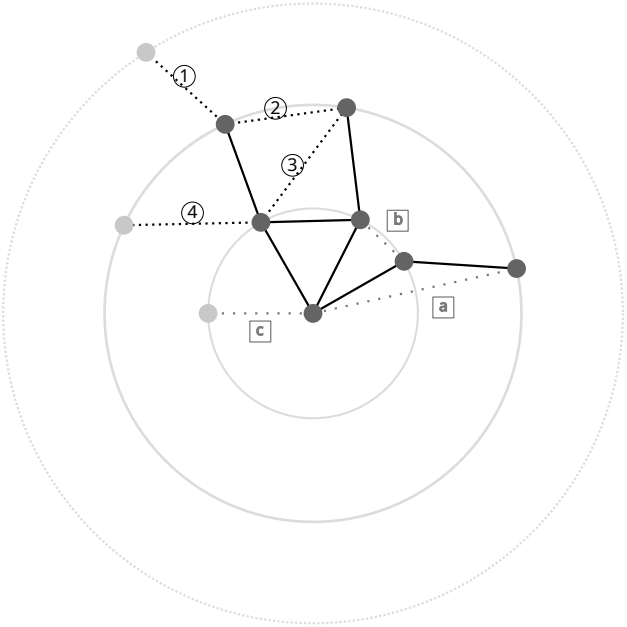

Figure 2. Considering a graph (solid lines), some edges can be added (dark dot lines), while other cannot (light gray lines). **Examples –** *(1) and (4)*: edge with a new node and a node at distance from center ≥ *radius* − 1; *(2)* edge between two nodes at a distance to the center equal to the radius of the graph; *(3)* edge between a node at distance *radius* to the center, and a node at distance *radius* − 1. **Counter examples –** *(a):* a new edge cannot create a shortcut between existing nodes; *(b):* no edge can be added between nodes that both are at a distance < *radius* from the center of the graph; *(c):* a new node cannot be connected to a node at distance < *radius* − 1.

or a "house" pattern (positive class). The other datasets (Aids [18], BBBP[35], Mutagen [18], DD [7], Proteins [2]) correspond to real molecules, and the class identifies important properties in Chemistry or Drug Discovery (i.e., possible activity against HIV, permeability and mutagenicity). Table 1 shows that the datasets are diverse, each having its own specificity. BA2 is synthetic and simple to interpret. BBBP and Aids are very unbalanced. DD has a large number of node attributes and is made of large graphs. Mutagen and Proteins are similar datasets, with Proteins graphs denser than Mutagen ones. All these characteristics witness that the benchmarks we consider are diverse. This supports thorough and systematic experimental study.

A 3-convolutional layer GNN (with $K = 20$) is trained on each dataset. Accuracy measures obtained on test sets are given in Table 1 column Acc. We use the method INSIDE-GNN to mine the GNN activation vectors $\mathbf{h}_v^\ell$ to discover the activation rules $\mathbf{A}^\ell$. We extract at most ten rules per layer and for each class $\{0, 1\}$ as explained in [29]. On some datasets, less than 10 rules per layer and per class are needed to describe the inner workings of a GNN (see Table 1 column #Rules). Our goal is to provide a representative graph for each rule with DISCERN. We generate graphs with labels appearing in at least 100 nodes in the dataset (see Table 1 column #Freq_T).

Table 1. Main characteristics of the datasets.

Dataset	$\#\mathcal{G}$	(# 0,#1)	#T	$\overline{V}$	$\overline{E}$	Acc.	#Rules	#Freq_T
DD	1168	(681, 487)	90	268	1352	0.692	47	21
Aids	2000	(400, 1600)	38	15.69	32.39	0.99	60	7
Mutagen	4337	(2401, 1936)	14	30.32	61.54	0.786	60	10
BBBP	1640	(389, 1251)	13	24.08	51.96	0.787	60	6
Proteins	1113	(663, 450)	3	39	145	0.768	29	3
BA2(syn)	1000	(500, 500)	-	25	50.92	0.97	20	-

This experimental study aims to answer the following questions:

- How does DISCERN behave?
- Are the activation rules good?
- Are the generated graphs representative and realistic?
- Which is the best metric?
- How does DISCERN behave against baselines?

To that end, especially the latter question, we compare our method against to both instance-level and model-level explanation methods. For instance-level methods, we consider four state of the art methods: **GNNExplainer** [38], **PGExplainer** [16], **PGM-Explainer** [31], and **GraphSVX** [8].

We also examine 3 model-level baselines:

- **Random** generates graphs randomly by calling the Roll-out function 250x.
- **XGNN++** is an extension of XGNN [39] to our problem. We integrate Cos, CE and Relative-CE metrics as function optimized by XGNN. We set a budget that corresponds to 5000 calls to the GNN to do a fair comparison with DISCERN so that both methods do exactly the same number of calls to the GNN.
- **DISC-GSPAN** is a sound and complete method that aims at discovering discriminant subgraphs within a collection based on GSPAN enumeration [37] while exploiting some tight upper bounds on the WRAcc measure [15]. The input dataset contains the set of ego-network of nodes that support the activation rule as the "positive class" and the ego-network of nodes not involved in the support of the rule as the negative class. Then, DISC-GSPAN consists in computing the top-k subgraphs that are discriminant for the positive class.

By default, we empirically set *min-edges*=10 and *min-vertices*=6 as terminal conditions for DISCERN in the following experiments. We also empirically set the hyper-parameter β to 0.5 (see Figure 5).

4.2. Studying DISCERN behavior

For an activation rule, DISCERN takes between 20 and 50 seconds for 5000 epochs. The deeper the GNN layer, the larger the ego-graph to be investigated, the higher the execution time.

4.2.1. Quality with respect to the number of epochs

Figure 3 (a–c) shows the maximum value of measure m (Cosine, Cross-entropy and Relative-CE) obtained on the graphs generated by DISCERN for each dataset with respect to the number of epochs. The value on the y-axis is the maximum value of the measure m evaluated on the explored graphs in the MCTS (*explored_graph* lines 9 and 13 in **DISCERN** function) at the corresponding x-axis epoch. For each dataset, the values are aggregated over all activation rules. The graphs show an asymptotic convergence for all the curves. Yet, the convergence is faster for some combinations of metric and dataset (e.g., Cosine on Mutagen) than others (e.g., Cosine on DD). This experiment demonstrates that DISCERN correctly learns which parts of the MCTS search space hold promise for generating high value graphs.

4.2.2. Are the results actually good?

We observe in the latter experiment that the graphs generated by DISCERN become better when increasing the number of epochs. However, one can wonder if the obtained graphs g_{mcts} are actually good compared to the ego-graphs $g_{support}$ that support the activation rules. Especially, we want to answer the following question:

Are the obtained scores higher than those of the ego-graph taken randomly in the dataset?

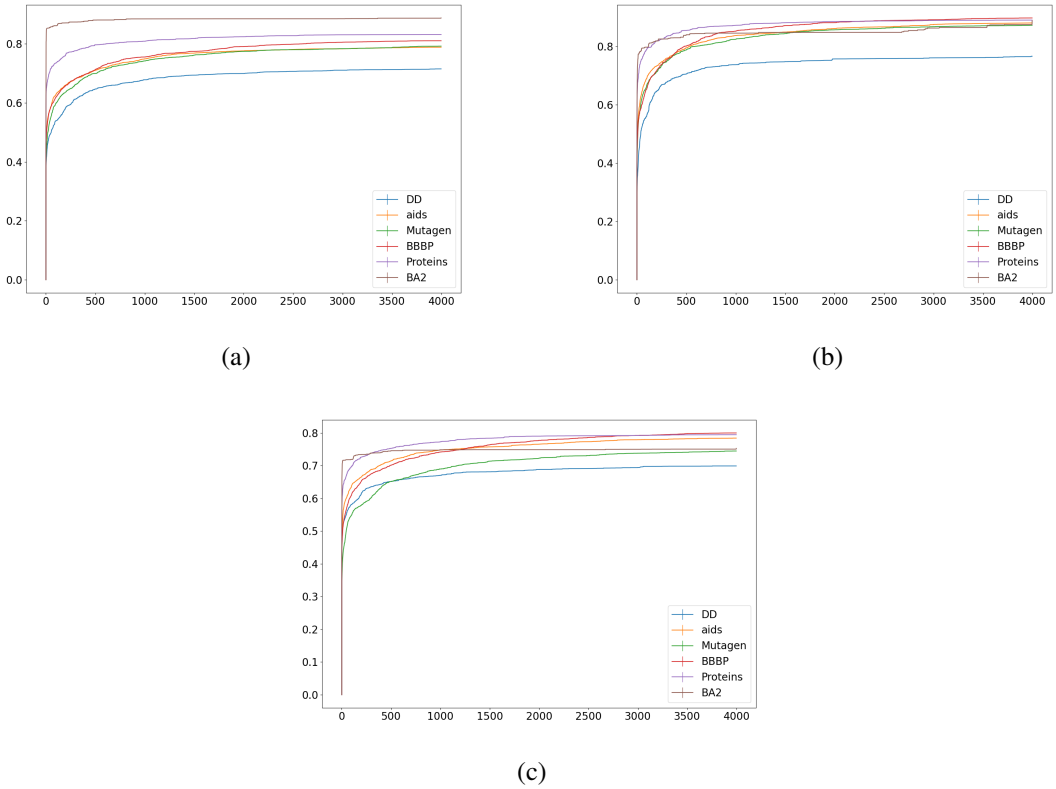

Figure 3. Maximum m values on the graphs generated by DISCERN for the metrics m (a) Cosine, (b) Cross-entropy, (c) Relative-CE on each dataset when varying the number of epochs.

To this end, we consider a sample of nodes known to be embedded in the targeted subspace as they support the activation rules. Similarly, we take a sample of random nodes. For both samples respectively denoted $E_{g_{sample}}$ and $E_{g_{rand.}}$, we consider the ego-graphs whose size equals the layer of the corresponding activation rule. For each sample, we compute the score values for each quartile (Q1, Q2, Q3) that partition the sample ordered by values into 4 subsets of equal sizes. We also consider the maximal value Q_{max}. We report in Table 2 the improvement factor of the graphs produced by DISCERN compared to each quartile, i.e. the ratio $\textbf{Score}\ (E_{g_{mcts}}, A^{\ell}, R)/Q_x$. Results are aggregated over all activation rules for each dataset.

Interestingly, we observe that, in most of the cases we obtain values greater than 1, even when we compare the generated graphs to Q_{max} (i.e., max value of the sample). This demonstrates the high quality of the graphs generated by DISCERN. Having values greater than 1 for Q_{max} of the supporting node sample means that the graphs we generate embed well in the targeted subspace, with less activated components outside this space than for supporting nodes. Indeed, the metrics we consider penalize activated components of the ego-graph that are not activated in the activation rules. Obviously, the improvement factor is always better when considering the random sample than the node support sample. Nevertheless, this gives interesting insights showing the ability of metrics to separate support nodes from random nodes well. It is important to notice that the difference between improvement factors between the samples $E_{g_{sample}}$ and $E_{g_{rand.}}$ is much larger for Cosine and Cross-Entropy than Relative-CE, especially for Q_3 and Q_{max}. Based on these observations, we can conclude that Cosine and Cross-Entropy metrics has a better ability to separate better than Relative-CE.

4.2.3. Which is the best metric?

To deeper investigate the quality of the generated graph, we now study in Figure 4 the L2 norm between $E_{g_{mcts}}$ and, $E_{g_{sample}}$ or $E_{g_{rand.}}$. We observe that both CE and Relative-CE tend to provide graphs whose embeddings are more

Table 2. Avg. improvement factor of the score provided by DISCERN against the score of the quartiles of two distributions: (1) nodes from rule support (supp) and (2) some random nodes (rand).

Dataset	Measure	Samp.	Q_1	Q_2	Q_3	Q_{max}
DD	Cosine	supp	1.19×10^8	2.72	1.56	1.28
	Cosine	rand	8.12×10^9	5.04×10^9	2.83×10^9	1.03×10^9
	Cross-Entropy	supp	1.26	1.16	1.14	1.12
	Cross-Entropy	rand	9.57×10^{11}	4.21×10^{11}	2.33×10^{11}	1.09×10^{11}
	Relative-CE	supp	1.35	1.05	0.96	0.89
	Relative-CE	rand	3.79×10^{11}	3.14	2.37	1.68
Aids	Cosine	supp	1×10^8	1.94	1.63	1.46
	Cosine	rand	7.34×10^9	1.52×10^9	7.50×10^8	2.30×10^8
	Cross-Entropy	supp	1.25	1.17	1.15	1.14
	Cross-Entropy	rand	9.48×10^{11}	1.44×10^{11}	6.42×10^{10}	7.82×10^9
	Relative-CE	supp	1.35	1.10	1.04	0.99
	Relative-CE	rand	4.02×10^{11}	3.66	2.37	1.74
Mutagen	Cosine	supp	12.59	2.69	2.14	1.66
	Cosine	rand	7.33×10^9	1.42×10^9	1.01×10^9	3.84×10^8
	Cross-Entropy	supp	1.22	1.16	1.15	1.13
	Cross-Entropy	rand	1.02×10^{12}	1.28×10^{11}	1.01×10^{11}	3.51×10^{10}
	Relative-CE	supp	1.32	1.17	1.12	1.06
	Relative-CE	rand	4.38×10^{11}	6.71	3.76	2.17
BBBP	Cosine	supp	5.27	2.31	1.78	1.58
	Cosine	rand	7.61×10^9	1.16×10^9	5.19×10^8	1.37×10^8
	Cross-Entropy	supp	1.21	1.15	1.14	1.13
	Cross-Entropy	rand	9.86×10^{11}	1.16×10^{11}	6.22×10^{10}	1.17×10^{10}
	Relative-CE	supp	1.27	1.09	1.03	0.99
	Relative-CE	rand	4.20×10^{11}	11.33	2.18	1.61
Proteins	Cosine	supp	4.49	1.67	1.45	1.29
	Cosine	rand	5.61×10^9	3.1×10^8	3.80	1.71
	Cross-Entropy	supp	1.32	1.22	1.20	1.19
	Cross-Entropy	rand	1.01×10^{12}	1.68×10^{10}	10.72	3.82
	Relative-CE	supp	1.31	1.06	1.00	0.96
	Relative-CE	rand	3.72×10^{11}	5.16	2.11	1.44
BA2	Cosine	supp	1.84×10^7	4.54	2.82	1.78
	Cosine	rand	7.25×10^9	1.77×10^9	7.71×10^8	43.69
	Cross-Entropy	supp	5.01×10^9	2.48	2.02	1.75
	Cross-Entropy	rand	1.29×10^{12}	1.23×10^{11}	8.59×10^{10}	1.26×10^{10}
	Relative-CE	supp	1.24	1.00	0.98	0.99
	Relative-CE	rand	5.69×10^{11}	3.74×10^5	2.74	1.63

similar of the support sample than Cosine. Cosine reports greater L2 norm to $E_{g_{rand.}}$ than the two other measures but this is not significant.

Similarly, we investigate the distances between the graphs generated by DISCERN with the three metrics, the graphs from the support sample and the ones form the random sample, for each activation rule and dataset. We consider the graph edit distance [11], an error tolerant matching technique between graphs that is computed directly from the graphs and not their embedding. Results are reported in Table 3. We consider, for each rule and each measure, the set G_m of graphs produced by 10 runs of DISCERN, or a subset of 10 graphs from the support or the random samples. For set $m1$ in row and set $m2$ in column, we report the value $\text{mean}_{g \in G_{m1}} \max_{h \in G_{m2}} \frac{GED(g,h)}{\#V_g + \#E_g}$, with GED, the graph edit distance. Results are similar to what we observe when studying the L2 norm between vectors: Graphs from the support sample are closer to graphs generated by DISCERN with Cross-Entropy than those generated with Relative-CE and Cosine metrics (see supp related rows in Table 3). Furthermore, graphs generated by DISCERN– particularly with Cross-Entropy metric – are far away from the graphs of the random samples (see rand related rows in Table 3).

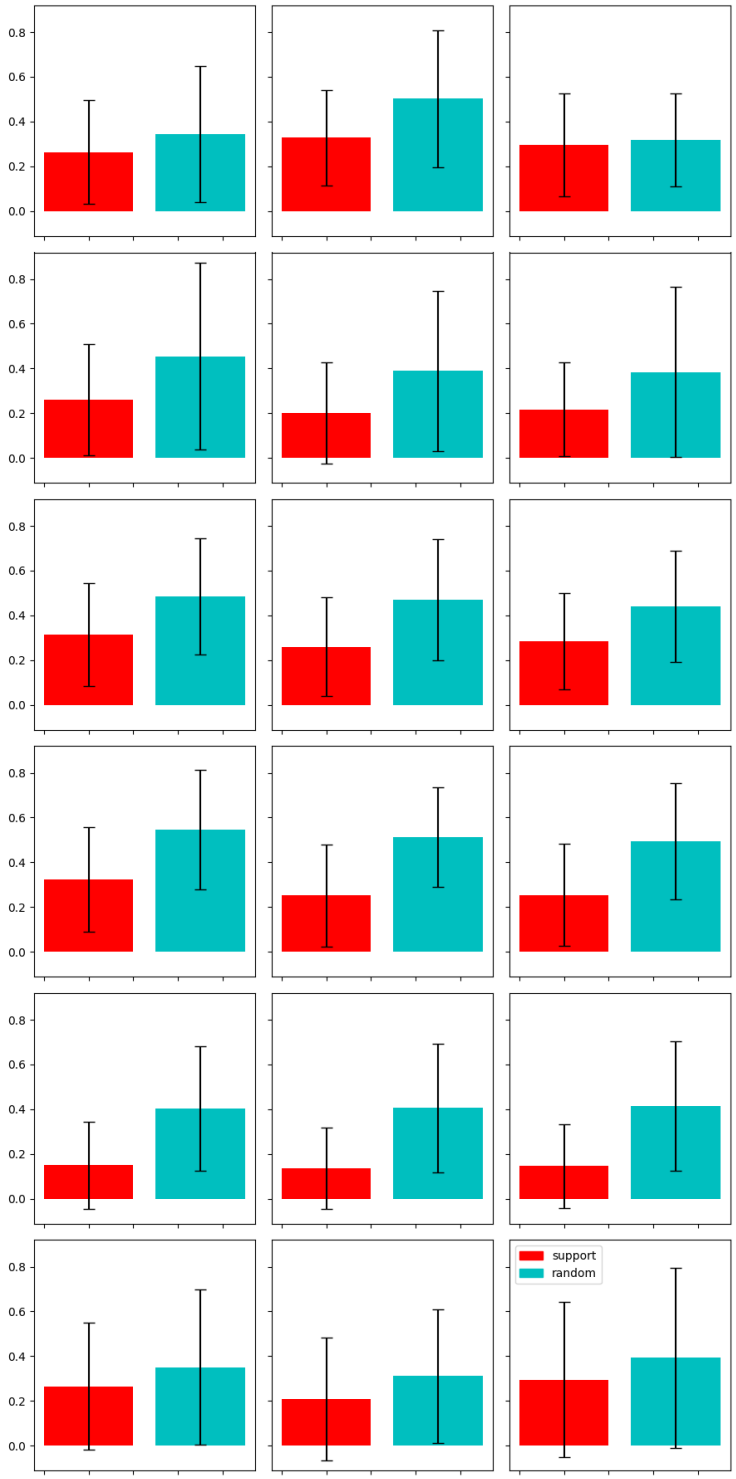

Figure 4. Average L2 norm between the generated graphs and support (red) and random (cyan) nodes for all activation rules. Dataset are (from top to bottom): DD, Aids, Mutagen, BBBP, Proteins, BA2. Left, middle and right columns depict Cosine, Cross-Entropy and Relative-CE measures. The lower the red histogram, the higher the cyan one, the better.

Table 3. Graph edit distances on Aids, Mutagen and BBBP. Distances are normalized by (#vertices + #edges) of the row graphs.

Aids	Cosine	Cross-Entropy	Relative-CE	supp	rand
Cosine	0.	0.751	0.755	0.582	0.647
Cross-Ent.	0.746	0.	0.643	0.499	0.654
Relative-CE	0.749	0.658	0.	0.517	0.661
Supp.	0.747	0.725	0.724	0.	0.526
Rand.	0.753	0.758	0.734	0.391	0.
Mutagen	Cosine	Cross-Ent.	Rel.-CE	Supp.	Rand.
Cosine	0.	0.755	0.712	0.653	0.66
Cross-Ent.	0.73	0.	0.622	0.61	0.678
Relative-CE	0.726	0.683	0.	0.599	0.672
Supp.	0.696	0.654	0.643	0.	0.419
Rand.	0.731	0.712	0.689	0.376	0.
BBBP	Cosine	Cross-Ent.	Rel.-CE	Supp.	Rand.
Cosine	0.	0.701	0.713	0.609	0.633
Cross-Ent.	0.689	0.	0.630	0.559	0.625
Relative-CE	0.718	0.657	0.	0.559	0.622
Supp.	0.694	0.645	0.636	0.	0.4
Rand.	0.716	0.685	0.663	0.352	0.

These experiments demonstrate that the ability of DISCERN to generate representative graphs for the activation rules regardless of the three measures Cosine, Cross-Entropy, and Relative-CE. These experiments suggest that Cross-Entropy is slightly better than the two other measures. Nevertheless, the difference are not significant.

4.3. Comparison to instance-level methods

We consider a ground-truth free metric to compare the methods. We opt for the *Fidelity* [21] which is defined as the difference of predicted probability between the predictions on the original graph and the one obtained when masking part of the graph based on the explanations:

$$\text{Fidelity} = \frac{1}{N} \times \sum_{i=1}^{N} \left(f(g_i)_{y_i} - f(g_i \setminus m_i)_{y_i} \right)$$

where m_i is the mask, $g_i \setminus m_i$ is the complementary mask and $f(g)_{y_i}$ is the prediction score for class y_i. Similarly, we can study the prediction change by keeping important features (i.e., the mask) and removing the others as Infidelity measure does:

$$\text{Infidelity} = \frac{1}{N} \times \sum_{i=1}^{N} \left(f(g_i)_{y_i} - f(m_i)_{y_i} \right)$$

The higher the fidelity, the lower the infidelity, the better the explanation.

Obviously, masking all the input graph would have important impact to the model prediction. Therefore, the former measures should not be studied without considering the *Sparsity* metric that aims to measure the fraction of graph selected as mask by the explainer:

$$\text{Sparsity} = \frac{1}{N} \sum_{i=1}^{N} \left(1 - \frac{|m_i|}{|g_i|} \right),$$

where $|m_i|$ denotes the size of the mask m_i and $|g_i|$ is the size of g_i (the size includes the number of nodes, of edges and the attributes associated to them). Based on these measures, a better explainability method achieves higher fidelity, lower infidelity while keeping a sparsity close to 1.

Table 4. Assessing the explanations with several metrics. A better explainer achieves higher fidelity, lower infidelity while keeping a sparsity close to 1.

Models \ Datasets	DD	Aids	Mutagen	BBBP	Proteins	BA2
(a) Fidelity						
AR(node)	**0.490**	**0.175**	**0.582**	**0.362**	**0.359**	0.342
DISCERN(Cosine)	0.124	0.031	0.319	0.172	0.086	**0.471**
DISCERN(Cross-Entropy)	0.130	0.026	0.360	0.188	0.072	0.460
DISCERN(Relative-CE)	0.113	0.054	0.366	0.198	0.070	0.446
GnnExplainer	0.077	0.036	0.177	0.100	0.021	0.093
PGExplainer	0.070	0.032	0.157	0.098	0.019	0.004
PGM-Explainer	0.059	0.080	0.123	0.212	0.073	0.222
SVXexplainer	0.010	0.003	0.039	0.008	0.006	0.004
(b) Infidelity						
AR(node)	0.133	0.767	0.237	0.374	0.160	**0.000**
DISCERN(Cosine)	0.239	0.119	0.290	0.193	0.069	0.149
DISCERN(Cross-Entropy)	0.228	0.125	0.196	0.411	0.056	0.124
DISCERN(Relative-CE)	0.271	0.039	0.195	0.134	0.076	0.246
GnnExplainer	**0.075**	**0.036**	**0.140**	0.099	**0.021**	0.223
PGExplainer	0.082	0.038	0.157	**0.098**	0.024	0.353
PGM-Explainer	0.343	0.766	0.347	0.482	0.324	0.296
SVXexplainer	0.343	0.771	0.356	0.489	0.292	0.341
(c) Sparsity						
AR(node)	0.769	0.897	0.731	0.870	0.249	0.002
DISCERN(Cosine)	**0.983**	0.664	0.741	0.630	0.649	0.641
DISCERN(Cross-Entropy)	0.977	0.721	0.742	0.623	0.645	0.674
DISCERN(Relative-CE)	0.979	0.478	0.195	0.707	0.669	0.676
GnnExplainer	0.502	0.501	0.505	0.501	0.986	0.804
PGExplainer	0.529	0.547	0.515	0.534	0.545	**0.955**
PGM-Explainer	0.976	0.862	0.900	**0.973**	0.957	0.746
SVXexplainer	0.965	**0.988**	**0.931**	0.940	**0.991**	0.943

Several policies to build a mask directly from an activation rule are possible. We opt for the simplest policy **AR(node)** which takes as a mask only the nodes that are covered by the activation rule and the edges adjacent to these nodes. This policy allows assessing how relevant are the activation rules. We also consider the graphs generated by DISCERN as masks.To this end, for an instance graph, we select among all the rules that are activated, the related generated graph that maximizes the trade-off between Fidelity and Infidelity. The selected subgraph is then used as a mask for explanation.

The average time to provide an explanation ranges from 8ms to 30ms for AR(node). This is faster than PGM-Explainer (about 5s), GNNExplainer (80ms to 240ms) and SVXexplainer (40ms to 60ms). It remains slightly slower than PGExplainer (6ms to 20ms). DISCERN is slower than AR(node) (about 1s). Even if the graph is already built for each activation rule, it requires several graph inclusion computations to provide a mask for an instance.

Table 4(a) outlines the performance of the explainers based on the Fidelity measure. Results show that **AR(node)** outperforms the baselines. These results must be analysed while considering the sparsity (see Table 4(c)). Except for Proteins and BA2, **AR(node)** provides explanations which have a comparable sparsity to the baselines. The quality of the explanations are also assessed with the Infidelity metrics in Table 4(b). **AR(node)** is outperformed by GnnExplainer and PGExplainer. This suggests that a rule taken in isolation does not allow a correct classification of a graph. It is undoubtedly necessary to consider combinations of graphs to explain a decision. Interestingly, DISCERN that builds on activation rules often provides better results in term of Indfidelity than **AR(node)**, achieving score that

are similar to the state of the art methods while having better fidelity and sparsity scores than these methods in most of the cases. Finally, the generated graph brings further interpretability on activation rules without altering too much the performance of the explainer directly built from these rules. All together, these results suggest that the activation rules allow identifying relevant representation space within the GNNs.

4.4. Comparison to model-level baselines

We now assess DISCERN against model-level explanation baselines. Notice that Random and DISC-GSPAN are not directed by the measures we introduce. On the contrary, XGNN++ optimizes either Cosine, Cross-Entropy or Relative-CE measures. We compare these baselines against DISCERN with a similar experimental protocol as in Section 4.2.3. For each activation rule, we study the L2 norm between the best graph generated by each method and the activation rule $\mathbf{A}^{\ell}$. We assume that each rule is represented by a vector whose values equal to 1 for components inside the rule, 0 otherwise. Results are reported in Table 5. Note that DISC-GSPAN fails for BA2 and Proteins because of the small number or the absence of labels on nodes, which makes extraction impossible. For all measures, DISCERN provides graphs that are better embedded within the target space than any other method. On average, DISCERN$_{Cos}$ outperforms the best solution based on either XGNN or DISC-GSPAN of about 12%.

Table 5. Average L2 norm between the graphs provided by each method aggregated over all activation rules. The lower the value, the better.

	DD	Aids	Mutagen	BBBP	Proteins	BA2
Random	4.18	4.12	4.63	4.34	5.32	6.59
DISC-GSPAN	3.52	3.62	3.90	3.47	•	•
XGNN++(Cosine)	4.19	4.04	4.61	4.26	5.39	5.58
XGNN++(Cross-Entropy)	4.09	3.97	4.46	4.15	5.23	6.57
XGNN++(Relative-CE)	4.16	4.01	4.50	4.22	5.15	6.56
DISCERN$_{Cosine}$	**3.11**	**3.18**	**3.47**	**3.15**	**4.42**	4.61
DISCERN$_{Cross-Entropy}$	3.16	3.47	3.76	3.45	4.54	4.64
DISCERN$_{\text{Relative-CE}}$	3.34	3.47	3.75	3.48	4.55	**4.59**

We study the importance of hyperparameter β (see section 3.3.2) for generating realistic graphs. To this end, we assess how realistic the generated graphs are compared to those from the related dataset. We compute the maximum common subgraph (MCSG) between the graph returned by DISCERN and each graph from the dataset. In Fig 5, we report the size of the MCSG normalized by the size of the generated graph when β varies. When this value reaches one, it means that the generated graph is a subgraph of a graph from the dataset. Therefore, the closer the value is to 1, the more realistic the generated graphs are. We observe that the graphs generated by DISCERN become more realistic when the hyperparameter β increases. Interestingly, Cross-Entropy and Relative-CE metrics allow obtaining a more realistic graph than Cosine metric for the same value of β. For information, the realism factor is also reported for the three baselines. By definition, DISC-GSPAN provides more realistic graphs since it mines frequent subgraphs. Hence, the realism factor is equal to 1. XGNN++ fails to generate realistic graphs whatever the metrics with a value ranging between 0.5 and 0.7 which still remains better than random.

4.4.1. Examples

We report in Figure 6 the best graphs provided by each method for two activation rules on Mutagen. These rules are highly correlated to the decision "Mutagenicity". For the first rule (top row), DISC-GSPAN and DISCERN identify parts of toxycophores (Bay-region, K-region) [13]. XGNN++ provides either unrealistic (i.e., Cosine) or too general graphs (i.e., only one carbon). Note that the graph generated by DISCERN for both Cross-Entropy and Relative-CE is not entirely realistic since a hydrogen atom cannot have two bonds. Nevertheless, duplicating this atom and binding it to another carbon (dashed node and edge) leads to a similar representation which is realistic.

For the second activation rule (bottom), both DISC-GSPAN and DISCERN$_{Cross-Entropy}$ depict a part of amine group (NH_2) while DISCERN$_{Relative-CE}$ outputs Ammonia. These molecules are known to be toxicophore. DISCERN$_{Cosine}$ generates a Vinylidene group also known to be toxic. It is interesting to note that the center nodes (red filled nodes)

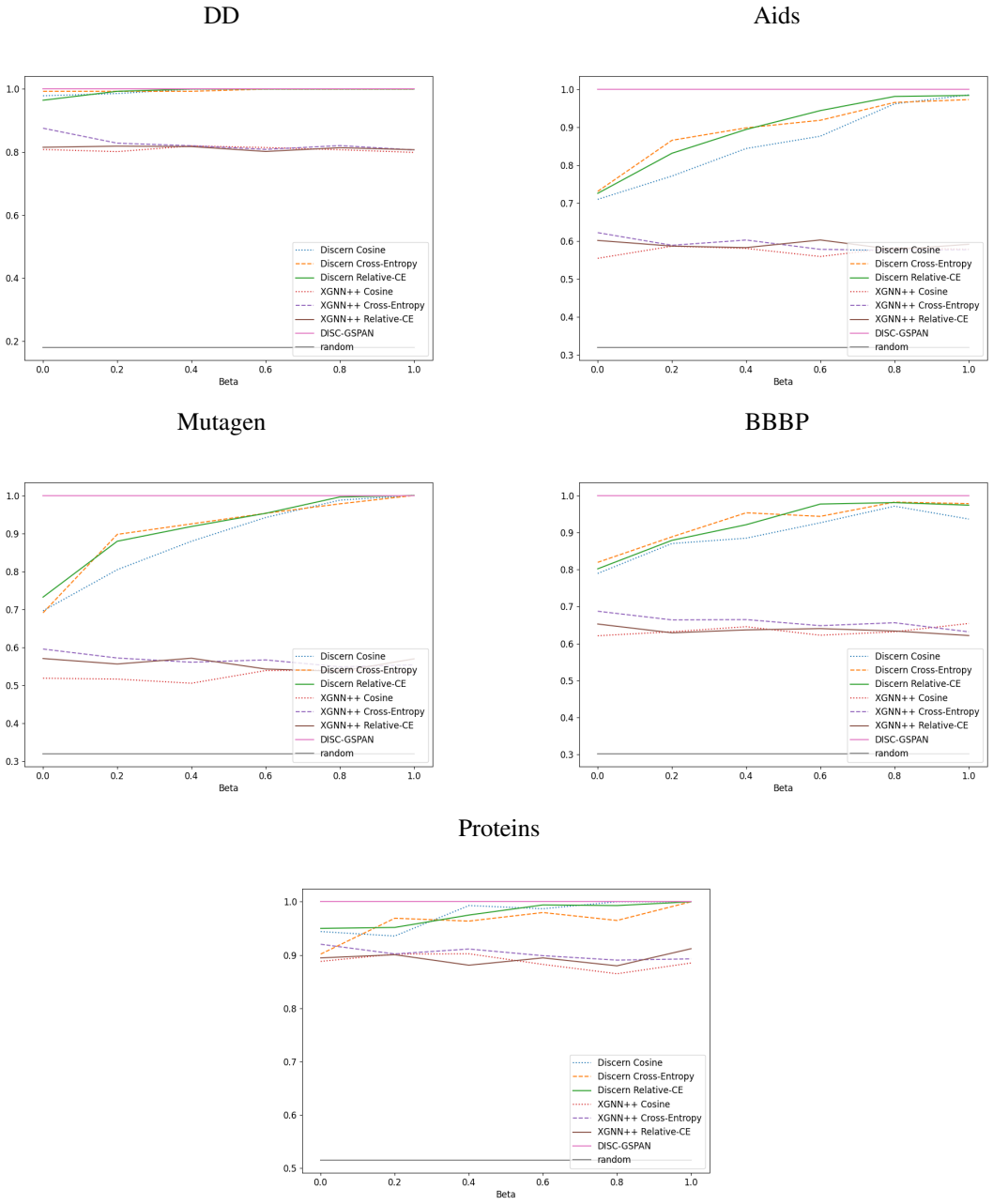

Figure 5. Normalized maximum common subgraph (MCSG) between DISCERN and generated graphs (y-axis) with respect to β for all datasets except BA2. The closer to 1, the more realistic the graph.

do not depict the same atom. Once again, graphs generated by XGNN++ are either unrealistic or too general (carbon atom). Graphs generated by Random are not shown as they are too unrealistic as shown in Figure 5.

4.5. Discussion

We state here the main conclusion we can draw from these experiments. Our method DISCERN makes it possible to generate representative graphs considering three metrics (i.e., Cross-Entropy, Cosine and Relative-CE). Experiments give evidence that these three metrics are well optimized through the MCTS-based generation. The quality of the

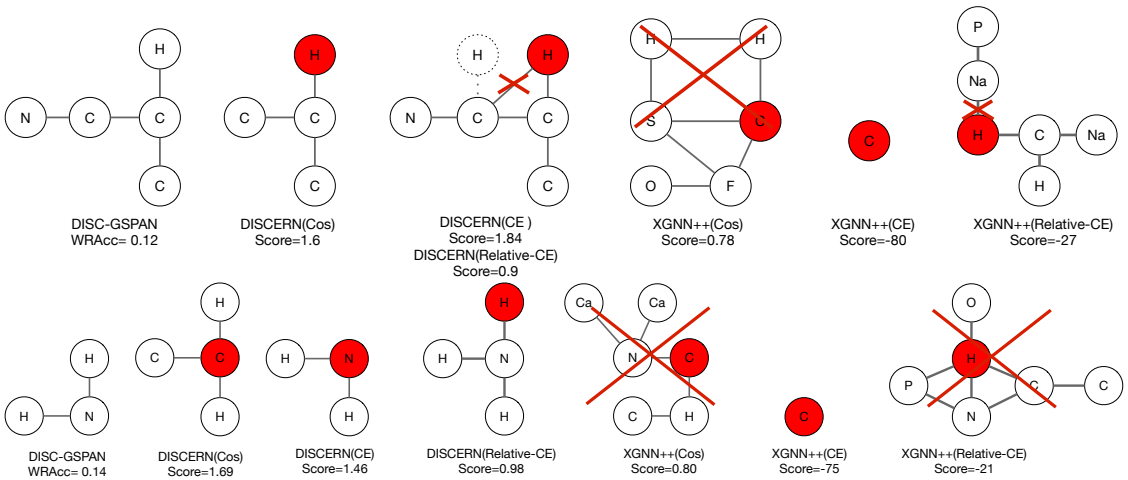

Figure 6. Graphs generated by each method for two activation rules (rows) that are highly correlated to mutagenicity. Red cross highlights unrealistic bonds or molecules. Red nodes are those that activate the rule.

representative graphs is statistically significant as reported in Table 2. Results suggest that Cross-Entropy is slightly better than two others but this is not significant. Experiments demonstrate that interest of activation rules. Building mask directly from activation rules allows us outperforming the state of the art instance-level explainers. Nevertheless, such rules are not interpretable. This motivates DISCERN whose instance explanation performance is comparable to competitors. The second series of experiments demonstrates that DISCERN outperforms the baselines by providing more realistic graphs. Nevertheless, we do not have theoretical guarantee of generating fully realistic graphs and DISCERN can generate graph with unrealistic configuration as shown in Figure 6.

5. Conclusion and future work

In this paper, we have tackled the problem of explaining GNNs with an original angle of attack. Instead of just assessing the importance of some input graph feature to the model decision, our goal is to study the GNN internal representation, i.e., to identify and highlight the features the GNN built through its different layers. To this end, we have introduced a novel method for explaining internal representations of GNNs. Given some activation rules that define internal representations having a strong impact on the classification process, DISCERN generates, with a MCTS approach, realistic graphs that fully embed in the related subspace identified by the rules. Our method relies on a proximity measure between a graph and an activation rule to assess how the generated graph embeds in the subspace defined by the activation rule. There are different ways to construct such a measure and we have proposed three different ones. We have reported an extensive empirical study on six real-world datasets. We have obtained comprehensive results proving that the activation rules allow identifying relevant representation spaces built by the GNN. Masks directly built from the activation rules allow obtaining a instance-level model explanation method that outperforms four state of the art methods while explanations directly based on the graph generated by DISCERN achieve performance comparable to state of the art methods. We have provided further evidence that DISCERN characterizes well each rule with realistic graphs. This makes it possible to capture interesting insights about how the internal representations are built by the GNN.

We believe that such method can support knowledge discovery from powerful GNNs and provide insights on object of study for scientists. Finally, a number of potential limitations need to be considered for future research to make this knowledge discovery from GNNs effective in practice. First, due to its intrinsic nature, the generated graphs pay attention to the graph structure. As a consequence, some findings may be over-specified especially in the case where the GNN builds only-content-related features. To overcome this limitation, we need to provide some additional assessment and/or to introduce a “wild-card label”. Second, the relations between layers are not taken into account in the discovery of activation rules. This may lead to redundant results when considering all the layers. To avoid such redundancy, it is necessary to take into account as prior knowledge the previous layers of a given layer.

The present study has only investigated the discovery of activation rules and then their characterization based on a representative generated graph. We believe that this allows to identify hidden features built by the GNN. However, the current study does not take into account how these features are combined to lead to the model decision. A next step toward interpretability is the investigation of how the model combine these features.

Acknowledgements

This work was supported by the ACADEMICS grant of the IDEXLYON project of the University of Lyon, PIA operated by ANR-16-IDEX-0005. It also benefited from the financial support CAF AMERICA OPE2020-0041.

References

[1] F. Baldassarre and H. Azizpour. Explainability for GCNs. *arXiv:1905.13686*, 2019.
[2] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H. Kriegel. Protein function prediction via graph kernels. In *ICISMB*, pages 47–56, 2005.
[3] N. Burkart and M. F. Huber. A survey on the explainability of supervised machine learning. *JAIR*, 70:245–317, 2021.
[4] T. De Bie. Finding interesting itemsets using a probabilistic model for binary databases. Technical report, University of Bristol, 2009.
[5] T. De Bie. An information theoretic framework for data mining. In *SIGKDD*, pages 564–572, 2011.
[6] J. Deng. *Mining subjectively interesting patterns in rich data*. PhD thesis, Ghent University, 2021.
[7] P. D. Dobson and A. J. Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Jour. Mol. Biol.*, 330(4):771–783, 2003.
[8] A. Duval and F. D. Malliaros. Graphsvx: Shapley value explanations for graph neural networks. In *ECMLPKDD'21*, pages 302–318, 2021.
[9] J. Fischer, A. Oláh, and J. Vreeken. What's in the box? exploring the inner life of neural networks with robust rules. In *ICML*, pages 3352–3362, 2021.
[10] T. Funke, M. Khosla, and A. Anand. Zorro: Valid, sparse, and stable explanations in graph neural networks. *CoRR*, abs/2105.08621, 2021.
[11] X. Gao, B. Xiao, D. Tao, and X. Li. A survey of graph edit distance. *Pattern Analysis and applications*, 13(1):113–129, 2010.
[12] Q. Huang, M. Yamada, Y. Tian, and et al. GraphLIME: Local Interpretable Model Explanations for Graph Neural Networks. *arXiv:2001.06216*, 2020.
[13] J. Kazius, R. McGuire, and R. Bursi. Derivation and validation of toxicophores for mutagenicity prediction. *Jour. med. chem.*, 48(1):312–320, 2005.
[14] T. Kipf and M. Welling. Semi-supervised classification with GCN. In *ICLR*, 2017.
[15] N. Lavrac, B. Kavsek, P. A. Flach, and L. Todorovski. Subgroup discovery with CN2-SD. *J. Mach. Learn. Res.*, 5:153–188, 2004.
[16] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang. Parameterized explainer for GNN. In *NeurIPS 2020*, 2020.
[17] C. Molnar. *Interpretable machine learning*. Lulu. com, 2020.
[18] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. TUDataset. *CoRR*, abs/2007.08663, 2020.
[19] C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2(11), 2017.
[20] H. Park and J. Neville. Exploiting interaction links for node classification with deep graph neural networks. In *IJCAI 2019*, pages 3223–3230, 2019.
[21] P. E. Pope, S. Kolouri, M. Rostami, C. E. Martin, and H. Hoffmann. Explainability methods for GCN. In *IEEE CVPR*, pages 10772–10781, 2019.
[22] M. T. Ribeiro, S. Singh, and C. Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *ACM SIGKDD*, pages 1135–1144, 2016.
[23] M. S. Schlichtkrull, N. D. Cao, and I. Titov. Interpreting graph neural networks for NLP with differentiable edge masking. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.
[24] T. Schnake, O. Eberle, J. Lederer, S. Nakajima, K. T. Schütt, K. Müller, and G. Montavon. XAI for graphs. *CoRR*, abs/2006.03589, 2020.
[25] T. Schnake, O. Eberle, J. Lederer, S. Nakajima, K. T. Schütt, K.-R. Müller, and G. Montavon. Higher-order explanations of graph neural networks via relevant walks. *arXiv preprint arXiv:2006.03589*, 2020.
[26] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks. In *ICLR 2014*, 2014.
[27] S. N. Tran and A. S. d'Avila Garcez. Deep logic networks: Inserting and extracting knowledge from deep belief networks. *IEEE TNNLS*, 29(2):246–258, 2018.
[28] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *ICLR 2018*, 2018.
[29] L. Veyrin-Forrer, A. Kamal, S. Duffner, M. Plantevit, and C. Robardet. On GNN explanability with activation patterns. *Data Min. Knowl. Discov.*, 2022.
[30] L. Veyrin-Forrer, A. Kamal, S. Duffner, M. Plantevit, and C. Robardet. Qu'est-ce que mon GNN capture vraiment ? exploration des représentations internes d'un GNN. In *Extraction et Gestion des Connaissances, EGC 2022, Blois, France, 24 au 28 janvier 2022*, pages 159–170, 2022.
[31] M. N. Vu and M. T. Thai. PGM-Explainer: Probabilistic Graphical Model Explanations for Graph Neural Networks . In *NeurIPS 2020*, 2020.
[32] X. Wang, Y. Wu, A. Zhang, F. Feng, X. He, and T.-S. Chua. Reinforced causal explainer for graph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2022.
[33] Z. Wang and S. Ji. Second-order pooling for graph neural networks. *IEEE TPAMI*, 2020.
[34] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on GNNs. *IEEE trans. on NN and learning systems*, 2020.

[35] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. S. Pande. Moleculenet. *CoRR*, abs/1703.00564, 2017.
[36] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are GNN? In *ICLR*, 2019.
[37] X. Yan and J. Han. GSPAN: graph-based substructure pattern mining. In *ICDM 2002*, pages 721–724. IEEE Computer Society, 2002.
[38] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec. GNNExplainer: Generating Explanations for Graph Neural Networks. In *NeurIPS 2019*, pages 9240–9251, 2019.
[39] H. Yuan, J. Tang, X. Hu, and S. Ji. XGNN: Towards Model-Level Explanations of Graph Neural Networks . In *KDD'20*, pages 430–438, 2020.
[40] H. Yuan, H. Yu, S. Gui, and S. Ji. Explainability in graph neural networks: A taxonomic survey. *arXiv:2012.15445*, 2020.
[41] M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An end-to-end deep learning architecture for graph classification. In *AAAI-2018*, pages 4438–4445, 2018.
[42] Y. Zhang, D. Defazio, and A. Ramesh. Relex: A model-agnostic relational model explainer. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, pages 1042–1049, 2021.