

GraphSVX: Shapley Value Explanations for Graph Neural Networks

Alexandre Duval and Fragkiskos D. Malliaros

Université Paris-Saclay, CentraleSupélec, Inria, Gif-Sur-Yvette, France
 alexandre.duval@centralesupelec.fr
 fragkiskos.malliaros@centralesupelec.fr

Abstract. Graph Neural Networks (GNNs) achieve significant performance for various learning tasks on geometric data due to the incorporation of graph structure into the learning of node representations, which renders their comprehension challenging. In this paper, we first propose a unified framework satisfied by most existing GNN explainers. Then, we introduce GraphSVX, a post hoc local model-agnostic explanation method specifically designed for GNNs. GraphSVX is a decomposition technique that captures the “fair” contribution of each feature and node towards the explained prediction by constructing a surrogate model on a perturbed dataset. It extends to graphs and ultimately provides as explanation the Shapley Values from game theory. Experiments on real-world and synthetic datasets demonstrate that GraphSVX achieves state-of-the-art performance compared to baseline models while presenting core theoretical and human-centric properties.

1 Introduction

Many aspects of the everyday life involve data without regular spatial structure, known as non-euclidean or geometric data, such as social networks, molecular structures or citation networks [1, 6, 11]. These datasets, often represented as graphs, are challenging to work with because they require modelling rich relational information on top of node feature information [47, 48]. Graph Neural Networks (GNNs) are powerful tools for representation learning of such data. They achieve state-of-the-art performance on a wide variety of tasks [10, 41, 46] due to their recursive message passing scheme, where they encode information from nodes and pass it along the edges of the graph. Similarly to traditional deep learning frameworks, GNNs showcase a complex functioning that is rather opaque to humans. As the field grows, understanding them becomes essential for well known reasons, such as ensuring privacy, fairness, efficiency, and safety [5, 16, 25].

While there exist a variety of explanation methods [12, 30, 32, 34, 35, 45], they are not well suited for geometric data as they fall short in their ability to incorporate graph topology information. Baldassarre and Azizpour [2] and Pope et al. [26] have proposed extensions to GNNs, but in addition to limited performance, they require internal knowledge of the model and show gradient saturation issues due to the discrete nature of the adjacency matrix.

GNNE explainer [42] is the first explanation method designed specifically for GNNs. It learns a continuous (and a discrete) mask over the edges (and features) of the graph by formulating an optimisation process that maximizes mutual information between the distribution of possible subgraphs and GNN prediction. More recently, PGExplainer [21] and GraphMask [31] generalize GNNE explainer to an inductive setting; they use re-parametrisation tricks to alleviate the “introduced evidence” problem [7]—continuous masks deform the adjacency matrix and introduce new semantics to the generated graph. Regarding other approaches; GraphLIME [15] builds on LIME [27] to provide a non-linear explanation model; PGM-Explainer [40] learns a simple Bayesian network handling node dependencies; XGNN [43] produces model-level insights via graph generation trained using reinforcement learning.

Despite recent progress, existing explanation methods do not relate much and show clear limitations. Apart from GNNE explainer, none considers node features together with graph structure in explanations. Besides, they do not present core properties of a “good” explainer [24] (see Sec. 2). Since the field is very recent and largely unexplored, there is little certified knowledge about explainers’ characteristics. It is, for instance, unclear whether optimising mutual information is pertinent or not. Overall, this often yields explanations with a poor signification, like a probability score stating how essential a variable is. Existing techniques not

only lack strong theoretical grounds, but also do not showcase an evaluation that is sophisticated enough to properly justify their effectiveness or other desirable aspects [29]. Lastly, little importance is granted to the comprehensibility of explanations from a human perspective, limiting their human-centric characteristics [23].

In light of these limitations, first, we propose a unified explanation framework encapsulating recently introduced explainers for GNNs. It not only serves as a connecting force between them but also provides a different and common view of their functioning, which should inspire future work. In this paper, we exploit it ourselves to define and endow our explainer, GraphSVX, with desirable properties. More precisely, GraphSVX carefully constructs and combines the key components of the unified pipeline so as to jointly capture the average marginal contribution of node features and graph nodes towards the explained prediction. We show that GraphSVX ultimately computes, via an efficient algorithm, the *Shapley values* from game theory [33], that we extend to graphs. The resulting unique explanation thus satisfy several theoretical properties by definition, while it is made more human-centric through several extensions. In the end, we evaluate GraphSVX on real-world and synthetic datasets for node and graph classification tasks. We show that it outperforms existing baselines in explanation accuracy, and verifies further desirable aspects such as robustness or certainty.

Source code. The source code is available at <https://github.com/AlexDuvalinho/GraphSVX>.

2 Related Work

Explanations methods specific to GNNs are classified into five categories of methods according to [44]: gradient-based, perturbation, decomposition, surrogate, and model-level. We utilise the same taxonomy in this paper to position GraphSVX.

Decomposition methods [2, 26] distribute the prediction score among input features using the weights of the network architecture, through backpropagation. Despite offering a nice interpretation, they are not specific to GNNs and present several major limits such as requiring access to model parameters, like **gradient-based methods** discussed in Sec. 1.

Perturbation methods [21, 31, 42] monitor variations in model prediction with respect to different input perturbations. Such methods provide as explanation a continuous mask over edges (and features) learned via a simple optimisation procedure, affected by the introduced-evidence problem.

Surrogate methods [15, 40] approximate the black box GNN model locally by learning an interpretable model on a dataset built around the instance of interest v (e.g., neighbours). Explanations for the surrogate model are used as explanations for the original model. For now, such approaches consider exclusively node features or graph topology, but not both.

Model level methods [43] provide general insights on the model functioning. However, it only supports graph classification, and some instance-level methods can be extended to provide global explanations [21].

As we will show shortly, GraphSVX bridges the gap between these categories by learning a surrogate explanation model on a perturbed dataset that ultimately decomposes the explained prediction among the nodes and features of the graph, depending on their respective contribution. It also derives model-level insights by explaining subsets of nodes, while avoiding the respective limits of each category.

Desirable properties of explanations have received subsequent attention from the social sciences and the machine learning communities, but are often overlooked when designing an explainer. From a theoretical perspective, good explanations are accurate, fidel (*truthful*), and reflect the proportional importance of a feature on prediction (*meaningful*) [4, 44]. They also are stable and consistent (*robust*), meaning with a low variance when changing to a similar model or a similar instance [24]. Besides, they reflect the certainty of the model (*decomposable*) and are as representative as possible of its (*global*) functioning [22]. Finally, since their ultimate goal is to help humans understand the model, explanations should be intuitive to comprehend (*human-centric*). Many sociological and psychological studies emphasise key aspects: only a few motives

(*selective*) [38], comparable to other instances (*contrastive*) [19], and interactive with the explaine (social). We refer to Appendix F for more rigorous definitions and to see how GNN explainers satisfy them.

3 Preliminary Concepts and Background

Notation. We consider a graph \mathcal{G} with N nodes and F features defined by (\mathbf{X}, \mathbf{A}) where $\mathbf{X} \in \mathbb{R}^{N \times F}$ is the feature matrix and $\mathbf{A} \in \mathbb{R}^{N \times N}$ the adjacency matrix. $f(\mathbf{X}, \mathbf{A})$ denotes the prediction of the GNN model f , and $f_v(\mathbf{X}, \mathbf{A})$ the score of the predicted class for node v . Let $\mathbf{X}_{*j} = (X_{1j}, \dots, X_{Nj})$ with feature values $\mathbf{x}_{*j} = (x_{1j}, \dots, x_{Nj})$ represent feature j 's value vector across all nodes. Similarly, $\mathbf{X}_i = \mathbf{X}_{i*} = (X_{i1}, \dots, X_{iF})$ stands for node i 's feature vector, with $\mathbf{X}_{iS} = \{X_{ik} | k \in S\}$. $\mathbf{1}$ is the all-ones vector.

3.1 Graph Neural Networks

GNNs adopt a message passing mechanism [14] where the update at each GNN layer ℓ involves three key calculations [3]: (i) The propagation step. The model computes a message $m_{ij}^\ell = \text{MSG}(\mathbf{h}_i^{\ell-1}, \mathbf{h}_j^{\ell-1}, a_{ij})$ between every pair of nodes (v_i, v_j) , that is, a function MSG of v_i 's and v_j 's representations $\mathbf{h}_i^{\ell-1}$ and $\mathbf{h}_j^{\ell-1}$ in the previous layer and of the relation a_{ij} between the nodes. (ii) The aggregation step. For each node v_i , GNN calculates an aggregated message M_i^ℓ from v_i 's neighbourhood \mathcal{N}_{v_i} , whose definition vary across methods. $M_i^\ell = \text{AGG}(m_{ij}^\ell | v_j \in \mathcal{N}_{v_i})$. (iii) The update step. GNN non-linearly transforms both the aggregated message M_i^ℓ and v_i 's representation $\mathbf{h}_i^{\ell-1}$ from the previous layer, to obtain v_i 's representation \mathbf{h}_i^ℓ at layer ℓ : $\mathbf{h}_i^\ell = \text{UPD}(M_i^\ell, \mathbf{h}_i^{\ell-1})$. The representation $\mathbf{z}_i = \mathbf{h}_i^L$ of the final GNN layer L serves as final node embedding and is used for downstream machine learning tasks.

3.2 The Shapley value

The Shapley value is a method from Game Theory. It describes how to fairly distribute the total gains of a game to the players depending on their respective contribution, assuming they all collaborate. It is obtained by computing the average marginal contribution of each player when added to any possible coalition of players [33]. This method has been extended to explain machine learning model predictions on tabular data [18, 36], assuming that each feature of the explained instance (\mathbf{x}) is a player in a game where the prediction is the payout.

The characteristic function $val : S \rightarrow \mathbb{R}$ captures the marginal contribution of the coalition $S \subseteq \{1, \dots, F\}$ of features towards the prediction $f(\mathbf{x})$ with respect to the average prediction: $val(S) = \mathbb{E}[f(\mathbf{X}) | \mathbf{X}_S = \mathbf{x}_s] - \mathbb{E}[f(\mathbf{X})]$. We isolate the effect of a feature j via $val(S \cup \{j\}) - val(S)$ and average it over all possible ordered coalitions S to obtain its Shapley value as:

$$\phi_j(val) = \sum_{S \subseteq \{1, \dots, F\} \setminus \{j\}} \frac{|S|! (F - |S| - 1)!}{F!} (val(S \cup \{j\}) - val(S)).$$

The notion of fairness is defined by four axioms (*efficiency*, *dummy*, *symmetry*, *additivity*), and the Shapley value is the unique solution satisfying them. In practice, the sum becomes impossible to compute because the number of possible coalitions (2^{F-1}) increases exponentially by adding more features. We thus approximate Shapley values using sampling [8, 20, 37].

4 A Unified Framework for GNN Explainers

As detailed in the previous section, existing interpretation methods for GNNs are categorised and often treated separately. In this paper, we approach the explanation problem from a new angle, proposing a unified view that regroups existing explainers under a single framework: GNNExplainer, PGExplainer, GraphLIME, PGM-Explainer, XGNN, and the proposed GraphSVX. The key differences across models lie in the definition and optimisation of the three main blocks of the pipeline, as shown in Fig. 1:

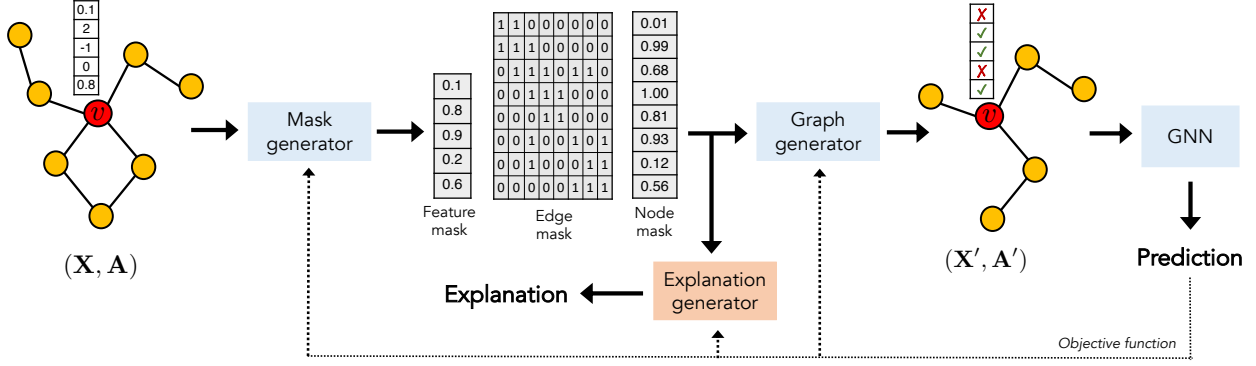


Fig. 1. Overview of unified framework. All methods take as input a given graph $\mathcal{G} = (\mathbf{X}, \mathbf{A})$, feed it to a mask generator (MASK) to create three masks over nodes, edges and features. These masks are then passed to a graph generator (GEN) that converts them to the original input space $(\mathbf{X}', \mathbf{A}')$ before feeding them to the original GNN model f . The resulting prediction $f(\mathbf{X}', \mathbf{A}')$ is used to improve the mask generator, the graph generator or the downstream explanation generator (EXPL), which ultimately provides the desired explanation—using masks and $f(\mathbf{X}', \mathbf{A}')$. This passage through the framework is repeated many times to create a proper dataset \mathcal{D} from which each generator block learns. Usually, only one is optimised with a carefully defined process involving the new and original GNN predictions.

- MASK generates discrete or continuous masks over features $\mathbf{M}_F \in \mathbb{R}^F$, nodes $\mathbf{M}_N \in \mathbb{R}^N$ and edges $\mathbf{M}_E \in \mathbb{R}^{N \times N}$, according to a specific strategy.
- GEN outputs a new graph $\mathcal{G}' = (\mathbf{X}', \mathbf{A}')$ from the masks $(\mathbf{M}_E, \mathbf{M}_N, \mathbf{M}_F)$ and the original graph $\mathcal{G} = (\mathbf{X}, \mathbf{A})$.
- EXPL generates explanations via a surrogate model g (its parameters). The input, target, loss function and interpretable domain of g vary across baselines.

In the following, we show how each baseline fits the pipeline. \odot stands for the element wise multiplication operation, σ the softmax function, \parallel the concatenation operation, and \mathbf{M}^{ext} describes the extended vector \mathbf{M} with repeated entries, whose size makes the operation feasible. All three masks are not considered for a single method; some are ignored as they have no effect on final explanations. Indeed, one often studies node feature \mathbf{M}_F or graph structure (via \mathbf{M}_E or \mathbf{M}_N).

GNNEExplainer’s key component is the mask generator. It generates both \mathbf{M}_F and \mathbf{M}_E , where \mathbf{M}_E has continuous values and \mathbf{M}_F discrete ones. They are both randomly initialised and jointly optimised via a mutual information loss function $MI(Y, (\mathbf{M}_E, \mathbf{M}_F)) = H(Y) - H(Y|\mathbf{A}', \mathbf{X}')$, where GEN gives $\mathbf{A}' = \mathbf{A} \odot \sigma(\mathbf{M}_E)$ and $\mathbf{X}' = \mathbf{X} \odot \mathbf{M}_F^{\text{ext}}$. Y represents the class label. EXPL is unimportant here, it simply outputs the learned masks as explanations, for instance via the coefficients of a linear function $g(\mathbf{1}) = (\mathbf{M}_E, \mathbf{M}_F)$.

PGExplainer is very similar to GNNEExplainer. MASK generates only an edge mask \mathbf{M}_E using a multi-layer neural network MLP_ψ and the learned matrix \mathbf{Z} of node representations: $\mathbf{M}_E = \text{MLP}_\psi(\mathcal{G}, \mathbf{Z})$. The new graph is constructed with $\text{GEN}(\mathbf{X}, \mathbf{A}, \mathbf{M}_E) = (\mathbf{X}, \mathbf{A} \odot \rho(\mathbf{M}_E))$, where ρ denotes a reparametrisation trick. The obtained prediction $f_v(\mathbf{X}, \mathbf{A}')$ is also used to maximise mutual information with $f_v(\mathbf{X}, \mathbf{A})$ and backpropagates the result to optimise MASK. As for GNNEExplainer, EXPL provides \mathbf{M}_E as explanations.

GraphLIME is a surrogate method with a simple and not optimised mask generator. Although it measures feature importance, it creates a node mask \mathbf{M}_N using the neighbourhood of v (i.e., \mathcal{N}_v). The k^{th} mask (or sample) is defined as $\mathbf{M}_{N,i}^k = 1$ if $v_i = \mathcal{N}_v[k]$ and 0 otherwise. $\text{GEN}(\mathbf{X}, \mathbf{A}, \mathbf{M}_N) = (\mathbf{X}, \mathbf{A})$, so in fact, it computes and stores the original model prediction. \mathbf{X} and $f(\mathbf{X}, \mathbf{A})$ are then combined with the mask \mathbf{M}_N via simple dot products $\mathbf{M}_N^\top \cdot \mathbf{X}$ and $\mathbf{M}_N^\top \cdot f(\mathbf{X}, \mathbf{A})$ respectively, to isolate the original feature vector and prediction of the k^{th} neighbour of v . These two elements are treated as input and target of an HSIC Lasso model. An adapted loss function is used to train the model.

PGM-Explainer builds a probabilistic graphical model on a local dataset that consists of random node masks $\mathbf{M}_N \in \{0, 1\}^N$. The associated prediction $f_v(\mathbf{X}', \mathbf{A}')$ is obtained by posing $\mathbf{A}' = \mathbf{A}$ and $\mathbf{X}' = \mathbf{M}_N^{\text{ext}} \odot \mathbf{X} + (\mathbf{1} - \mathbf{M}_N^{\text{ext}} \odot \boldsymbol{\mu}^{\text{ext}})$, with $\boldsymbol{\mu} = (E[\mathbf{X}_{*1}], \dots, E[\mathbf{X}_{*F}])^\top$. This means that each excluded node feature ($\mathbf{M}_{N,j} = 0$) is set to its mean value across all nodes. This dataset is fed sequentially to the main component EXPL – a Bayesian Network with input \mathbf{M}_N (made sparser by looking at the Markov-blanket of v), *BIC score* loss function, and target $I(f_v(\mathbf{X}', \mathbf{A}'))$, where $I(\cdot)$ is a specific function that quantifies the difference in prediction between original and new prediction.

XGNN trains a graph generator via Reinforcement Learning. It adds an edge to the existing graph at each iteration and uses the new graph as input of the next one. The final graph generated is given as explanation. More formally, we let \mathbf{M}_N masks all non-existing nodes in \mathcal{G} , and $\mathbf{M}_E = \mathbf{A}$. The graph generator outputs the two endpoints of the new added edge, at each iteration. It uses a candidate set \mathcal{C} of vertices of \mathcal{G} , a GCN model $\hat{\mathbf{X}} = \text{GCN}(\mathcal{C}, \mathbf{X}, \mathbf{A})$, and an MLP network $\sigma(\text{MLP}_1(\hat{\mathbf{X}})) \cdot \mathbf{M}_N$ to derive the first endpoint of the new edge. Another MLP is then applied to obtain the other endpoint, using the starting node as $\sigma(\text{MLP}_2(\hat{\mathbf{X}}, \hat{\mathbf{X}}_{\text{start}}))$. This edge is added to the existing graph to produce \mathcal{G}' , which is fed to the GNN. The prediction is backpropagated to update GEN via a policy gradient loss function. \mathcal{G}' becomes the new input of the next iteration and is provided as final explanation after T iterations. In fact, EXPL returns \mathbf{M}_E .

GraphSVX. As we will see in the next section, the proposed GraphSVX model carefully exploits the potential of this framework through a better design and combination of complex mask, graph and explanation generators—in the perspective of improving performance and embedding desirable properties in explanations.

5 Proposed Method

GraphSVX is a post hoc model-agnostic explanation method specifically designed for GNNs, that jointly computes graph structure and node feature explanations for a single instance. More precisely, GraphSVX constructs a perturbed dataset made of binary masks for nodes and features ($\mathbf{M}_N, \mathbf{M}_F$), and computes their marginal contribution $f(\mathbf{X}', \mathbf{A}')$ towards the prediction, using a graph generator $\text{GEN}(\mathbf{X}, \mathbf{A}, \mathbf{M}_F, \mathbf{M}_N)$ to obtain $(\mathbf{X}', \mathbf{A}')$. It then learns a carefully defined explanation model on the dataset $(\mathbf{M}_N || \mathbf{M}_F, f(\mathbf{X}', \mathbf{A}'))$ and provides it as explanation. Ultimately, it produces a unique deterministic explanation that decomposes the original prediction and has a real signification (Shapley values) as well as other desirable properties evoked in Sec. 2. Without loss of generality, we consider a node classification task for the presentation of the method.

5.1 Mask and graph generators

First of all, we create an efficient mask generator algorithm that constructs discrete feature and node masks, respectively denoted by $\mathbf{M}_F \in \{0, 1\}^F$ and $\mathbf{M}_N \in \{0, 1\}^N$. Intuitively, for the explained instance v , we aim at studying the joint influence of a subset of features and neighbours of v towards the associated prediction $f_v(\mathbf{X}, \mathbf{A})$. The mask generator helps us determine the subset being studied. Associating 1 with a variable (node or feature) means that it is considered, 0 that it is discarded. For now, we let MASK randomly sample from all possible (2^{F+N-1}) pairs of masks \mathbf{M}_F and \mathbf{M}_N , meaning all possible coalitions S of features and nodes (v is not considered in explanations). Let \mathbf{z} be the random variable accounting for selected variables, $\mathbf{z} = (\mathbf{M}_F || \mathbf{M}_N)$. This is a simplified version of the true mask generator, which we will come back to later, in Sec. 5.4.

We now would like to estimate the joint effect of this group of variables towards the original prediction. We thus isolate the effect of selected variables marginalised over excluded ones, and observe the change in prediction. We define $\text{GEN} : (\mathbf{X}, \mathbf{A}, \mathbf{M}_F, \mathbf{M}_N) \rightarrow (\mathbf{X}', \mathbf{A}')$, which converts the obtained masks to the original input space, in this perspective. Due to the message passing scheme of GNNs, studying jointly node and features' influence is tricky. Unlike GNNExplainer, we avoid any overlapping effect by considering feature values of v (instead of the whole subgraph around v) and all nodes except v . Several options are possible to cancel out a node's influence on the prediction, such as replacing its feature vector by random or expected values. Here, we decide to isolate the node in the graph, which totally removes its effect on the prediction.

Similarly, to neutralise the effect of a feature, as GNNs do not handle missing values, we set it to the dataset expected value. Formally, it translates into:

$$\mathbf{X}' = \mathbf{X} \quad \text{with} \quad \mathbf{X}'_v = \mathbf{M}_F \odot \mathbf{X}_v + (\mathbf{1} - \mathbf{M}_F) \odot \boldsymbol{\mu} \quad (1)$$

$$\mathbf{A}' = (\mathbf{M}_N^{\text{ext}\top} \cdot \mathbf{A} \cdot \mathbf{M}_N^{\text{ext}}) \odot I(\mathbf{A}), \quad (2)$$

where $\boldsymbol{\mu} = (\mathbb{E}[\mathbf{X}_{*1}], \dots, \mathbb{E}[\mathbf{X}_{*F}])^\top$ and $I(\cdot)$ captures the indirect effect of k -hop neighbours of v ($k > 1$), which is often underestimated. Indeed, if a 3-hop neighbour w is considered alone in a coalition, it becomes disconnected from v in the new graph \mathcal{G}' . This prevents us from capturing its indirect impact on the prediction since it does not pass information to v anymore. To remedy this problem, we select one shortest path \mathcal{P} connecting w to v via *Dijkstra's* algorithm, and include \mathcal{P} back in the new graph. To keep the influence of the new nodes (in $\mathcal{P} \setminus \{w, v\}$) switched off, we set their feature vector to mean values obtained by Monte Carlo sampling. The pseudocode is in the Supplementary material.

To finalize the perturbation dataset, we pass $\mathbf{z}' = (\mathbf{X}', \mathbf{A}')$ to the GNN model f and store each sample $(\mathbf{z}, f(\mathbf{z}'))$ in a dataset \mathcal{D} . \mathcal{D} associates with a subset of nodes and features of v their estimated influence on the original prediction.

5.2 Explanation generator

In this section, we build a surrogate model g on the dataset $\mathcal{D} = \{(\mathbf{z}, f(\mathbf{z}'))\}$ and provide it as explanation. More rigorously, an explanation ϕ of f is normally drawn from a set of possible explanations, called interpretable domain Ω . It is the solution of the following optimisation process: $\phi = \arg \min_{g \in \Omega} \mathcal{L}_f(g)$, where the loss function attributes a score to each explanation. The choice of Ω has a large impact on the type and quality of the obtained explanation. In this paper, we choose broadly Ω to be the set of interpretable models, and more precisely the set of Weighted Linear Regression (WLR).

In short, we intend our model to learn to calculate the individual effect of each variable towards the original prediction from the joint effect $f(\mathbf{z}')$, using many different coalitions S of nodes and features. This is made possible by the definition of the input dataset \mathcal{D} and is enforced by a cross entropy loss function:

$$\mathcal{L}_{f,\pi}(g) = \sum_{\mathbf{z}} (g(\mathbf{z}) - f(\mathbf{z}'))^2 \pi_{\mathbf{z}}, \quad (3)$$

where $\pi_{\mathbf{z}} = \frac{F + N - 1}{(F + N) \cdot |\mathbf{z}|} \cdot \binom{F + N - 1}{|\mathbf{z}|}^{-1}.$

π is a kernel weight that attributes a high weight to samples \mathbf{z} with small or large dimension, or in different terms, groups of features and nodes with few or many elements—since it is easier to capture individual effects from the combined effect in these cases.

In the end, we provide the learned parameters of g as explanation. Each coefficient corresponds to a node of the graph or a feature of v and represents its estimated influence on the prediction $f_v(\mathbf{X}, \mathbf{A})$. In fact, it approximates the extension of the Shapley value to graphs, as shown in next paragraph.

5.3 Decomposition model

We first justify why it is relevant to extend the Shapley value to graphs. Looking back at the original theory, each player contributing to the total gain is allocated a proportion of that gain depending on its fair contribution. Since a GNN model prediction is fully determined by node feature information (\mathbf{X}) and graph structural information (\mathbf{A}), both edges/nodes and node features are players that should be considered in explanations. In practice, we extend to graphs the four Axioms defining fairness (please see the Supplementary Material), and redefine how is captured the influence of players (features and nodes) towards the prediction as $val(S) = \mathbb{E}_{\mathbf{X}_v}[f_v(\mathbf{X}, \mathbf{A}_S) | \mathbf{X}_{vS} = \mathbf{x}_{vS}] - \mathbb{E}[f_v(\mathbf{X}, \mathbf{A})]$. \mathbf{A}_S is the adjacency matrix where all nodes in \bar{S} (not in S) have been isolated.

Assuming model linearity and feature independence, we show that GraphSVX, in fact, captures via $f(\mathbf{z}')$ the marginal contribution of each coalition S towards the prediction:

$$\begin{aligned}\mathbb{E}_{\mathbf{X}_v}[f_v(\mathbf{X}, \mathbf{A}_S)|\mathbf{X}_{vS}] &= \mathbb{E}_{\mathbf{X}_{v\bar{S}}|\mathbf{X}_{vS}}[f_v(\mathbf{X}, \mathbf{A}_S)] \\ &\approx \mathbb{E}_{\mathbf{X}_{v\bar{S}}}[f_v(\mathbf{X}, \mathbf{A}_S)] && \text{by independence} \\ &\approx f_v(\mathbb{E}_{\mathbf{X}_{v\bar{S}}}[\mathbf{X}], \mathbf{A}_S) && \text{by linearity} \\ &= f_v(\mathbf{X}', \mathbf{A}'),\end{aligned}$$

where $\mathbf{A}' = \mathbf{A}_S$ and $\mathbf{X}'_{ij} = \begin{cases} \mathbb{E}[\mathbf{X}_{*j}] & \text{if } i = v \text{ and } j \in \bar{S} \\ \mathbf{X}_{ij} & \text{otherwise.} \end{cases}$

Using the above, we prove that GraphSVX calculates the Shapley values on graph data. This builds on the fact that Shapley values can be expressed as an additive feature attribution model, as shown by [20] in the case of tabular data.

In this perspective, we set π such that $\pi_{\mathbf{z}} \rightarrow \infty$ when $|\mathbf{z}| \in \{0, F + N\}$ to enforce the *efficiency* axiom: $g(\mathbf{1}) = f_v(\mathbf{X}, \mathbf{A}) = \mathbb{E}[f_v(\mathbf{X}, \mathbf{A})] + \sum_{i=1}^{F+N} \phi_i$. This holds due to the specific definition of GEN and g (i.e., EXPL), where $g(\mathbf{1}) = f_v(\mathbf{X}, \mathbf{A})$ and the constant ϕ_0 , also called base value, equals $\mathbb{E}_{\mathbf{X}_v}[f_v(\mathbf{X}, \mathbf{A}_v)] \approx \mathbb{E}[f_v(\mathbf{X}, \mathbf{A})]$, so the mean model prediction. \mathbf{A}_v refers to \mathbf{A}_\emptyset , where v is isolated.

Theorem 1. *With the above specifications and assumptions, the solution to $\min_{g \in \Omega} \mathcal{L}_{f, \pi}(g)$ under Eq. (3) is a unique explanation model g whose parameters compute the extension of the Shapley values to graphs.*

Proof. Please see Appendix D.

5.4 Efficient approximation specific to GNNs

Similarly to the euclidean case, the exact computation of the Shapley values becomes intractable due to the number of possible coalitions required. Especially that we consider jointly features and nodes, which augments exponentially the complexity of the problem. To remedy this, we derive an efficient approximation via a smart mask generator.

Firstly, we reduce the number of nodes and features initially considered to $D \leq N$ and $B \leq F$ respectively, without impacting performance. Indeed, for a GNN model with k layers, only k -hop neighbours of v can influence the prediction for v , and thus receive a non-zero Shapley value. All others are allocated a null importance according to the *dummy* axiom¹ and can therefore be discarded. Similarly, any feature j of v whose value is comprised in the confidence interval $I_j = [\mu_j - \lambda \cdot \sigma_j, \mu_j + \lambda \cdot \sigma_j]$ around the mean value μ_j can be discarded, where σ_j is the corresponding standard deviation and λ a constant.

The complexity is now $\mathcal{O}(2^{B+D})$ and we further drive it down to $\mathcal{O}(2^B + 2^D)$ by sampling separately masks of nodes and features, while still considering them jointly in g . In other words, instead of studying the influence of possible combinations of nodes and features, we consider all combinations of features with no nodes selected, and all combinations of nodes with all features included: $(2^B + 2^D)$. We observe empirically that it achieves identical explanations with fewer samples, while it seems to be more intuitive to capture the effect of nodes and features on prediction (expressed by Axiom 1).

Axiom 1 (Relative efficiency) *Node contribution to predictions can be separated from feature contribution, and their sum decomposes the prediction with respect to the average one:*
$$\begin{cases} \sum_{j=1}^B \phi_j = f_v(\mathbf{X}, \mathbf{A}_v) - \mathbb{E}[f_v(\mathbf{X}, \mathbf{A})] \\ \sum_{i=1}^D \phi_{B+i} = f_v(\mathbf{X}, \mathbf{A}) - f_v(\mathbf{X}, \mathbf{A}_v). \end{cases}$$

Lastly, we approximate explanations using $P \ll 2^B + 2^D$ samples, where P is sufficient to obtain a good approximation. We reduce P by greatly improving MASK, as evoked in Sec. 5.1. Assuming we have a budget of P samples, we develop a smart space allocation algorithm to draw in priority coalitions of order k , where k starts at 0 and is incremented when all coalitions of the order are sampled. This means that we sample in priority coalitions with high weight, so with nearly all or very few players. If they cannot all be chosen (for current k) due to space constraints, we proceed to a smart sampling that favours unseen players. The pseudocode and an evaluation of its efficiency lie in Appendix C and E respectively.

¹ Axiom: If $\forall S \in \mathcal{P}(\{1, \dots, p\})$ and $j \notin S$, $val(S \cup \{j\}) = val(S)$, then $\phi_j(val) = 0$.

5.5 Desirable properties of explanations

In the end, GraphSVX generates fairly distributed explanations $\sum_j \phi_j = f_v(\mathbf{X}, \mathbf{A})$, where each ϕ_j approximates the average marginal contribution of a node or feature j towards the explained GNN prediction (with respect to the average prediction ϕ_0). By definition, the resulting explanation is unique, consistent, and stable. It is also truthful and robust to noise, as shown in Sec. 6.2. The last focus of this paper is to make them more selective, global, contrastive and social; as we aim to design an explainer with desirable properties. A few aspects are detailed here.

Contrastive. Explanations are contrastive already as they yield the contribution of a variable with respect to the average prediction $\phi_0 = \mathbb{E}[f(\mathbf{X}, \mathbf{A})]$. To go further and explain an instance with respect to another one, we could substitute \mathbf{X}_v in Eq. (1) by $\mathbf{X}'_v = \mathbf{M}_F \odot \mathbf{X}_v + (\mathbf{1} - \mathbf{M}_F) \odot \boldsymbol{\xi}$, with $\boldsymbol{\xi}$ being the feature vector of a specific node w , or of a fictive representative instance from class C .

Global. We derive explanations for a subset T of nodes instead of a single node v , following the same pipeline. The neighbourhood changes to $\bigcup_i^T \mathcal{N}_i$, Eq. (1) now updates \mathbf{X}_T instead of \mathbf{X}_v and $f(\mathbf{z}')$ is calculated as the average prediction score for nodes in T . Also, towards a more global understanding, we can output the global importance of each feature j on v 's prediction by enforcing in Eq. (1) $\mathbf{X}_{\mathcal{N}_v \cup \{v\}, j}$ to a mean value obtained by Monte Carlo sampling on the dataset, when $\mathbf{z}_j = 0$. This holds when we discard node importance, otherwise the overlapping effects between nodes and features render the process obsolete.

Graph classification. Until now, we had focused on node classification but the exact same principle applies for graph classification. We simply look at $f(\mathbf{X}, \mathbf{A}) \in \mathbb{R}$ instead of $f_v(\mathbf{X}, \mathbf{A})$, derive explanations for all nodes or all features (not both) by considering features across the whole dataset instead of features of v , like our global extension.

6 Experimental Evaluation

In this section, we present several experiments designed to determine the quality of our explanation method using synthetic and real world datasets, on both node and graph classification tasks. We first study the effectiveness of GraphSVX in presence of ground truth explanations. We then consider more complex real-world datasets to show how our explainer generalises to them. Since we do not have ground truth, we test GraphSVX's ability to filter noisy features and noisy nodes from explanations. Detailed dataset statistics, hyper-parameter tuning, properties' check and further experimental results including ablation study, are provided in Appendix E.

6.1 Synthetic and real datasets with ground truth

Synthetic node classification task. We follow the same setting as [21] and [42], where four kinds of datasets are constructed. Each input graph is a combination of a base graph together with a set of motifs, where both differ across datasets. The label of each node is determined based on its belonging and role in the motif. As a consequence, the explanation for a node in a motif should be the nodes in the same motif, which creates ground truth explanation. This ground truth can be used to measure the performance of an explainer via an accuracy metric.

Synthetic and real-world graph classification task. With a similar evaluation perspective, we measure the effectiveness of our explainer on graph classification, also using ground truth. We use a synthetic dataset *BA-2motifs* that resembles the previous ones, and a real life dataset called *MUTAG*. It consists of 4,337 molecule graphs, each assigned to one of 2 classes based on its mutagenic effect [28]. As discussed in [9], carbon rings with groups NH_2 or NO_2 are known to be mutagenic, and could therefore be used as ground truth.

Baselines. We compare the performance of GraphSVX to the main instance-level explanation baselines that incorporate graph structure in explanations: GNNExplainer, PGExplainer and PGM-Explainer. GraphLIME

	Node Classification				Graph Classification	
	BA-Shapes	BA-Community	Tree-Cycles	Tree-Grid	BA-2motifs	MUTAG
Base						
Motifs						
Features	None	$\mathcal{N}(\mu_l, \sigma_l)$	None	None	None	Atom types
Visualization						
Explanations by GraphSVX						
Explanation AUC						
GNExplainer	0.83	0.75	0.86	0.84	0.68	0.65
PGM-Explainer	0.96	0.92	0.95	0.87	0.91	0.72
PGExplainer	0.92	0.81	0.96	0.88	0.85	0.79
GraphSVX	0.99	0.93	0.97	0.93	0.99	0.77

Table 1. Illustration of different datasets together with performance evaluation of GraphSVX and other GNN explainers baselines.

and XGNN are not applicable here because they do not provide graph structure explanations for such tasks.

Experimental setup and metrics. We train the same 3-layer GNN model on every dataset. The settings are described in Appendix E. The performance is measured with an accuracy metric (node accuracy or edge accuracy depending on the nature of explanations) on top- k explanations, where k is equal to the ground truth dimension. More precisely, we formalise the evaluation as a binary classification of nodes (or edges) where nodes (or edges) inside motifs are positive, and the rest negative.

Results. The results on both synthetic and real-life datasets are summarized in Table 1. As shown in the table, GraphSVX correctly identifies essential graph structure, outperforming the leading baselines on all but one task, in addition to offering higher theoretical guarantees and human-friendly explanations. On *MUTAG*, the special nature of the dataset and the ground truth favours edge explanation methods, which capture slightly more information than node explainers. Hence, PGExplainer was expected to perform better. For *BA-Community*, GraphSVX demonstrates its ability to identify relevant features and nodes together, as it also identifies important node features with 100% accuracy. In terms of efficiency, our explainer is slower than the very scalable PGExplainer despite our efficient approximation, but is often comparable to GNExplainer. Running time experiments as well as existence of desirable properties (certainty, stability, consistency, comprehensibility, etc.) are given in Appendix E and F.

6.2 Real-world datasets without ground truth

In this paragraph, we evaluate GraphSVX on two real-world datasets without ground truth explanations: *Cora* and *PubMed*. We design experiments where we add noisy features and noisy nodes to the data, train a GNN model, and observe if an explainer includes them in explanations. In different terms, we investigate if the explainer can filter useless features/nodes and select relevant information pieces in explanations.

Datasets. *Cora* is a citation graph where nodes represent articles and edges represent citations between pairs of papers. The task involved is document classification where the goal is to categorise each paper into one out of seven categories. Each feature indicates the absence/presence of the corresponding term in its abstract. *PubMed* is also a publication dataset with three classes and 500 features, each indicating the TF-IDF value of the corresponding word.

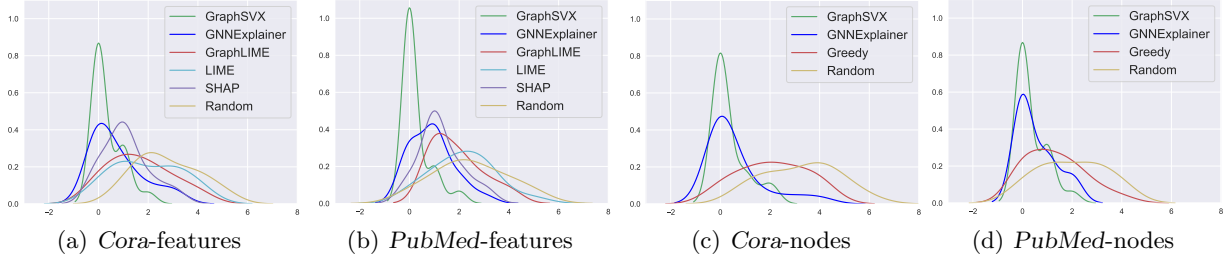


Fig. 2. Frequency distributions of noisy features (a), (b) and nodes (c), (d) using a GAT model on *Cora* and *PubMed*.

Noisy features. Concretely, we artificially added 20% of new “noisy” features to the dataset. We define these new features using existing ones’ distribution. We re-trained a GCN and a GAT model on this noisy data, whose test accuracy was above 75%. The detailed experimental settings are provided in Appendix E. We then produced explanations for 50 test samples using different explainer baselines, on *Cora* and *PubMed*, and we compared their performance by assessing how many noisy features are included in explanations among top- k features. Ultimately, we compare the resulting frequency distributions using a kernel density estimator (KDE). Intuitively, since features are noisy, they are not used by the GNN model, and thus are unimportant. Therefore, the less noisy features are included in the explanation, the better the explainer.

Baselines include GNNExplainer, GraphLIME (described previously) as well as the well-known SHAP [20] and LIME [27] models. We also compare GraphSVX to a method based on a Greedy procedure, which greedily removes the most contributory features/nodes of the prediction until the prediction changes, and the Random procedure, which randomly selects k features/nodes as the explanations for the prediction being explained.

The results are depicted in Fig. 2 (a)-(b). For all GNNs and on all datasets, the number of noisy features selected by GraphSVX is close to zero, and in general lower than existing baselines—demonstrating its robustness to noise.

Noisy nodes. We follow a similar idea for noisy neighbours instead of noisy features. Each new node’s connectivity and feature vector are determined using the dataset’s distribution. Only a few baselines (GNNExplainer, Greedy, Random) among the ones selected previously can be included for this task since GraphLIME, SHAP, and LIME do not provide explanations for nodes.

As before, this evaluation builds on the assumption that a well-performing model will not consider as essential these noisy nodes. We check the validity of this assumption for the GAT model by looking at its attention weights. We retrieve the average attention weight of each node across the different GAT layers and compare the one attributed to noisy nodes versus normal nodes. We expect it to be lower for noisy nodes, which proves to be true: 0.12 vs. 0.15.

As shown in Fig. 2 (c)-(d), GraphSVX also outperforms all baselines, showing nearly no noisy nodes in explanations. Nevertheless, GNNExplainer achieves almost as good performance on both datasets (and in many different evaluation configurations). Random performs slightly better than Greedy, which could be a bit surprising at the first sight. But this difference disappears when we increase the number of most important nodes we look at.

7 Conclusion

In this paper, we have first introduced a unified framework for explaining GNNs, showing how various explainers could be expressed as instances of it. We then use this complete view to define GraphSVX, which conscientiously exploits the above pipeline to output explanations for graph topology and node features endowed with desirable theoretical and human-centric properties, eligible of a good explainer. We achieve this by defining a decomposition method that builds an explanation model on a perturbed dataset, ultimately computing the Shapley values from game theory, that we extended to graphs. Through a comprehensive evaluation, we not only achieve state-of-the-art performance on various graph and node classification tasks but also demonstrate the desirable properties of GraphSVX.

Acknowledgements. Supported in part by ANR (French National Research Agency) under the JCJC project GraphIA (ANR-20-CE23-0009-01).

References

1. Backstrom, L., Leskovec, J.: Supervised random walks: predicting and recommending links in social networks. In: WSDM (2011)
2. Baldassarre, F., Azizpour, H.: Explainability techniques for graph convolutional networks. arXiv (2019)
3. Battaglia, P.W., Hamrick, J.B., Bapst, V., Sanchez-Gonzalez, A., et al.: Relational inductive biases, deep learning, and graph networks. arXiv (2018)
4. Burkart, N., Huber, M.F.: A survey on the explainability of supervised machine learning. JAIR **70**, 245–317 (2021)
5. Caruana, R., Lou, Y., Gehrke, J., Koch, P., Sturm, M., Elhadad, N.: Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In: KDD (2015)
6. Cho, E., Myers, S.A., Leskovec, J.: Friendship and mobility: user movement in location-based social networks. In: KDD (2011)
7. Dabkowski, P., Gal, Y.: Real time image saliency for black box classifiers. In: NeurIPS (2017)
8. Datta, A., Sen, S., Zick, Y.: Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. In: IEEE symposium on security and privacy (SP) (2016)
9. Debnath, K., et al.: Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. Journal of medicinal chemistry **34**(2), 786–797 (1991)
10. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: NeurIPS (2016)
11. Duvenaud, D., et al.: Convolutional networks on graphs for learning molecular fingerprints. In: NeurIPS (2015)
12. Goldstein, A., Kapelner, A., Bleich, J., Pitkin, E.: Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. JCGS **24**(1), 44–65 (2015)
13. Goodman, B., Flaxman, S.: Eu regulations on algorithmic decision-making and a “right to explanation”. In: ICML workshop on Human Interpretability in Machine Learning (2016)
14. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: NeurIPS (2017)
15. Huang, Q., Yamada, M., Tian, Y., Singh, D., Yin, D., Chang, Y.: GraphLIME: Local interpretable model explanations for graph neural networks. arXiv (2020)
16. Kim, B.: Interactive and interpretable machine learning models for human machine collaboration. Ph.D. thesis, Massachusetts Institute of Technology (2015)
17. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
18. Lipovetsky, S., Conklin, M.: Analysis of regression in game theory approach. ASMBI **17**(4), 319–330 (2001)
19. Lipton, P.: Contrastive explanation. Royal Institute of Philosophy Supplement **27**, 247–266 (1990)
20. Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. In: NeurIPS (2017)
21. Luo, D., Cheng, W., Xu, D., Yu, W., Zong, B., Chen, H., Zhang, X.: Parameterized explainer for graph neural network. In: NeurIPS (2020)
22. Miller, T.: Explanation in Artificial Intelligence: Insights from the social sciences. Artificial intelligence **267**, 1–38 (2019)
23. Miller, T., Howe, P., Sonenberg, L.: Explainable AI: Beware of inmates running the asylum or: How I learnt to stop worrying and love the social and behavioural sciences. arXiv (2017)
24. Molnar, C.: Interpretable Machine Learning. Lulu. com (2020)
25. O’neil, C.: Weapons of math destruction: How big data increases inequality and threatens democracy. Broadway Books (2016)
26. Pope, P.E., Kolouri, S., Rostami, M., Martin, C.E., Hoffmann, H.: Explainability methods for graph convolutional neural networks. In: CVPR (2019)
27. Ribeiro, M.T., Singh, S., Guestrin, C.: “Why should I trust you?” Explaining the predictions of any classifier. In: KDD (2016)
28. Riesen, K., Bunke, H.: Iam graph database repository for graph based pattern recognition and machine learning. In: Joint IAPR - SPR & SSPR. Springer (2008)
29. Robnik-Šikonja, M., Bohanec, M.: Perturbation-based explanations of prediction models. In: Human and machine learning, pp. 159–175. Springer (2018)
30. Saltelli, A.: Sensitivity analysis for importance assessment. Risk analysis **22**(3), 579–590 (2002)
31. Schlichtkrull, M.S., Cao, N.D., Titov, I.: Interpreting graph neural networks for NLP with differentiable edge masking. In: ICLR (2021)

32. Selvaraju, R., et al.: Grad-cam: Visual explanations from deep networks via gradient-based localization. In: ICCV (2017)
33. Shapley, L.S.: A value for n-person games. *Contributions to the Theory of Games* **2**(28), 307–317 (1953)
34. Shrikumar, A., Greenside, P., Kundaje, A.: Learning important features through propagating activation differences. In: ICML (2017)
35. Simonyan, K., Vedaldi, A., Zisserman, A.: Deep inside convolutional networks: Visualising image classification models and saliency maps. arXiv preprint arXiv:1312.6034 (2013)
36. Strumbelj, E., Kononenko, I.: An efficient explanation of individual classifications using game theory. *JMLR* **11**, 1–18 (2010)
37. Štrumbelj, E., Kononenko, I.: Explaining prediction models and individual predictions with feature contributions. *KAIS* **41**(3), 647–665 (2014)
38. Ustun, B., Rudin, C.: Methods and models for interpretable linear classification. arXiv (2014)
39. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: ICLR (2018)
40. Vu, M.N., Thai, M.T.: PGM-Explainer: Probabilistic graphical model explanations for graph neural networks. In: NeurIPS (2020)
41. Xu, D., Cheng, W., Luo, D., Liu, X., Zhang, X.: Spatio-temporal attentive rnn for node classification in temporal attributed graphs. In: IJCAI. pp. 3947–3953 (2019)
42. Ying, Z., Bourgeois, D., You, J., Zitnik, M., Leskovec, J.: GNNExplainer: Generating explanations for graph neural networks. In: NeurIPS (2019)
43. Yuan, H., Tang, J., Hu, X., Ji, S.: XGNN: Towards model-level explanations of graph neural networks. In: KDD (2020)
44. Yuan, H., Yu, H., Gui, S., Ji, S.: Explainability in graph neural networks: A taxonomic survey. arXiv preprint arXiv:2012.15445 (2020)
45. Zhang, J., Bargal, S.A., Lin, Z., Brandt, J., Shen, X., Sclaroff, S.: Top-down neural attention by excitation backprop. *IJCV* **126**(10), 1084–1102 (2018)
46. Zhang, M., Chen, Y.: Link prediction based on graph neural networks. In: NeurIPS (2018)
47. Zhang, Z., Cui, P., Zhu, W.: Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering* (2020)
48. Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M.: Graph neural networks: A review of methods and applications. arXiv (2018)

A Graph Neural Networks

In this section, we define the two main GNNs used during the evaluation phase. Consider a graph \mathcal{G} with feature matrix \mathbf{X} , adjacency matrix \mathbf{A} , and diagonal degree matrix \mathbf{D} . Let N be the number of nodes, C the number of input features, F of output features, and \mathbf{Z} the output. Note that, $\mathbf{Z} \in \mathbb{R}^{N \times F}$, $\mathbf{X} \in \mathbb{R}^{N \times C}$, $\mathbf{W} \in \mathbb{R}^{C \times F}$, where \mathbf{W} is a weight matrix.

A.1 Graph Convolution Networks (GCN)

The output of one GCN [17] layer is obtained as follows:

$$\mathbf{H}^{(l+1)} = f(\mathbf{H}^{(l)}, \mathbf{A}) = \sigma(\tilde{\mathbf{D}}^{-1/2} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-1/2} \mathbf{H}^{(l)} \mathbf{W}^{(l)}),$$

with $\mathbf{H}^0 = \mathbf{X}$, $\mathbf{H}^L = \mathbf{Z}$ and σ often chosen to be the ReLU function. Also, $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\tilde{\mathbf{D}} = \mathbf{D} + \mathbf{I}$ (adding self loops). Each element in $\mathbf{H}^{(l+1)}$ can thus be written as $\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}_{v_i}} \frac{1}{c_{ij}} \mathbf{W}^{(l)} \mathbf{h}_j^{(l)}\right)$.

A.2 Graph Attention Networks (GAT)

The GAT [39] model is extremely similar to the GCN model. The key difference lies in how information is aggregated from one-hop neighbour:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}_{v_i}} \alpha_{ij}^{(l)} \mathbf{W}^{(l)} \mathbf{h}_j^{(l)}\right),$$

$$\text{with } \alpha_{ij} = \frac{\exp(\text{LeakyReLU}(\tilde{a}^\top [\mathbf{W}\mathbf{h}_i || \mathbf{W}\mathbf{h}_j]))}{\sum_{k \in \mathcal{N}_{v_i}} \exp(\text{LeakyReLU}(\tilde{a}^\top [\mathbf{W}\mathbf{h}_i || \mathbf{W}\mathbf{h}_k]))}.$$

In the above, \tilde{a} is single feed forward neural network that concatenates both inputs and produces a unique value.

We can have multi-head attention where the various heads utilised are either concatenated or averaged as follows:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\frac{1}{K} \sum_{k=1}^K \sum_{j \in \mathcal{N}_{v_i}} \alpha_{ij}^{(l)} \mathbf{W}_k^{(l)} \mathbf{h}_j^{(l)}\right) \quad \text{or} \quad \mathbf{h}_i^{(l+1)} = ||_{k=1}^K \sigma\left(\sum_{j \in \mathcal{N}_{v_i}} \alpha_{ij}^{(l)} \mathbf{W}_k^{(l)} \mathbf{h}_j^{(l)}\right).$$

B Extended Shapley Value Axioms

Here, we provide the extension of the Shapley Value Axioms [33] to graphs. Players designate nodes of the graph (except v) and features of v ; they are $F + N - 1$. \mathcal{P} is the power set, meaning all possible combinations of players.

Axiom 2 (Efficiency) *Features and nodes' contributions must add up to the difference between the original prediction and the average model prediction, i.e., $\sum_{j=1}^{F+N-1} \phi_j = f_v(\mathbf{X}, \mathbf{A}) - \mathbb{E}[f_v(\mathbf{X}, \mathbf{A})]$.*

Axiom 3 (Symmetry) *If $\forall S \in \mathcal{P}$, and $k, j \notin S$, $\text{val}(S \cup \{k\}) = \text{val}(S \cup \{j\})$, then $\phi_k(\text{val}) = \phi_j(\text{val})$. The contribution of two players j and k should be identical if they contribute equally to all possible coalitions of nodes and features.*

Axiom 4 (Dummy) *If $\forall S \in \mathcal{P}$, and $j \notin S$, $\text{val}(S \cup \{j\}) = \text{val}(S)$, then $\phi_j(\text{val}) = 0$. A player j which does not influence the predicted value regardless of the coalition sampled should receive a Shapley Value of 0.*

Axiom 5 (Additivity) *For all pairs of characteristic functions $v, w : \phi_j(v+w) = \phi_j(v) + \phi_j(w)$, where $(v+w)(S) = v(S) + w(S)$ for all $S \in \mathcal{P}$. For each instance, the sum of the Shapley Values for two different prediction tasks is equal to the Shapley Value of one task whose prediction is defined from the sum of the two instances' predictions. This last axiom constrains the value to be consistent in the space of all prediction tasks.*

C GraphSVX

In this section, we provide the pseudocode of some algorithms evoked in the paper: GraphSVX, the mask generator MASK and the indirect effect of nodes on prediction.

Algorithm 1 GraphSVX

```

1: Input:  $\mathcal{G} = (\mathbf{X}, \mathbf{A})$  with  $N$  nodes and  $F$  features. GNN model  $f$ . Interpretable domain  $\Omega$ , loss function  $\mathcal{L}$  and
   kernel weight  $\pi$  as defined in Eq.(3).  $P$ : number of samples. GEN, MASK, EXPL are detailed in the paper;
2:  $\mathcal{D} \leftarrow \{\}$  ;
3:  $\mathbf{Z} \leftarrow \text{MASK}(\mathbf{X}, \mathbf{A})$  ; // create a dataset of binary masks (feat and nodes)
4: for  $(\mathbf{M}_F, \mathbf{M}_N)$  in  $\mathbf{Z}$ : do
5:    $\mathbf{z} \leftarrow (\mathbf{M}_F || \mathbf{M}_N)$  ; // concatenation
6:    $(\mathbf{X}', \mathbf{A}') \leftarrow \text{GEN}(\mathbf{M}_F, \mathbf{M}_N, \mathbf{X}, \mathbf{A})$  ; // convert masks to the original input space
7:    $(\mathbf{X}', \mathbf{A}') \leftarrow \text{IE}(\mathbf{X}', \mathbf{A}', \mathbf{X}, \mathbf{A})$  ; // see Algorithm 2
8:    $\mathbf{y} \leftarrow f(\mathbf{X}', \mathbf{A}')$  ; // new GNN prediction
9:    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{z}, \mathbf{y})\}$  ; // add sample to dataset
10: end for
11:  $g \leftarrow \text{EXPL}(\mathcal{D}, \mathcal{L}_f, \pi, \Omega)$  ; // learn explanation model  $g$ 
12: Return parameters  $\phi$  of  $g$ 

```

Algorithm 2 IE - Indirect effect

```

1: Input: Explained node  $v$ ,  $\mathcal{N}_v^k$  set of  $k$ -hop neighbours of  $v$ , binary masks  $\mathbf{z}$ ,  $\mathcal{G} = (\mathbf{X}, \mathbf{A})$  and  $\mathcal{G}' = (\mathbf{X}', \mathbf{A}')$ ;
2:  $\mathcal{G}'_S \leftarrow \text{Subgraph}_{\mathcal{G}'}(\mathcal{N}_v^k)$  ; // k-hop subgraph of  $v$  in  $\mathcal{G}'$ 
3: for  $w$  in  $\{u \in \mathcal{N}_v^k | z_u = 1\}$  do
4:   if DISCONNECTED( $w, v$ ) in  $\mathcal{G}'_S$  then
5:      $\mathcal{P}_{all} \leftarrow \text{DIJKISTRAG}(v, w)$  ; // finds shortest paths in  $\mathcal{G}$  between ( $v, w$ )
6:      $\mathcal{P} \leftarrow \text{RANDOMSAMPLE}(\mathcal{P}_{all}, 1)$  ; // sample one path at random
7:     for  $e$  in  $\mathcal{P}_{edges}$  do
8:        $\mathbf{A}'_e = 1$  ; // include in  $\mathcal{G}'$  edges of the selected path
9:     end for
10:    for  $n$  in  $\mathcal{P}_{nodes} \setminus \{v, w\}$  do
11:       $\mathbf{X}'_{n*} = \mathbf{X}_{v*}$  ; // set new nodes of  $\mathcal{G}'$  to feature values of  $v$ 
12:    end for
13:  end if
14: end for
15: Return  $(\mathbf{X}', \mathbf{A}')$ 

```

While the first two pseudocodes are well-detailed within the paper and easy to understand, we will give some more intuition regarding the mask generator, split into two pseudocodes. The first one, MASK, constructs the dataset of masks, separating samples where we consider nodes from those where we consider features, following from the *relative efficiency* axiom. In practice, it samples P' coalitions of features (\mathbf{M}_F) according to our sampling algorithm SMARTERSEPARATE (Algorithm 4) and computes the associated kernel weights of this “incomplete” sample. This means that high weight is given to coalitions having all features but zero nodes, which was not the case beforehand. We then set to $\mathbf{0}$ the corresponding node masks \mathbf{M}_N to form final samples $\mathbf{z} = (\mathbf{M}_F, \mathbf{M}_N)$, because we capture better the influence of v ’s features on prediction when v is isolated. Note that P' is determined based on the proportion of nodes among all variables (nodes and features) considered. We repeat the same process for nodes, but this time, we include all features ($\mathbf{M}_F = \mathbf{1}$) in the final sample $\mathbf{z} = (\mathbf{M}_F, \mathbf{M}_N)$ as we would like to study the influence of neighbours of v when v is defined as it is (with all its features). Note the sum of features and nodes coefficients are fixed by the close to infinity weights enforced on the null coalition, the full coalition and the coalition of all features but zero nodes. We ultimately generate the dataset \mathbf{Z} of binary concatenated masks of nodes and features, as well as their respective weights. The global process is detailed in Algorithm 1 (GraphSVX).

Algorithm 3 MASK - Smart sampling for nodes and features

```

1: Input: number of samples  $P$  (budget),  $F$  features and  $N$  neighbours,  $M = F + N$ ,  $r$  regularisation parameter for
   node/feature importance, SMARTERSEPARATE, kernel weight function  $\pi_v$ , dataset  $\mathbf{Z} \in \mathbb{R}^{P \times (F+N)}$  of samples  $\mathbf{z}$ ;
2:  $P' = \begin{cases} \text{int}(r * P) & \text{if } r \text{ exists} \\ \text{int}(P \times F/M) & \text{if } F = 0 \text{ or } D = 0 ; \\ \text{int}(0.5 * P/2 + 0.5 * P * F/M) & \text{else} \end{cases} \quad // \text{ samples dedicated to features}$ 
3:  $\pi \leftarrow \{\}$  ;
4:  $\mathbf{Z}[P', F:] \leftarrow \mathbf{0}$ ,  $\mathbf{Z}[P', :, F] \leftarrow \mathbf{1}$  ;
5:  $\mathbf{Z}_f \leftarrow \text{SMARTERSEPARATE}(\mathbf{X}, \mathbf{A}, P', F)$  ; // see Algorithm 4
6: Shuffle rows of  $\mathbf{Z}_f$  ;
7:  $\pi \leftarrow \pi \cup \pi_v(\mathbf{Z}_f)$  ; // compute weights for each sample in  $\mathbf{Z}_f$ 
8:  $\mathbf{Z}[P', :, F] \leftarrow \mathbf{Z}_f$  ;
9:  $\mathbf{Z}_n \leftarrow \text{SMARTERSEPARATE}(\mathbf{X}, \mathbf{A}, P - P', N)$  ; // repeat process for nodes
10: Shuffle rows of  $\mathbf{Z}_n$  ;
11:  $\pi \leftarrow \pi \cup \pi_v(\mathbf{Z}_n)$  ;
12:  $\mathbf{Z}[P', F:] \leftarrow \mathbf{Z}_n$  ;
13: Return  $(\pi, \mathbf{Z})$ 

```

Algorithm 4 SMARTERSEPARATE - Smarter Mask generator

```

1: Input: Number of samples  $P$ ,  $M$  variables, maximum size  $S$  of samples favoured, graph  $\mathcal{G} = (\mathbf{X}, \mathbf{A})$ ;
2:  $\mathbf{Z} \leftarrow \{\}$  ;
3:  $P' \leftarrow \frac{9}{10} \times P$  ;
4:  $i = 0, k = 0$  ;
5: while  $i < P'$  and  $k \leq \min(S, M - 1)$  do
6:   if  $i + 2 \times \binom{M}{k} < P'$  then
7:      $\mathcal{C} \leftarrow \{\text{All\_coal\_of\_order\_}k\}$  ; // set of all subsets of order k
8:     for  $c$  in  $\mathcal{C}$  do
9:        $\mathbf{v} \leftarrow \mathbf{1}, \mathbf{u} \leftarrow \mathbf{0}$  ;
10:       $\mathbf{v}[c] \leftarrow \mathbf{0}, \mathbf{u}[c] \leftarrow \mathbf{1}$  ; // change value of subset's elements only
11:       $\mathbf{Z} \leftarrow \mathbf{Z} \cup \{\mathbf{u}, \mathbf{v}\}$  ; // add two samples to dataset
12:    end for
13:     $k += 1, i += \binom{M}{k}$  ;
14:   else
15:      $\mathbf{w} \in \mathbb{R}^M \leftarrow \mathbf{1}$  ; // init weights
16:      $\mathcal{C} \leftarrow \{\text{All\_coal\_of\_order\_}k\}$  ;
17:     while  $i < P'$  do
18:        $\mathbf{cw} \leftarrow \text{COMPUTEWEIGHTS}(\mathcal{C}, \mathbf{w})$  ; // def subset weight as sum of its elements' w
19:        $\mathcal{C}_{max} \leftarrow \text{MAXWEIGHTCOAL}(\mathcal{C}, \mathbf{cw})$  ; // keep only coalitions with max weight
20:        $c \leftarrow \text{RANDOMSAMPLE}(\mathcal{C}_{max})$  ; // randomly sample one coalition
21:        $p \leftarrow \text{BERNOULLI}(0.5)$  ; // p=0 or 1
22:        $\mathbf{v} \leftarrow [p.\text{repeat}(M)]$  ; // vector of dim M with repeated value p
23:        $\mathbf{v}[c] \leftarrow [(1 - p).\text{repeat}(\text{len}(c))]$  ; // replace value of elements in c by 1-p
24:        $\mathbf{Z} \leftarrow \mathbf{Z} \cup c$  ;
25:        $\mathbf{w}[c] \leftarrow (1 + w^{-1})^{-1}$  ; // update weights of coalition's elements
26:        $i += 1$  ;
27:     end while
28:   end if
29: end while
30:  $\mathbf{Z} \leftarrow \mathbf{Z} \cup \text{RANDOMCOALITIONS}(M, P - P')$  ; // sample (P-P') random binary vectors of size M
   using a Bernoulli(0.5) distribution, and add them to the dataset
31: Return  $\mathbf{Z}$ 

```

SMARTERSEPARATE (Algorithm 4), is a sampling algorithm that favours coalitions with high weight (small or large dimension) using smart space allocation. 10% of the space is left for random coalitions. The rest is divided as follows. If there is enough space, it samples all possible combinations of variables (from 1 to M) of order k , where k starts at 0 and is incremented once they are all sampled. We create and add to the dataset two samples from each subset, one where all variables are included (1) except those in the subset (0), the other where all variables are excluded (0) except those in the subset (1). We repeat the process until there is not enough space left to sample all subsets of order k . In this case, we select iteratively samples of order k (**1** with k zeros, **0** with k ones) until we reach the desired number, weighting the probability of each subset being drawn as the sum of its components' weights. The latter is defined such that variables that have already been sampled receive lower weights and therefore get a lower probability of being sampled again. This fosters diversity. Note that this is the most complex mask generator. There are some more simple ones that you can choose from in the code, and which are mentioned in the evaluation part below.

D Proof of Theorem 1

In this section, we prove that GraphSVX computes the extended Shapley Values to graphs via the coefficients ϕ of the explanation model g as defined in the paper. Note that, we take some ideas from [20] in this proof.

Let $M = F + N$ be the total number of players (nodes and features). $\mathbf{Z} \in \mathbb{R}^{2^M \times M}$ is the matrix whose rows represent the random variables $\mathbf{z} = (\mathbf{M}_F || \mathbf{M}_N)$, that is, all possible coalitions of features of v and graph nodes (but v). Let S the set of indices $j \in \{1, \dots, M\}$ where $\mathbf{z}_j = 1$ and $s = |S|$. $\mathbf{W} \in \mathbb{R}^{2^M \times 2^M}$ is a diagonal matrix containing the kernel weights $k(M, s) = \pi_{\mathbf{z}} = \frac{M-1}{M \cdot s} \cdot \binom{M-1}{s}^{-1}$, and $y_i = f_v(\mathbf{z}^{(i)})$ represents the output of the GNN model f for a new instance $\mathbf{z}^{(i)}$ —obtained via $\text{GEN}(\mathbf{z}^{(i)})$. Since g is a Weighted Linear Regression, the known formula to estimate its parameters is:

$$\phi = (\mathbf{Z}^\top \mathbf{W} \mathbf{Z})^{-1} \mathbf{Z}^\top \mathbf{W} \mathbf{y}.$$

In the above expression, the term $\mathbf{Z}^\top \mathbf{W}$ is an $M \times 2^M$ matrix whose each entry (i, j) contains the sample weight $\pi_{\mathbf{z}^{(i)}}$ if the corresponding player is present in this coalition $\mathbf{z}_j^{(i)} = 1$, and 0 otherwise. $\mathbf{Z}^\top \mathbf{W} \mathbf{Z}$ has a more tricky interpretation. An entry (j, j) in the diagonal contains the sum of all sample weights where player j is present, and the $(i, j)^{th} = (j, i)^{th}$'s entry corresponds to the sum of sample weights where both player i and player j are present in the coalition, i.e. $z_i = z_j = 1$.

Remember we must have $k(M, 0) = k(M, M) = \infty$, so \mathbf{W} is infinity for the zero row of \mathbf{Z} and its row of ones. However, if we set these infinite weights to a large positive constant c , (with \mathbf{I} being the identity matrix and \mathbf{J} the matrix of ones), then

$$(\mathbf{Z}^\top \mathbf{W} \mathbf{Z}) = \frac{M-1}{M} \mathbf{I} + c \mathbf{J}. \quad (4)$$

Computing the expression of $\mathbf{Z}^\top \mathbf{W} \mathbf{Z}$, the right-hand side of the sum in the equation follows easily, as the constant weight c attributed to the complete coalition (set of 1s) is added to each entry of the matrix $\mathbf{Z}^\top \mathbf{W} \mathbf{Z}$. Let's thus study how the first component is obtained. Denoting by E_j the set of samples where $z_j = 1$, it comes back to showing that the difference between the $(j, j)^{th}$ entry, which is in fact the sum of the weights given to each sample in E_j , and the $(j, i)^{th}$ entry, which denotes the sum of the weights of samples in $E_j \cap E_i$, equals $\frac{M-1}{M}$. Because we are looking at the identity matrix, we can choose any i, j such that $i \neq j$. We thus want to prove that each diagonal entry is equal to $\frac{M-1}{M}$, and we proceed as follows.

The number of $\mathbf{z} \in E_j \setminus E_i$ is $\binom{M-2}{s-1}$ as we need to place $(s-1)$ 1s into $M-2$ free spots, since in this set, $\mathbf{z}_j = 1$ and $\mathbf{z}_i = 0$ by definition. It follows that $1 \leq s \leq M-1$. Thus,

$$\begin{aligned} \sum_{\mathbf{z} \in E_j \setminus E_i} \pi_{\mathbf{z}} &= \sum_{s=1}^{M-1} \binom{M-2}{s-1} \frac{M-1}{M \cdot s} \cdot \binom{M-1}{s}^{-1} \\ &= \sum_{s=1}^{M-1} \frac{\binom{M-2}{s-1} (M-1) (s-1)! (M-s-1)!}{M!} \\ &= \sum_{s=1}^{M-1} \frac{(M-1) \binom{M-2}{s-1}}{\binom{M-2}{s-1} (M-1) M} \\ &= \sum_{s=1}^{M-1} \frac{1}{M} = \frac{M-1}{M}. \end{aligned}$$

We are, in fact, interested in the inverse of such matrix (when c tends to infinity):

$$(\mathbf{Z}^\top \mathbf{W} \mathbf{Z})^{-1} = \mathbf{I} + \frac{1}{M-1} (\mathbf{I} - \mathbf{J}).$$

To show this, we use the specific form of $(\mathbf{Z}^\top \mathbf{W} \mathbf{Z})$ given in Eq. (4) and $(\mathbf{Z}^\top \mathbf{W} \mathbf{Z})^{-1} (\mathbf{Z}^\top \mathbf{W} \mathbf{Z}) = \mathbf{I}$ to derive the expression of its inverse. In the equation below, we write $\mathbf{J}^2 = M \cdot \mathbf{J}$ because \mathbf{J} is the $M \times M$ matrix of all ones:

$$\begin{aligned} (a\mathbf{I} + c\mathbf{J})(b\mathbf{I} + d\mathbf{J}) &= ab\mathbf{I} + ad\mathbf{J} + bc\mathbf{J} + cd\mathbf{J}^2 \\ &= ab\mathbf{I} + \mathbf{J}(ad + bc) + Mcd\mathbf{J} \\ &= ab\mathbf{I} + \mathbf{J}(ad + bc + Mcd). \end{aligned}$$

This equals \mathbf{I} if $b = \frac{1}{a}$ and $ad + bc + Mcd = 0$, so $d = -\frac{c}{a(Mc + a)}$, meaning that $(b\mathbf{I} + d\mathbf{J})$ is the inverse of $(a\mathbf{I} + c\mathbf{J})$. Using the above, we have that

$$\begin{aligned} (\mathbf{Z}^\top \mathbf{W} \mathbf{Z})^{-1} &= (a\mathbf{I} + c\mathbf{J})^{-1} = \frac{1}{a} \mathbf{I} - \frac{c}{a(Mc + a)} \mathbf{J} \\ &= \frac{1}{a} \mathbf{I} - \frac{1}{a(M + \frac{a}{c})} \mathbf{J} \\ &\approx \frac{1}{a} \mathbf{I} - \frac{1}{aM} \mathbf{J} && \text{as } c \rightarrow \infty \\ &= \frac{M}{M-1} \left(\mathbf{I} - \frac{\mathbf{J}}{M} \right) && \text{as } a = \frac{M-1}{M} \\ &= \frac{M}{M-1} \mathbf{I} - \frac{\mathbf{J}}{M-1} \\ &= \mathbf{I} + \frac{1}{M-1} (\mathbf{I} - \mathbf{J}). \end{aligned}$$

Coming back again to our expression of the weighted least squares, multiplying $(\mathbf{Z}^\top \mathbf{W} \mathbf{Z})^{-1}$ by $\mathbf{Z}^\top \mathbf{W}$ creates a matrix of weights to apply to \mathbf{y} . Here, we consider without loss of generality the Shapley value for a single feature j (any feature), written ϕ_j and thus only need to consider a single row of this $M \times 2^M$ matrix. This is equivalent to using only the j^{th} row of $(\mathbf{Z}^\top \mathbf{W} \mathbf{Z})^{-1}$:

$$\phi_j = \sum_{i=1}^{2^M} (\mathbf{Z}^\top \mathbf{W} \mathbf{Z})_{(j,*)}^{-1} \mathbf{Z}^\top \mathbf{W}_{(*,i)} y_i.$$

We shall first place our attention on a single component $(\mathbf{Z}^\top \mathbf{W} \mathbf{Z})_{(j,*)}^{-1} (\mathbf{Z}^\top \mathbf{W})_{(*,i)}$ of the sum for now (for any i) and use the formula of $(\mathbf{Z}^\top \mathbf{W} \mathbf{Z})^{-1}$ derived above to rewrite it. Letting $\mathbf{1}_{Z_{i,j}=1} = 1$ if $Z_{i,j} = 1$ (feature j of coalition i is included) and 0 otherwise, this yields:

$$\begin{aligned} (\mathbf{Z}^\top \mathbf{W} \mathbf{Z})_{(j,*)}^{-1} (\mathbf{Z}^\top \mathbf{W})_{(*,i)} &= \left[\mathbf{1}_{Z_{i,j}=1} - \frac{(s_i - \mathbf{1}_{Z_{i,j}=1})}{M-1} \right] k(M, s_i) \\ &= \frac{M-1}{M \cdot s_i} \cdot \binom{M-1}{s_i}^{-1} \mathbf{1}_{Z_{i,j}=1} - (M \cdot s_i)^{-1} \cdot \binom{M-1}{s_i}^{-1} (s_i - \mathbf{1}_{Z_{i,j}=1}) \\ &= \frac{(M-1)(M-s_i-1)!(s_i-1)!}{M!} \mathbf{1}_{Z_{i,j}=1} - \frac{(s_i - \mathbf{1}_{Z_{i,j}=1})(M-s_i-1)(s_i-1)!}{M!} \\ &= \frac{(M-s_i-1)!(s_i-1)!}{M!} [(M-1)\mathbf{1}_{Z_{i,j}=1} - (s_i - \mathbf{1}_{Z_{i,j}=1})] \end{aligned}$$

When $\mathbf{1}_{Z_{i,j}=1} = 0$ we get

$$\frac{(M-s_i-1)!(s_i-1)!}{M!} [0 - s_i] = -\frac{(M-s_i-1)!s_i!}{M!}.$$

When $\mathbf{1}_{Z_{i,j}=1} = 1$ we get

$$\frac{(M-s_i-1)!(s_i-1)!}{M!} [(M-1) - (s_i-1)] = \frac{(M-s_i-2)!(s_i-1)!}{M!}.$$

So, back to our formula for the coefficient ϕ_j , we write $y_i = f_v(S_i)$, define $\mathcal{N} = \{1, \dots, M\}$ and use the simplified form of each component of the sum, to obtain:

$$\begin{aligned} \phi_j &= \sum_{S_i \subseteq \mathcal{N}, j \in S_i} \frac{(M-s_i-2)!(s_i-1)!}{M!} f_v(S_i) + \sum_{S_i \subseteq \mathcal{N}, j \notin S_i} -\frac{(M-s_i-1)!s_i!}{M!} \cdot f_v(S_i) \\ &= \sum_{S_i \subseteq \mathcal{N}, j \notin S_i} \frac{(M-s_i-1)!s_i!}{M!} f_v(S_i \cup \{j\}) - \sum_{S_i \subseteq \mathcal{N}, j \notin S_i} \frac{(M-s_i-1)!s_i!}{M!} \cdot f_v(S_i) \\ &= \sum_{S_i \subseteq \mathcal{N}, j \notin S_i} \frac{(M-s_i-1)!s_i!}{M!} [f_v(S_i \cup \{j\}) - f_v(S_i)]. \end{aligned}$$

Assuming model linearity and feature independence,

$$f_v(S_i \cup \{j\}) - f_v(S_i) = \mathbb{E}[f_v(\mathbf{X}, \mathbf{A}_{S_i \cup \{j\}}) | \mathbf{X}_{S_i \cup \{j\}}] - \mathbb{E}[f_v(\mathbf{X}, \mathbf{A}_{S_i}) | \mathbf{X}_{S_i}],$$

which is the form of Shapley value ϕ_j in the case of graphs. This proves that GraphSVX computes the extension of Shapley values to graphs via the coefficients of its weighted linear regression g . This however requires GraphSVX to use all 2^M samples for an exact computation. As this is computationally expensive, as we presented in the main paper, we have to resort to an approximation.

E Evaluation

In this section, we provide additional information about the evaluation phase. All experiments are conducted on a Linux machine with an Nvidia Tesla V100-PCIE-32GB model, driver version 455.32.00 and Cuda 11.1. GraphSVX is implemented using Pytorch 1.6.0.

E.1 Synthetic and real datasets with ground truth

We follow the setting exposed in [42] and construct four kinds of node classification datasets: *BA-Shapes*, *BA-Community*, *Tree-Cycles*, *Tree-Grids* as well as two graph classification datasets: *BA-2motifs* and *MUTAG*. Table 2 displays the statistics of these datasets.

	Node Classification				Graph Classification	
	<i>BA-Shapes</i>	<i>BA-Community</i>	<i>Tree-Cycles</i>	<i>Tree-Grid</i>	<i>BA-2motifs</i>	<i>MUTAG</i>
#graphs	1	1	1	1	1,000	4,337
#classes	4	8	2	2	2	2
#nodes	700	1,400	871	1,231	25,000	131,488
#features	10	10	10	10	10	14
#edges	4,110	8,920	1,950	3,410	51,392	266,894

Table 2. Statistics of datasets with ground truth.

- *BA-Shapes* consists of a single graph with Barabasi-Albert (BA) base graph composed of 300 nodes and 80 “house”-structured motifs. These motifs are attached to randomly selected nodes from the BA graph. In addition, $0.1N$ random edges are added to perturb the graph. Overall, nodes are assigned to one of 4 classes based on their structural role. All the ones belonging to the base graph are labelled with 0. For those in the house motifs, they are labelled with 1,2,3 if they belong respectively to the top/middle/bottom of the “house”. Features are constant across nodes.
- *BA-Community* dataset is a union of two *BA-Shapes* graphs. Node features are sampled using two Gaussian distributions, one for each *BA-Shapes* graph. Nodes are labelled based on their structural role and community membership, leading to 8 classes in total.
- *Tree-Cycles* dataset has an 8-level balanced binary tree as base graph. A set of 80 cycle motifs of size 6 are attached to randomly selected nodes from the base graph.
- *Tree-Grid* is constructed in the same way as *Tree-Cycles*, except that the cycle motifs are replaced by 3-by-3 grid motifs. Like the above, it is designed for node classification.
- *BA-2motifs* is made for graph classification tasks and contains 800 graphs. We adopt the BA graph as base. Half of the graphs are attached with “house” motifs and the rest with five-node cycle motifs. Graphs are assigned to one of 2 classes according to the type of attached motifs.
- *MUTAG* is a real-life dataset for graph classification composed of 4,337 molecule graphs, each assigned to one of 2 classes given its mutagenic effect on the Gram-negative bacterium *S.typhimurium* [28]. According to [9], carbon rings with chemical groups NO2 or NH2 are mutagenic. Since carbon rings exist in both mutagenic and nonmutagenic graphs, they are not discriminative and are thus treated as the shared base graph. NH2, NO2 are viewed as motifs for the mutagen graphs. Regarding nonmutagenic ones, there are no explicit motifs so we do not consider them.

Experimental setup. We share a single GNN model architecture across all datasets. We write as $\text{GNN}(a, b, f)$ a GNN layer with input dimension a , b output neurons, and activation function f . A similar notation applies for a fully connected layer, $\text{FC}(a, b, f)$. For node classification, the structure is the following: $\text{GNN}(10, 20, \text{ReLU})$ - $\text{GNN}(20, 20, \text{ReLU})$ - $\text{GNN}(20, 20, \text{ReLU})$ - $\text{FC}(20, \text{\#label}, \text{softmax})$. For graph classification, $\text{GNN}(10, 20, \text{ReLU})$ - $\text{GNN}(20, 20, \text{ReLU})$ - $\text{GNN}(20, 20, \text{ReLU})$ - $\text{Maxpooling-FC}(20, \text{\#label}, \text{softmax})$. Each model is trained for 1,000 epochs with Adam optimizer and initial learning rate 1.0×10^{-3} . For all datasets, we use a train/validation/test split of 80/10/10%. The accuracy metric reached on each dataset is displayed in Table 3. The results are good enough to make explanations of this model relevant.

	Node Classification				Graph Classification	
	<i>BA-Shapes</i>	<i>BA-Community</i>	<i>Tree-Cycles</i>	<i>Tree-Grid</i>	<i>BA-2motifs</i>	<i>MUTAG</i>
Training	0.97	0.96	0.98	0.90	1.00	0.86
Validation	0.98	0.87	0.98	0.89	1.00	0.84
Testing	0.96	0.88	0.98	0.87	1.00	0.85

Table 3. Accuracy performance of GNN models.

E.2 Real-life datasets without ground truth

We use two real-world datasets: *Cora* and *PubMed*, whose statistical details are provided in Table 4. We add noise to these datasets for the next experiments and obtain six datasets in total.

- *Cora* is a citation graph where nodes represent machine learning papers and edges represent citations between pairs of papers. The task involved is document classification where the goal is to categorise each paper into one of 7 categories. Each feature indicates the absence/presence of the corresponding word in its abstract.
- *PubMed* is also a publication dataset where each feature indicates the TF-IDF value of the corresponding word. The task is also document classification into one of 3 classes.
- *NFCora* is the *Cora* dataset where we have added 20% of noisy features (287) to all nodes. These new noisy features are binary and have a similar distribution as existing features. We sample their value using a Bernoulli(p) distribution, where $p = 0.013$.
- *NFPubMed* is the *PubMed* dataset where we have added 20% of noisy features (100) to all nodes. These new noisy features are continuous and have a similar distribution as existing features: they follow a Uniform distribution within $[0,1)$ with probability 0.1 and take a null value otherwise.
- *NNCora* is the *Cora* dataset where we have added 20% of noisy nodes to the graph (541). Each new noisy node is connected to an existing node with probability 0.003, to present similar connectedness properties as existing nodes. Features of these new nodes are defined as for *NFCora*.
- *NNPubMed* is the *PubMed* dataset where we have added 20% of noisy nodes to the graph. Each new node is connected to an existing node according to a Bernoulli(0.0005) distribution, to present similar connectedness properties as existing nodes. Features of these new nodes are defined as for *NFPubMed*.

Datasets	<i>Cora</i>	<i>Pubmed</i>
#classes	7	3
#nodes	2,708	19,717
#features	1,433	500
#edges	5,429	44,338

Table 4. Statistics of *Cora* and *Pubmed* datasets.

Model-Data	<i>Cora</i>	<i>Pubmed</i>	<i>NFCora</i>	<i>NFPubMed</i>	<i>NNCora</i>	<i>NNPubMed</i>
GCN-Train	0.91	0.85	0.92	0.85	0.90	0.73
GCN-Val	0.88	0.89	0.88	0.88	0.86	0.77
GCN-Test	0.86	0.88	0.86	0.86	0.84	0.77
GAT-Train	0.78	0.83	0.79	0.83	0.74	0.76
GAT-Val	0.88	0.88	0.88	0.88	0.86	0.85
GAT-Test	0.86	0.85	0.86	0.85	0.86	0.83

Table 5. Performance of GCN/GAT models – on original and noisy *Cora* and *PubMed* datasets. *NFCora* signifies *Cora* dataset with Noisy Features, *NNCora* points to *Cora* dataset with Noisy Nodes added. Similarly for *NFPubMed* and *NNPubMed*.

Experimental setup. We use the same notation as before to describe GCN and GAT models’ architecture, except for GAT where we add one last element representing the number of attention heads. For all models below, we use an Adam optimiser and a negative log-likelihood loss function.

For *Cora*, we train a 2-layer GCN model GCN(1433, 16, ReLU)-GCN(16, 7, ReLU) with 50 epochs, dropout= 0.5, learning rate (lr) = 0.01 and weight-decay (wd) = $5e - 4$. We also train a 2-layer GAT

model with structure: GAT(1433, 8, ReLU, 8)-GAT(8, 7, ReLU, 1) with 80 epochs, dropout= 0.6, lr= 0.005, wd= $5e - 4$.

For *PubMed*, we train a 2-layer GCN model GCN(500, 16, ReLU)-GCN(16, 3, ReLU) with dropout= 0.5, 150 epochs, lr= 0.01, wd = $5e - 4$. We also train a 2-layer GAT model with structure: GAT(500, 8, ReLU, 8)-GAT(8, 3, ReLU, 8) with 120 epochs, dropout= 0.6, lr= 0.005 and wd= $5e - 4$.

The results provided in the paper are obtained by constructing a Kernel Density Estimator (KDE) plot from the distribution of noisy nodes/features that are included in top-k explanations for 50 different test samples (50 node predictions explained), using the above trained GAT model.

E.3 Ablation study and hyperparameters tuning

During the evaluation process, we have tested the impact of different hyperparameters as well as different versions of GraphSVX so as to validate the improvements we thought of. Here is a non-exhaustive list of the conclusions we were able to draw for this analysis:

- We show empirically that **reducing the number of features and nodes** considered from $F + N$ (all to $D + B$ (D : k -hop neighbours; B : features not within a confidence interval around the mean value or 0) is extremely significant. On average, features that are removed from consideration occupy 5% of the explainable part $f_v(\mathbf{X}, \mathbf{A}) - \mathbb{E}[f_v(\mathbf{X}, \mathbf{A})]$ when considered. Similarly, all nodes that are removed occupy 8% of the explainable part.
- Capturing the **indirect effect** of nodes in `GEN()` augments performance by 19% (on average) on *TreeCycles* and *TreeGrid* datasets, where higher-order relations matter more and more coalitions are sampled. For *BA-Shapes* and *BA-Community*, where most nodes in the ground truth are 1-hop neighbours, higher order relation are less important and adding indirect effect has no effect on performance.
- The extension involving considering the influence of a certain **feature from the subgraph perspective** instead of v 's feature value is highly relevant, especially for our evaluation framework as we add noisy features using Gaussian distributions on all nodes. It is thus easier to spot noisy features on *Cora* and *PubMed*. On *BA-Community*, we demonstrated its higher ability to capture global feature importance for similar reasons (important features are defined using the whole graph and not single nodes), as it achieved 100% accuracy for essential features against 65% for the standard method.
- Enforcing a high (close to infinity) weight to the masks **1** and **0** to satisfy the *efficiency* axiom is not necessary. Identical explanations are reached most of the time when these coalitions are not sampled.
- Weighted Least Squares (WLS) and Weighted Linear Regression (WLR) yield equivalent results. In different terms, the learning method in the **explanation generator** does not impact much the results, although WLR yields slightly better ones. Furthermore, weighted Lasso is not extremely useful for synthetic datasets, rather small and sparse.
- The **base value**, or constant of our explanation model g , approximates the expected model prediction, as stated in the paper. For *Cora*, the class repartition is the following: [0.13, 0.09, 0.15, 0.29, 0.15, 0.11, 0.07], for the seven targets. The base value for all classes, obtained via g are: [0.12, 0.10, 0.15, 0.27, 0.15, 0.12, 0.08], which is a great approximation.
- **GAT** is more effective than **GCN** for our evaluation on real-world datasets without a ground truth, and yields better results for most baselines.
- The **seed** chosen to generate pseudo-random numbers and thus be able to replicate identical experiments impact significantly the results for methods with high variability when a small test sample is chosen. We choose a large test sample to counter this.
- We observe in Table 6 that our SMARTERSEPARATE **mask generator** outperforms sampling baselines on all tasks - for a same number of samples $P \ll 2^{B+D}$ (except ALL which uses $\rightarrow 2^{B+D}$). ALL approximately samples every possible combinations of nodes and features. RANDOM simply chooses randomly binary masks. SMART samples in priority masks with a high weight (wrt g). SMARTSEPARATE additionally separates the effect of nodes and features. Finally, SMARTERSEPARATE is the current approach, described in the paper. It uses a smart space allocation algorithm on top of SMARTSEPARATE.

MASK	<i>BA-Shapes</i>	<i>BA-Community</i>	<i>Tree-Cycles</i>	<i>Tree-Grid</i>
SMARTERSEPARATE	0.99	0.93	0.97	0.93
SMARTSEPARATE	0.94	0.92	0.85	0.83
SMART	0.93	0.91	0.81	0.79
RANDOM	0.84	0.75	0.74	0.64
ALL	0.80	0.79	0.87	0.77

Table 6. RANDOM is improved by SMART, so sampling masks of high weight (few or many variables) is relevant. SMART is itself improved by SMARTSEPARATE, which shows that separating nodes and features in coalitions increases performance for an identical number of coalitions P —in fact reduces complexity. Finally, SMARTER is improved by SMARTERSEPARATE, which justify the efficiency of the smart space allocation algorithm.

Alongside with the Table 6, we remark that increasing P (**number of samples**) and S (**maximum order of coalitions favoured**) is relevant until a certain point, where performance starts to slightly decrease and often converges to ALL coalition results. For example, on *Tree-Cycles*, the performance of default GraphSVX for 500 samples is 0.83, for 1,500 samples 0.95 and for 3,000 samples 0.88; while GraphSVX with ALL method yields 0.87. For simple datasets (*BA-Shapes*, *BA-Community*), few samples is enough to get good accuracy, meaning when looking only at individual effect ($S = 1$). Increasing S and *num_samples* does not change performance, it only increases computational time. Nevertheless, on slightly more complex datasets (*Tree-Cycles*, *Tree-Grid*), optimal performance is reached when considering all coalitions of higher-order ($S = 4$), which require more coalitions to be sampled.

To continue with **parameter sensitivity**, when testing robustness to noise, the proportion of noisy features and nodes introduced, as well as the connectivity of the nodes, the distribution of the features and the number/proportion of most important features we look at all have a significant impact on results. Nevertheless, this impact is consistent across all baselines. We therefore chose a configuration that appeared relevant, with enough noise introduced to differentiate between good and bad explainers.

Running times for explanation of a single node are displayed in Table 8 for all synthetic node classification datasets. Comparing running times with baselines [21, 42], GraphSVX is still slower than PGExplainer, which is very scalable, but matches GNNExplainer thanks to our efficient approximation, especially when the data is relatively sparse (e.g., *BA-Shapes* or *Tree-Cycles*). With a 2-hop subgraph approximation 7, it is faster than GNNExplainer while maintaining state-of-the-art performance.

		<i>BA-Shapes</i>	<i>BA-Community</i>	<i>Tree-Cycles</i>	<i>Tree-Grid</i>
GraphSVX:	Time (s)	2.12	8.31	3.65	6.22
	Samples	400	800	1,400	1,500
GNNExplainer:	Time (s)	6.63	7.08	6.89	7.31

Table 7. Average running time for the explanation of a single node with GraphSVX (3-hop subgraph approximation) and GNNExplainer.

	<i>BA-Shapes</i>	<i>BA-Community</i>	<i>Tree-Cycles</i>	<i>Tree-Grid</i>
Time (s)	0.30	1.09	1.21	1.87
Samples	65	100	300	400

Table 8. Average running time for the explanations of a single node with GraphSVX (2-hop subgraph).

F Properties

F.1 Desirable properties of explanations from a machine learning perspective

- **Accuracy and Fidelity:** How relevant is an explanation? And for an explanation model, how well does it approximate the prediction of the black box model? There must be a ground truth explanation or a human judge to assess the accuracy of an explainer [4].
- **Robustness:** How different are explanations for similar models/instances? It encapsulates consistency, stability, and resistance to noise. High robustness means that slight variations in the features of an instance, or in the model functioning, do not substantially change the explanation. Sampling or random initialisation of mask generator goes usually against robustness [24].
- **Certainty:** Does the explanation reflect the certainty of the machine learning model? In different terms, does the explanation indicate the confidence of the model for the explained instance prediction? In the paper, we refer to it as **decomposability**—the most common way to reflect certainty [29].
- **Meaningfulness:** How well does the explanation reflect the importance of features/nodes? Here, meaningful explanations means “with a nice interpretation/signification” [44].
- **Representativeness (or Global):** How general is an explanation? People like explanations that cover many instances or the general functioning of the model. This global scope is concerned with overall actions and usually provides a pattern that the prediction model discovers [22].
- **Comprehensibility:** How well do humans understand explanations? This property comprises several aspects that are further analysed below [24].

F.2 Human-centric explanations

- **Contrastive:** Humans usually do not ask why a certain prediction was made, but why it was made instead of another prediction. In different terms, humans want to compare it against another instance’s prediction (could be an artificial one) [19].
- **Selective:** People do not expect explanations to cover the actual and complete list of causes of an event. They are used to being given one or two key causes as the explanation [38].
- **Social:** Explanations are part of an interaction between the explainer and the receiver. The social context determines the content and nature of explanations; they should be targeted to the audience [23].

F.3 Explainers’ properties

Method	Task	Target	<i>Decomposable</i>	<i>Global</i>	<i>Robust</i>	<i>Meaningful</i>	<i>Human-centric</i>
GNNExplainer	GC/NC	E/F	×	✓	×	×	✓
PGEExplainer	GC/NC	E	×	✓	×	×	✓
GraphLIME	NC	F	×	×	×	✓	✓
PGM-Explainer	GC/NC	N	×	×	×	×	✓✓
XGNN	GC	N	×	✓	×	×	✓
GraphSVX	GC/NC	N/F	✓	✓	✓	✓	✓✓✓

Table 9. Comparison of explanation methods for GNNs. GC/NC denote graph and node classification tasks. Target explanation points to features (F), edges (E) or nodes (N). *Human-centric* is decomposed into social, contrastive and selective—and one checkmark is attributed for each property that holds. We regroup under *Robust*: Consistent, Stable and robust to noise.

In Table 9, we show how explainers belonging to the unified framework proposed in the paper satisfy these properties. Below, we develop why/how GraphSVX (and the baselines) satisfy them.

Fidel and accurate. GraphSVX achieves state-of-the-art results on all but one task, which proves its high accuracy. It is also fidel to the GNN model locally, as the explanation model g reaches $R^2 > 0.90$ for all tasks.

Decomposable. GraphSVX is the only fairly distributed method. In practice, we verify that $\sum_i \phi_i = f_v(\mathbf{X}, \mathbf{A})$ for all predictions, which is naturally enforced by the high weight given to both null and full coalitions.

Meaningful. Along with GraphLIME, GraphSVX is the only meaningful method—it outputs the “fair” contribution of each feature/node towards the prediction (with respect to the average prediction). Overall, it provides more information than all methods, including GraphLIME. For instance, on synthetic datasets, the importance granted to nodes belonging to a motif achieves 85% of the explainable part $f_v(X, A) - \mathbb{E}[f_v(X, A)]$ for *Tree-Grid* and *Tree-Cycle*, while it represents only 20% on *BA-Community*, where node features and the relation to other motifs (inside another community) are also essential for prediction. This provides key information to an end-user. Note that, we consider GraphLIME as meaningful although the coefficients themselves do not have proper signification and only account for feature importance. None the less, they do give an idea of the influence of a feature in prediction. Regarding other baselines, PGExplainer and GNNExplainer output probability scores, PGM-Explainer a bayesian network, and XGNN a subgraph without further information.

Global. Similarly to GNNExplainer, GraphSVX is endowed with an extension that enables it to explain a subset of nodes together, without aggregating local explanations. Furthermore, it can study feature importance in the subgraph of the node being explained instead of the node itself, which also goes towards a more global comprehension. Two other explainers provide even more general explanations: XGNN, a true model-level explanation method and PGExplainer, which provides collective and inductive explanations.

Robust. GraphSVX is by definition consistent and stable. In addition to showing its robustness to noise in the experimental evaluation section, we further investigate those two properties. We assess stability by computing variance statistics on synthetic datasets explanations, for nodes occupying the same role in a motif. Explanations are very stable across all datasets, with a variance in accuracy always inferior to 0.1. Consistency is less trivial to examine. We compare the explanations obtained for the GCN and GAT models on *Cora*, and compute the intersection over union (IOU) score (0.76) between the two sets of top-20% (or top 10) most important features and nodes. The core differences in behaviour and performance between the GCN and GAT models prevents us from obtaining a better result. Consistency is tricky to assess for every explainer. We justify theoretically why baseline explainers are not considered as robust (w.r.t. to GraphSVX). GraphLIME and PGExplainer use sampling without any approximation guarantees. XGNN uses a reinforcement learning approach, a candidate node set and several deep learning models to produce subgraph explanations. PGExplainer initialises randomly the parameters of the mask generator and uses intermediate model representations, which could respectively lead to local optimum and differences across similar models. Similarly for GNNExplainer but in a smaller extent since it does not use intermediate representations.

Human-centric. Let’s split this part into different sub-properties. (1) **Selective.** Instead of fitting a WLR explanation model g , we fit instead a weighted Lasso regression, which produces sparser results, more easily comprehensible by humans [38]. (2) **Contrastive.** As detailed in the paper, we extend GraphSVX to enable explaining an instance with respect to another instance. (3) **Social.** Explanations should be adapted to the target audience. Here, we offer complete explanations appropriate for domain experts, which answers the recent European regulations concerning the legal right to explanation: GDPR [13]. We also provide for non-domain expert selective explanations with intuitive visualisations. Furthermore, the pipeline is flexible. We can fully regulate the amount of importance granted to nodes and features, either via shutting down completely the importance given to either category, or by rescaling it:

$$\phi_{\text{node}} = \phi_{\text{node}} \cdot \frac{\alpha * e}{\sum(\phi_{\text{node}})} \quad \text{and} \quad \phi_{\text{feat}} = \phi_{\text{feat}} \cdot \frac{(1 - \alpha) * e}{\sum(\phi_{\text{feat}})},$$

with $e = (f_v(\mathbf{X}, \mathbf{A}) - \mathbb{E}[f_v(\mathbf{X}, \mathbf{A})])$ and $\alpha \in [0, 1]$ the regularisation coefficient. In the above table, this leads to 3 checkmarks out of 3 for GraphSVX. PGM-Explainer is selective and social, as it produces a causal graph with feature selection, but is not contrastive. GNNExplainer, XGNN, GraphLIME and PGExplainer are simply selective.