IEEE Access
Multidisciplinary ⁝ Rapid Review ⁝ Open Access Journal

Received 10 January 2023, accepted 27 January 2023, date of publication 6 February 2023, date of current version 14 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3242866

RESEARCH ARTICLE

Graph-Assisted Bayesian Node Classifiers

HAKIM HAFIDI[1,2], (Student Member, IEEE), PHILIPPE CIBLAT[2], (Senior Member, IEEE), MOUNIR GHOGHO[1], (Fellow, IEEE), AND ANANTHRAM SWAMI[3], (Life Fellow, IEEE)

[1]TICLab, College of Engineering and Architecture, Université Internationale de Rabat, Rabat 11103, Morocco
[2]LTCI, Telecom Paris, Institut Polytechnique de Paris, 91764 Palaiseau, France
[3]U.S. Army Research Laboratory, Adelphi, MD 20783, USA

Corresponding author: Hakim Hafidi (hakim.hafidi@uir.ac.ma)

ABSTRACT Many datasets can be represented by attributed graphs on which classification methods may be of interest. The problem of node classification has attracted the attention of scholars due to its wide range of applications. The problem consists of predicting nodes' labels based on their intrinsic features, features of their neighboring nodes and the graph structure. Graph Neural Networks (GNN) have been widely used to tackle this task. Thanks to the graph structure and the node features, they are able to propagate information over the graph and aggregate it to improve the classification performance. Their performance is however sensitive to the graph topology, especially its degree of impurity, a measure of the proportion of connected nodes belonging to different classes. Here, we propose a new Graph-Assisted Bayesian (GAB) classifier, which is designed for the problem of node classification. By using the Bayesian theorem, GAB takes into consideration the degree of impurity of the graph when classifying the nodes. We show that the proposed classifier is less sensitive to graph impurity, and less complex than GNN-based classifiers.

INDEX TERMS Node classification, attributed graphs, degree of impurity, Bayesian framework.

I. INTRODUCTION

Attributed graphs are useful tools for representing interactive phenomena such as social networks [1], financial market fluctuations [2], road or air traffic, human scene [3], brain activity [4], or gene interaction [5]. Graphs are described by *i)* a set of nodes associated with the entities (subscribers in social networks, planes in air traffic, etc.) and *ii)* a set of edges connecting them in order to represent their relationships. Graphs are said to be attributed when nodes and/or edges have assigned values, called *features*. While this type of data is rich for inference, it is not well suited to standard signal processing or machine learning techniques. The adaptation of these techniques to graph data has been the subject of numerous studies that aim to perform graph classification (e.g., molecules, handwritten characters) [6], [7], [8], edge prediction (e.g., social network links) [9], node regression or classification (e.g., citation networks) [10], [11].

This works focuses on node classification, which is one of the most important tasks of machine learning on graphs. Its objective is to predict the class (called also *label*) of each unlabeled node of the graph by relying on both nodes' features and nodes' connections within the graph.

In most real world graphs, connections between nodes are far from arbitrary. In social networks for instance, people are more likely to connect with those who share similar characteristics or areas of interest [12]. A citation network has connections between articles if one cites another, so links between articles addressing the same research topic are more likely than links between those addressing different topics. [13]. As in social sciences literature, a *homophilic* graph can be described as one where similar nodes tend to be connected to each other [12]. Using this principle, one should make use of the topology of the graph to determine the class of a node, rather than relying solely on intrinsic features, as is commonly done with standard machine learning approaches.

When taking into account the graph topology in a classifier, the first question is: what kind of information to share over the graph? There are two approaches.

- Label Propagation (LP): only the labels (true or estimated) are propagated through the adjacent nodes in the graph to make a new decision. Several methods relying on voting have been developed to merge labels' information [14]. The labels at the initial step are either

The associate editor coordinating the review of this manuscript and approving it for publication was Jolanta Mizera-Pietraszko.

provided or estimated using the node's features only, i.e. ignoring the graph.

- Feature propagation (FP): the nodes' features are propagated through the adjacent nodes at each step. Typically, a weighted averaging of the current features of adjacent nodes (sometimes followed by a nonlinear function) is carried out for each node at each step.

In this paper, we focus only on the FP approach. Graph neural networks (GNN) have recently been developed in order to adapt Neural Networks to attributed graphs. Typically, at each layer, they linearly combine the features of adjacent nodes and then apply an activation function. The training on labelled nodes consists of finding out the best weights of each linear combination. In this paper, we propose to derive in closed-form a classifier relying on (Bayesian) decision theory. As a result, we obtain an interpretable algorithm based on some parameters whose estimation step replaces the training phase of the GNN.

Before we go any further, let us briefly review the works on FP based on GNN. The most common way has been to extend the convolutional neural network (CNN) to the graph structure [15] by redefining the convolution operator on the graph domain. These CNN-based methods suffer from a high computational cost due to the necessity of performing an eigendecomposition of the graph Laplacian. To overcome this problem, different approaches avoid explicit computation of this eigendecomposition by using a polynomial expansion to represent the filters [16], [17]. More specifically, in [17], the authors consider a first-order polynomial approximation to build a neural network which they called Graph Convolutional Network (GCN) to do semi-supervised node classification. The GCN can also be seen as an aggregation operator, i.e. the representation of a node is obtained by averaging its intrinsic features with those of its first-order neighbors. The authors in [18] and [19] respectively introduced Graph Attention Network (GAT) and Attention-based Graph Neural Network (AGNN) where different weights are assigned to neighbors based on nodes' and edges' features. Other researchers explored higher-order (or equivalently, multi-hop) information of the graph by repeatedly mixing features of neighbors at various distances [20] or by modifying the propagation strategy of GCN [21]. Although these approaches have achieved remarkable results on a number of benchmark datasets, we notice that their performance vary significantly across datasets. For instance, the gain compared to a simple logistic regression (i.e. no contribution from the neighbors) highly depends on the dataset.

We use the following example to explain the underlying reason for the performance variations over datasets. We define $\bar{p}$ as the average probability of intra-class connection (i.e., the probability that two nodes from the same class are connected), and $\bar{q}$ as the average probability of inter-class connection (i.e., the probability that two nodes from different classes are connected). The degree of *impurity* of the graph may be represented by the ratio $\bar{q}/\bar{p}$. Datasets with a degree of impurity less than 1 are called *assortative* [22] and correspond to graphs containing communities (nodes with similar features are connected to each other). In Table 1, we show the classification accuracies of a logistic regression classifier and a two-layer GCN for two widely-used datasets used for benchmarking GNN algorithms. We also display the estimated values for $\bar{p}$, $\bar{q}$ and the degree of impurity $\bar{q}/\bar{p}$. As expected, node classification using graph structure is easier with graphs offering low degree of impurity (like Cora). This may explain the performance variations over datasets.

TABLE 1. Intra- and inter-class connection probabilities and classification accuracies.

	Cora	Citeseer
Intra-class connectivity (p)	23×10^{-3}	12×10^{-3}
Inter-class connectivity (q)	5.5×10^{-3}	4.3×10^{-3}
Degree of Impurity (q/p)	0.23	0.36
Logistic Regression (LR)	56.0%	57.2%
Two-layer GCN [17]	81.5%	70.3%
Gain between GCN and LR	+45.5%	+22.9%

In this paper, a different point of view is taken by proposing a Bayesian classifier which does not rely on a neural network structure and is able to better adapt to the level of impurity than GNN-based classifiers.

Our approach to tackle the node classification problem is related to the so-called collective classification [23], [24] which refers to the classification of a set of connected nodes by using their intrinsic features and/or labels and their relative connections. Optimal collective classifications are carried out by maximizing the joint likelihood. However, optimal (and so exact) inference is an NP-hard issue in general, and is thus generally not well suited for the real-world networks. As a consequence, most collective classifiers rely on developing approximate inference [23], [25]. Some recent studies combine methods from collective classification with neural networks to ensure a better end-to-end learning, e.g. [26]. However, all the above-mentioned algorithms only make use of features of first-hop neighbors and thus rely on a propagation step to make use of higher-hop nodes' information in an iterative manner. We instead introduce a new classifier that directly takes into account higher-hop nodes. This has the additional advantage of being more interpretable. In our previous work [27], we proposed a first order version of the Bayesian based classifier which is only able to take into account first order neighbors. In this paper, we conduct detailed derivations of the classifier which allow us to consider higher orders. We also conducted a deeper comparison between our classifier and GNN based classifiers in terms of performance and complexity.

Let us define some notations. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph where $\mathcal{V}$ is a set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges. Each node $u \in \mathcal{V}$ is represented by a feature vector $\boldsymbol{x}_u \in \mathbb{R}^{F \times 1}$ where F is the number of node's features. An adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ represents the topological structure of the graph where $N = |\mathcal{V}|$ is the number of nodes in the graph. Without loss of generality we assume

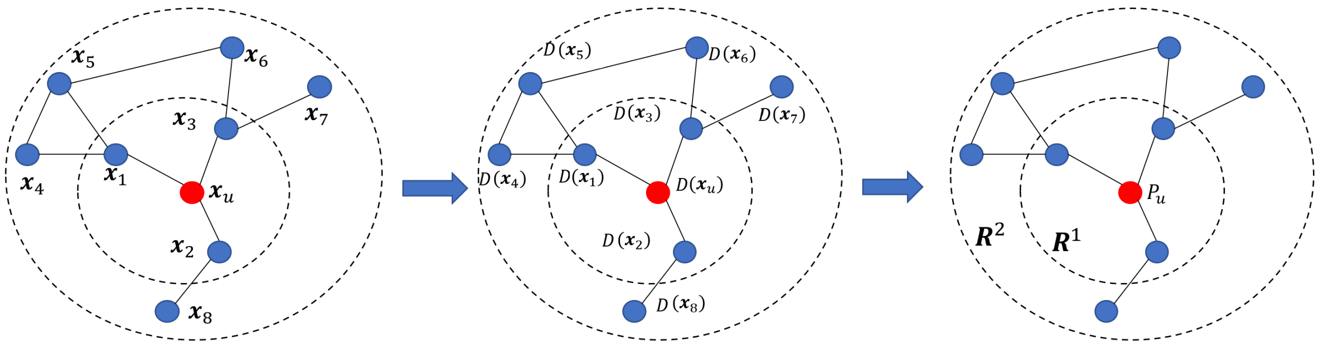

FIGURE 1. An illustration of second order GAB for a node of interest u where $\mathcal{N}_1(u) = \{1, 2, 3\}$ and $\mathcal{N}_2(u) = \{4, 5, 6, 7, 8\}$. First, we estimate the probability of belonging to each one of the classes based on nodes' features $D(x_u) = (D_1(x_u), \ldots, D_k(x_u))$ either by training a multi-layer perceptron or by estimating parameters of the generating distributions of nodes' features. Then we compute $P_u = (P_u(1), \ldots, P_u(k))$ using R^1 as a transition matrix for nodes in $\mathcal{N}_1(u)$ and R^2 for nodes in $\mathcal{N}_2(u)$ (See Eq. (14)).

the graph to be unweighted i.e $A_{u,v} = 1$ if $(u, v) \in \mathcal{E}$ and $A_{u,v} = 0$ otherwise. Let $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N]^\top$. Let y_u denote the label of the u-th node and let K denote the number of classes, i.e. $y_u \in \{1, \ldots, K\}$.

The objective of node classification in this paper is to predict the class of all unlabeled nodes in the graph given the adjacency matrix $\boldsymbol{A}$, the feature matrix $\boldsymbol{X}$ and the set of available labels.

The paper is organized as follows: in Section II, we introduce our new graph-assisted Bayesian classifier (GAB). In Section III, we compare our approach with GNN-based classifiers. In particular, we show our classifier has the shape of a GNN only in the noise-free case (i.e., $q = 0$) and under further conditions on the distribution of the features. In Section IV, complexity issues are discussed. In Section V, numerical results are provided. Comparison with existing GNN-based methods are done on real datasets whose *degree of impurity* has been modified by injecting artificial noise (i.e., introducing fictitious edges between classes). We see that our proposed Bayesian classifier offers better performance than GNN-based classifiers. In Section VI, concluding remarks are drawn.

II. NEW GRAPH-ASSISTED BAYESIAN CLASSIFIER

A. THE CLASSIFIER DERIVATION

In this section, we derive our new GAB classifier based on Maximum A Posteriori (MAP) principle. We develop this classifier for a node u, called *node of interest* in the rest of the paper. Obviously, in practice, any node in the graph will be seen as a node of interest in a Round-Robin manner. We first consider the entire graph and then, in order to simplify the derivations, we consider only the information provided by the hop distance between the node and a node of interest.

Let

- $\mathcal{V}_u$ be the set of nodes which will be involved in the classification of node u. Node u is not included in this set.
- $\mathcal{X}_u = \{\boldsymbol{x}_u\} \cup \{\boldsymbol{x}_v, v \in \mathcal{V}_u\}$ be the set of feature vectors of node u and its "helping" nodes.
- D_k be the distribution generating the feature vectors of nodes belonging to class k.

We do not assume that theses distributions are known. We instead either assume a shape for these distributions and then estimate their parameters, or approximate them using a neural network. The objective is to compute the posterior probability that a node u belongs to class k knowing information on the graph $\mathcal{I}_\mathcal{G}$ (typically its partial connectivity through the set $\mathcal{V}_u$), and $\mathcal{X}_u$. Consequently, the classifier makes the following decision

$$\hat{y}_u = \arg\max_k P_u(k).$$

where the posterior probability that needs to be computed is defined as:

$$P_u(k) = P(y_u = k | \mathcal{X}_u, \mathcal{I}_\mathcal{G}).$$

Using the Bayes' rule, we obtain

$$\begin{aligned} P_u(k) &= P(y_u = k | \mathcal{X}_u, \mathcal{I}_\mathcal{G}) \\ &= \frac{P(\mathcal{X}_u | y_u = k, \mathcal{I}_\mathcal{G}) P(y_u = k | \mathcal{I}_\mathcal{G})}{P(\mathcal{X}_u | \mathcal{I}_\mathcal{G})} \\ &\propto P(\mathcal{X}_u | y_u = k, \mathcal{I}_\mathcal{G}) P(y_u = k) \qquad (1) \end{aligned}$$

since the denominator does not depend on k and the prior probability of an individual node u to belong to class k does not depend on the graph information. The above posterior probability can be rewritten as

$$P_u(k) \propto Q_u(k)\pi_k \qquad (2)$$

with

$$\begin{cases} Q_u(k) = P(\mathcal{X}_u | y_u = k, \mathcal{I}_\mathcal{G}), \\ \pi_k \quad = P(y_u = k). \end{cases}$$

Let V be the size of the set $\mathcal{V}_u$. Let $\{v_1, \cdots, v_V\}$ be the nodes of $\mathcal{V}_u$. In Appendix A, we show that

$$\begin{aligned} Q_u(k) = D_k(\boldsymbol{x}_u) &\sum_{k_{v_1}, \cdots, k_{v_V} = 1}^{K} \prod_{\ell=1}^{V} D_{k_{v_\ell}}(\boldsymbol{x}_{v_\ell}) \\ &\times p(y_{v_1} = k_{v_1}, \cdots, y_{v_V} = k_{v_V} | y_u = k, \mathcal{I}_\mathcal{G}). \quad (3) \end{aligned}$$

Eq. (3) cannot be simplified further since the term $p(y_{v_1} = k_{v_1}, \cdots, y_{v_V} = k_{v_V}|y_u = k, \mathcal{I_G})$ cannot be split into individual posterior probabilities. Indeed according to Example 1 below, one can see that in general $p(y_{v_1} = k_{v_1}, \cdots, y_{v_V} = k_{v_V}|y_u = k, \mathcal{I_G}) \neq \prod_{\ell=1}^{V} p(y_{v_\ell} = k_{v_\ell}|y_u = k, \mathcal{I_G})$.

Example 1: Let us consider the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{A, B, C\}$ and $\mathcal{E} = \{(B, C)\}$. We set the number of classes to $K = 3$. We assume that only nodes belonging to the same class can be connected. We now compute $P(y_B = k, y_C = k'|y_A = 1, \mathcal{I_G})$ in two different cases.

In the first case, we consider that $\mathcal{I_G}$ provides only information on the connections between the pairs (A, B) and (A, C). Consequently, we only know A is not connected to B and A is not connected to C. This leads to the following expression for $k \in \{2, 3\}$

$$P(y_B = k|y_A = 1, \mathcal{I_G}) = P(y_C = k|y_A = 1, \mathcal{I_G}) = 0.5, \tag{4}$$

and

$$P(y_B = k, y_C = k'|y_A = 1, \mathcal{I_G}) = 0.25 \tag{5}$$

which is equal to $P(y_B = k|y_A = 1, \mathcal{I_G}) \times P(y_C = k'|y_A = 1, \mathcal{I_G})$.

In the second case, we consider that $\mathcal{I_G}$ provides information on all possible connections. Once again, for $k \in \{2, 3\}$

$$P(y_B = k|y_A = 1, \mathcal{I_G}) = P(y_C = k|y_A = 1, \mathcal{I_G}) = 0.5. \tag{6}$$

But we now have for $k \neq k' \in \{2, 3\}$

$$P(y_B = k, y_C = k'|y_A = 1, \mathcal{I_G}) = 0, \tag{7}$$

which is different from $P(y_B = k|y_A = 1, \mathcal{I_G}) \times P(y_C = k'|y_A = 1, \mathcal{I_G})$. □

Consequently, in order to pursue analytical derivations offering practical algorithms, we make the following simplifying assumption:

$$\begin{aligned} p(y_{v_1} &= k_{v_1}, \cdots, y_{v_V} = k_{v_V}|y_u = k, \mathcal{I_G}) \\ &= \prod_{\ell=1}^{V} p(y_{v_\ell} = k_{v_\ell}|y_u = k, \mathcal{I_G}) \end{aligned} \tag{8}$$

which corresponds to assuming statistical independence between the classes of node u's neighboring nodes given the graph. As seen in Example 1, this independence generally does not hold true, but it is required to pursue closed-form derivations.

Using Eqs. (3) and (8), we obtain

$$\begin{aligned} Q_u(k) = D_k(\boldsymbol{x}_u) &\sum_{k_{v_1}, \cdots, k_{v_V}=1}^{K} \prod_{\ell=1}^{V} D_{k_{v_\ell}}(\boldsymbol{x}_{v_\ell}) \\ &\times \prod_{\ell'=1}^{V} p(y_{v_{\ell'}} = k_{v_{\ell'}}|y_u = k, \mathcal{I_G}) \\ = D_k(\boldsymbol{x}_u) &\sum_{k_{v_1}, \cdots, k_{v_V}=1}^{K} \prod_{\ell=1}^{V} D_{k_{v_\ell}}(\boldsymbol{x}_{v_\ell}) \\ &\times p(y_{v_\ell} = k_{v_\ell}|y_u = k, \mathcal{I_G}) \\ = D_k(\boldsymbol{x}_u) & \\ &\times \prod_{\ell=1}^{V} \left(\sum_{k'=1}^{K} p(y_{v_\ell} = k'|y_u = k, \mathcal{I_G}) D_{k'}(\boldsymbol{x}_{v_\ell}) \right). \end{aligned}$$

Finally, we get

$$Q_u(k) = D_k(\boldsymbol{x}_u) \prod_{\ell=1}^{V} \left(\sum_{k'=1}^{K} r_{u,v_\ell}(k, k') D_{k'}(\boldsymbol{x}_{v_\ell}) \right) \tag{9}$$

with

$$r_{u,v}(k, k') = p(y_v = k'|y_u = k, \mathcal{I_G}). \tag{10}$$

The term $r_{u,v}(k, k')$ is the probability to be in class k' for node v given that node u is in class k and that we have partial (or total) information on the graph $\mathcal{I_G}$. Notice that the term $r_{u,v}(k, k')$ depends on the graph statistics as we will see in Eq. (13). In general, deriving a closed-form expression of $r_{u,v}(k, k')$ with respect to the graph statistics is very difficult due to combinatorial complexity.

Before going further, we make the following remark.

Remark 1 (How to Take Into Account the Labelled Nodes?): Some of the nodes in $\mathcal{V}_u$ may have already been labeled, so their classes are known. Eq. (9) should then be adapted to account for this knowledge. Let us consider that node v_1 is labelled and belongs to class k_1. The following term

$$\sum_{k'=1}^{K} r_{u,v_1}(k, k') D_{k'}(\boldsymbol{x}_{v_1})$$

should be replaced with

$$\sum_{k'=1}^{K} r_{u,v_1}(k, k') \delta_{k',k_1} = r_{u,v_1}(k, k_1)$$

where δ is the so-called Kronecker index. Consequently, if $\mathcal{V}_u = \mathcal{L}_u \cup \mathcal{U}_u$ with $\mathcal{L}_u$ being the set of labelled nodes and $\mathcal{U}_u$ being the set of unlabelled nodes, we have that

$$\begin{aligned} Q_u(k) = D_k(\boldsymbol{x}_u) &\prod_{v \in \mathcal{L}_u} r_{u,v}(k, k_v) \\ &\times \prod_{v \in \mathcal{U}_u} \left(\sum_{k'=1}^{K} r_{u,v}(k, k') D_{k'}(\boldsymbol{x}_v) \right) \end{aligned}$$

where k_v is the class of the labelled node v.

As an example, we consider $\mathcal{L}_u = \{v_1\}$ and $\mathcal{U}_u = \emptyset$, and obtain

$$Q_u(k) = D_k(\boldsymbol{x}_u).r_{u,v_1}(k, k_1),$$

which is the likelihood function for $\boldsymbol{x}_u$ assuming class k, corrected by a term depending on the probability of connection between class k and that of the labelled neighbor node.

In order to obtain a practical algorithm, $\mathcal{I_G}$ for each node of interest, u, will be made to consist of only the distances between this node and other nodes of the graph, i.e.,

$$\mathcal{I_G} = \{\text{distance of each node of } \mathcal{G} \text{ to root } u\}.$$

Therefore, we order $\mathcal{V}_u$ as follows

$$\mathcal{V}_u = \mathcal{N}_1(u) \cup \mathcal{N}_2(u) \cdots \cup \mathcal{N}_{\Delta_u}(u)$$

where $\mathcal{N}_d(u)$ is the set of d-hops neighbors of u and Δ_u is the maximum distance from node u; we have that $\Delta_u \leq V$. Hence, Eq. (9) can be rewritten as

$$\begin{aligned} Q_u(k) &= D_k(\boldsymbol{x}_u) \\ &\quad \times \prod_{d=1}^{\Delta_u} \prod_{v \in \mathcal{N}_d(u)} \left(\sum_{k'=1}^{K} r_{u,v}(k,k') D_{k'}(\boldsymbol{x}_v) \right). \end{aligned}$$

The above expression will be next simplified further.

We recall that $r_{u,v}(k,k')$ is equal to $p(y_v = k' | y_u = k, \mathcal{I}_\mathcal{G})$. As $\mathcal{I}_\mathcal{G}$ now merely consists of the distances between each node and the node of interest, $r_{u,v}(k,k')$ no longer depends on the specific path(s) connecting u to v but only hop distance, between node v and node u. Moreover we assume that the probability of connection between two nodes only depends on the nodes' classes but not on the specific nodes. Consequently, $r_{u,v}(k,k')$ is replaced with $r^{(d)}(k,k')$.

To control the contributions of neighbors to the computation of the posterior probability, depending on their distance to node u, we propose to modify the expression of $Q_u(k)$ as follows

$$Q_u(k) = D_k(\boldsymbol{x}_u) \prod_{d=1}^{\Delta_u} \prod_{v \in \mathcal{N}_d(u)} \left(\sum_{k'=1}^{K} r^{(d)}(k,k')^{\gamma_d} D_{k'}(\boldsymbol{x}_v) \right). \tag{11}$$

where $\{\gamma_d\}_{d=1,\cdots,\Delta_u}$ are hyper-parameters to tune. Simulation results show that hyper-parameter γ_d decreases with d, as expected. Setting $\gamma_d = 1$ for all values of d implies that all the nodes in $\mathcal{V}_u$ have the same weights and so contribute equally. This can be counter-productive especially when the degree of impurity is high. Indeed, information from distant nodes may not be reliable, mostly because of the simplifications made to $\mathcal{I}_\mathcal{G}$. The optimization of the hyperparameters should counter-act this kind of phenomenon.

The goal now is to derive $r^{(d)}(k,k')$. Let $\mathbf{R}^{(d)} \in \mathbb{R}^{K \times K}$ be the matrix whose (k,k')th element is $r^{(d)}(k,k')$. It is worth pointing out that, in general, $\mathbf{R}^{(1)}$ is not symmetric but is non-negative with the row-sums equal to 1 (while the column-sums are not necessary equal to 1). Consequently, $\mathbf{R}^{(1)}$ (shortened to $\mathbf{R}$) is a row-stochastic matrix. First of all, we have that

$$r^{(d)}(k,k') = P(y_v = k' | y_u = k, \mathcal{C}_d)$$

where $\mathcal{I}_\mathcal{G}$ has been replaced with $\mathcal{C}_d$, which is defined as the knowledge that both considered nodes are connected in d hops.

In Appendix B, we show that

$$\mathbf{R}^{(d)} = \mathbf{R}^d. \tag{12}$$

Since we are able to find $r^{(d)}(k,k')$ with respect to $r^{(1)}(k,k')$, we should now derive $r^{(1)}(k,k')$ in closed-form. We first define the following parameters:

- Let $p(k)$ denote the probability that any two randomly selected nodes belonging to the same class k are directly connected.
- Let $q(k,k')$ denote the probability that two randomly selected nodes not belonging to the same class are connected. We let $q(k,k) = p(k)$. We also assume symmetry, i.e. $q(k,k') = q(k',k)$.
- Let $\bar{p}$ define the average probability of connection between nodes of the the same class, i.e.

$$\bar{p} = \frac{1}{K} \sum_{k=1}^{K} p(k).$$

Similarly, let $\bar{q}$ define the average probability of connection between nodes not belonging to the same class, i.e.

$$\bar{q} = \frac{1}{K(K-1)} \sum_{\substack{k,k'=1 \\ k \neq k'}}^{K} q(k,k').$$

- The degree of impurity (DoI), roughly evoked in Section I, is defined as

$$\text{DoI} = \frac{\bar{q}}{\bar{p}}.$$

This defines a general stochastic block model (SBM), a widely used random graph model for community detection and clustering [28], [29]. Note that we use SBM only for analytic tractability, and that unlike the work on community detection, we are interested in node classification, given some labeled samples.

By using Bayes' rule, we have that

$$\begin{aligned} r^{(1)}(k,k') &= P(y_B = k' | y_A = k, \widetilde{\mathcal{I}}_\mathcal{G}) \\ &= \frac{P(\widetilde{\mathcal{I}}_\mathcal{G} | y_B = k', y_A = k) P(y_B = k' | y_A = k)}{P(\widetilde{\mathcal{I}}_\mathcal{G} | y_A = k)} \\ &= \frac{P(\widetilde{\mathcal{I}}_\mathcal{G} | y_B = k', y_A = k) P(y_B = k' | y_A = k)}{\sum_{k''=1}^{K} P(\widetilde{\mathcal{I}}_\mathcal{G} | y_A = k, y_B = k'') P(y_B = k'')} \\ &= \frac{q(k,k') P(y_B = k' | y_A = k)}{\sum_{k''=1}^{K} q(k,k'') P(y_B = k'')} \\ &= \frac{q(k,k') P(y_B = k')}{\sum_{k''=1}^{K} q(k,k'') P(y_B = k'')} \\ &= \frac{q(k,k') \pi_{k'}}{\sum_{k''=1}^{K} q(k,k'') \pi_{k''}}. \end{aligned} \tag{13}$$

As an example, if the probabilities of connection do not depend on the classes, i.e., $p = p(k)$ for any k and $q = q(k,k')$ for any $k \neq k'$, we obtain

$$r^{(1)}(k,k') = \begin{cases} \frac{p\pi_k}{p\pi_k + q \sum_{\substack{k''=1 \\ k'' \neq k}} \pi_{k''}} & k = k' \\ \frac{q\pi_{k'}}{p\pi_k + q \sum_{\substack{k''=1 \\ k'' \neq k}} \pi_{k''}} & k \neq k' \end{cases}.$$

Hence, according to Eqs. (2) and (11), we obtain

$$\hat{u}_u = \arg\max_k \pi_k D_k(\boldsymbol{x}_u) \prod_{d=1}^{\Delta_u} \prod_{v \in \mathcal{N}_d(u)}$$

$$\times \left(\sum_{k'=1}^{K} r^{(d)}(k,k')^{\gamma_d} D_{k'}(\boldsymbol{x}_v) \right) \tag{14}$$

with $r^{(d)}(k,k')$ given by Eq. (12). We notice that the shape of the function to maximize is simple and corresponds to a product over all the considered nodes of a weighted sum of the possible distributions. The weights are perfectly characterized thanks to our derivations.

B. PARAMETERS' ESTIMATION

In order to implement our node classifier, i.e. to compute Eq. (14), we need π_k, $p(k)$, $q(k,k')$, and the distributions $D_k(\cdot)$ for all k and all k'. Since these are unknown, they have to be estimated using the data. Here, in order to perform this estimation using simple algorithms, we consider only the available labeled nodes of the graph in the estimation. These algorithms are described below:

- Estimation of π_k: this is obtained by counting the number of labeled nodes belonging to class k divided by the total number of labeled nodes.
- Estimation of $p(k)$ and $q(k,k')$: we count all the pairs of labeled nodes belonging to class k and k'; then we count the number of these pairs that are connected in 1-hop. The estimate of $\hat{q}(k,k')$ is obtained by dividing the latter count by the former one. if these values do not depend on k and k', we also average over all the involved pairs (with $k \neq k'$ to obtain $q(k,k') = \bar{q}$ and when $k = k'$ to obtain $q(k,k) = \bar{p}$).
- Estimation of classes' statistics $D_k(\cdot)$: we assume a shape for the distribution, and this shape is dependent on a set of parameters. For instance, in the Cora dataset, the features are binary, so they are modeled as independent Bernoulli random variables; the probability associated with each feature is estimated using the proportion of non-zero elements of this feature in the labeled nodes, and Laplace smoothing. If the features are continuous-valued, we use the Gaussian distribution.

It is worth pointing out that a semi-supervised estimation approach may also be possible but this would require an iterative approach that cycles between parameter estimation and node classification. It is also worth noting that the estimation step plays the role of the learning phase in GNN-based classifiers or in the classifiers developed in [26].

The values of the hyperparameters γ_v are set to 1 by default. The optimization of these hyperparameters is addressed in Sections IV and V.

This optimization will be shown to improve classification performance. Indeed, these hyperparameters will adjust the degree to which neighbors of different orders should contribute to the classification of a node.

C. WHEN DOES GRAPH STRUCTURE NOT HELP GAB

Thanks to Eq. (14), we will be able to characterize some conditions on the graph's parameters for which the graph through the proposed GAB helps each node to improve its classification performance. Reasoning by contradiction, one can see that the classifier does not take into account the neighbors if and only if (iff) the function

$$k \mapsto \prod_{v \in \mathcal{N}_d(u)} \left(\sum_{k'=1}^{K} r^{(d)}(k,k')^{\gamma_v} D_{k'}(\boldsymbol{x}_v) \right)$$

is independent of k for any d. Indeed, if true, the a posteriori distribution to maximize depends on k only through $D_k(\boldsymbol{x}_u)$.

The above-mentioned function leads to the same output regardless of k for any feature values iff the weights $r^{(d)}(k,k')$ are identical for any k. Therefore the condition for the neighbors to be useless is

$$r^{(d)}(k,k') = t^{(d)}(k'), \forall k. \tag{15}$$

According to Eq. (12), it is easy to prove that if it is satisfied for $d = 1$, then it remains true for any d. Therefore, we need to only focus on $d = 1$. According to Eqs. (13) and (15), we obtain

$$\frac{q(k,k')\pi_{k'}}{\sum_{k''=1}^{K} q(k,k'')\pi_{k''}} = t^{(1)}(k'), \forall k. \tag{16}$$

We will now inspect some particular cases:

- In the case of constant intra-class and inter-class probabilities of connection, we obtain the following constraint for any pair (k,k') such that $k \neq k'$

  $$\pi_{k'} + (\bar{q}/\bar{p}) \sum_{\substack{k''=1 \\ k'' \neq k'}}^{K} \pi_{k''} = (\bar{p}/\bar{q})\pi_k + \sum_{\substack{k''=1 \\ k'' \neq k}}^{K} \pi_{k''}.$$

  By setting $\nu = \sum_{k=1}^{K} \pi_k$, we have

  $$\pi_{k'}(1 - \bar{q}/\bar{p}) + \pi_k(1 - \bar{p}/\bar{q}) = \nu(1 - \bar{q}/\bar{p})$$

  which implies that

  $$\underline{\pi}_{k'} + \underline{\pi}_k \frac{1 - \bar{p}/\bar{q}}{1 - \bar{q}/\bar{p}} = 1 \tag{17}$$

  where $k \mapsto \underline{\pi}_k := \pi_k/\nu$ is a probability mass function. As $(1 - \bar{p}/\bar{q})/(1 - \bar{q}/\bar{p})$ is negative, this equation does not hold except if $\bar{p} = \bar{q}$. Consequently, the neighbors are not involved in the GAB classifier when the degree of impurity is 1 since there is no community structure.
- In the case of $K = 2$, Eq. (16) implies that

  $$\frac{p(2)}{\bar{q}\pi_1 + p(2)\pi_2} = \frac{\bar{q}}{p(1)\pi_1 + q\pi_2}$$

  and

  $$\frac{\bar{q}}{\bar{q}\pi_1 + p(2)\pi_2} = \frac{p(1)}{p(1)\pi_1 + \bar{q}\pi_2},$$

  which leads to

  $$\bar{q} = \sqrt{p(1)p(2)}. \tag{18}$$

  In this setup, the neighbors are not involved in the GAB classifier when the inter-class probability of connection is the geometric mean of the intra-class probabilities of connection, and so not necessary when the degree of impurity (defined through the arithmetic mean) is equal to 1. Obviously, if $p(1) = p(2)$, we go back to the first item leading to $p = q$, i.e., a degree of impurity equal to 1. But when $p(1) \neq p(2)$, the degree of impurity leading to a graph-agnostic classifier is equal to the ratio between the geometric mean and the arithmetic mean of the intra-class probabilities of connection which is strictly smaller than 1 in general case.

III. LINK TO GRAPH NEURAL NETWORKS

A. RECAP OF GRAPH NEURAL NETWORKS

GNNs are a class of graph embedding architectures which use the graph structure in addition to node and edge features to generate a representation vector (i.e., embedding) for each node. GNNs learn node representations by aggregating the features of neighboring nodes and edges. The output of the ℓ-th layer of these GNNs is generally expressed as:

$$\boldsymbol{h}_u^{(\ell)} = \sigma^{(\ell)}(\phi^{(\ell)}(\boldsymbol{h}_u^{(\ell-1)}, \{\boldsymbol{h}_v^{(\ell-1)} : v \in \mathcal{N}_1(u)\})) \quad (19)$$

where $\boldsymbol{h}_u^{(\ell)}$ is the feature vector of node u at the ℓ-th layer initialized by $\boldsymbol{h}_u^{(0)} = \boldsymbol{x}_u$ and $\mathcal{N}_1(u)$ is the set of first-order neighbors of node u. Different GNNs use different formulations for the non-linear function $\sigma^{(\ell)}$ (called activation function) and the linear function $\phi^{(\ell)}$ [30]. Note that a first-order GNN based classifier relies on one layer or equivalently considers only the 1-hop neighborhood in the graph.

1) GRAPH CONVOLUTIONAL NEURAL NETWORK (GCN)

The convolutional propagation rule used in GCN is defined as follows

$$\phi^{(\ell)} = (\boldsymbol{W}^{(\ell)})^\top \left(\frac{\boldsymbol{h}_u^{(\ell-1)}}{d_u + 1} + \sum_{v \in \mathcal{N}_1(u)} \frac{\boldsymbol{h}_v^{(\ell-1)}}{\sqrt{(d_u+1)(d_v+1)}} \right) \quad (20)$$

where

- $\boldsymbol{W}^{(\ell)}$ is a learnable weight matrix,
- d_u is the degree of node u.

The activation function (for any layer except the last one) is a rectified linear unit (ReLU). For the last layer, we consider the softmax which for each node u outputs the probability that node u belongs to class k. Then the node u is assigned to the class with the highest probability [17].

2) GRAPH CONVOLUTION OPERATOR NETWORK (GON)

In [31] and [32], GON is defined as GCN where Eq. (20) is replaced with the following one

$$\phi_u^{(\ell)} = (\boldsymbol{W}_1^{(\ell)})^\top \boldsymbol{h}_u^{(\ell-1)} + (\boldsymbol{W}_2^{(\ell)})^\top (\sum_{v \in \mathcal{N}_1(u)} \boldsymbol{h}_v^{(\ell-1)}). \quad (21)$$

Unlike GCN, GON computes a transformation matrix of the central node that is different from the transformation of its neighbors.

3) GRAPH ISOMORPHISM NETWORK (GIN)

In [33], GIN is defined as GCN or GON where Eqs.(20)-(21) are replaced with the following one

$$\phi_u^{(\ell)} = (\boldsymbol{W}^{(\ell)})^\top \left((1+\alpha)\boldsymbol{h}_u^{(\ell-1)} + \sum_{v \in \mathcal{N}_1(u)} \boldsymbol{h}_v^{(\ell-1)} \right). \quad (22)$$

where α is a positive hyper-parameter. GIN thus attributes a different learnable weight to the central node (through α) when combining information from its neighbors.

4) GRAPH ATTENTION NETWORK (GAT)

In [34], GAT is defined as GCN, GON or GIN but with the following layer link

$$\phi_u^{(\ell)} = \sum_{v \in \mathcal{N}_1(u) \cup \{u\}} \alpha_{u,v}^{(\ell)} (\boldsymbol{W}^{(\ell)})^\top \boldsymbol{h}_v^{(\ell-1)}, \quad (23)$$

where $\alpha_{u,v}^{(\ell)}$ are normalized attention coefficients computed by an attention mechanism as follows:

$$\alpha_{u,v}^{(\ell)} = \frac{e^{\varsigma(\boldsymbol{w}^{(\ell)}[(\boldsymbol{W}^{(\ell)})^\top \boldsymbol{h}_u^{(\ell-1)} \| (\boldsymbol{W}^{(\ell)})^\top \boldsymbol{h}_v^{(\ell-1)}])}}{\sum_{k \in \mathcal{N}_1(u)} e^{\varsigma(\boldsymbol{w}^{(\ell)}[(\boldsymbol{W}^{(\ell)})^\top \boldsymbol{h}_u^{(\ell-1)} \| (\boldsymbol{W}^{(\ell)})^\top \boldsymbol{h}_k^{(\ell-1)}])}}. \quad (24)$$

with ς the leaky ReLu function, the weighting row vector $\boldsymbol{w}^{(\ell)} \in \mathbb{R}^{2H}$, where H is the size of the hidden layer and $\|$ corresponds to column concatenation.

B. RELATIONSHIP WITH A GNN BASED CLASSIFIER

In this Section, we compare the shape of the proposed GAB and the GNN. In GNN, there is one activation function between each layer which implies that the multi-hop information has undergone several non-linear functions before arriving at the node of interest. In GAB, the multi-hop mixture is done prior to making the final decision and do not follow a successive concatenation of linear combination and activation function. All the operations are intermixed, therefore GNN and GAB are very different in terms of structure, except for the one-layer/one-hop case. We therefore focus here on the relation between the first-order GAB classifier and first-order GNN-based classifier. For doing that, we consider a binary classification problem (i.e $K = 2$). According to Eq. (14), we assign node u to class 1 if:

$$P_u(1) \geq P_u(2),$$

which implies that :

$$\frac{\pi_1 D_1(\boldsymbol{x}_u) \prod_{v \in \mathcal{N}_1(u)} (r(1,1)D_1(\boldsymbol{x}_v) + r(1,2)D_2(\boldsymbol{x}_v))}{\pi_2 D_2(\boldsymbol{x}_u) \prod_{v \in \mathcal{N}_1(u)} (r(2,1)D_1(\boldsymbol{x}_v) + r(2,2)D_2(\boldsymbol{x}_v))} > 1.$$

By setting

$$S(x) = \frac{D_1(x)}{D_2(x)}$$

and taking the log, we obtain the following test T (where $T > 0$ means "decide class 1"):

$$T = \log\left(\frac{\pi_1}{\pi_2}\right) + \log(S(\boldsymbol{x}_u))$$

$$+\sum_{v\in\mathcal{N}(u)}\log\left(\frac{r(1,2)+r(1,1)S(\boldsymbol{x}_v)}{r(2,2)+r(2,1)S(\boldsymbol{x}_v)}\right).$$

As $q(1,2)=q(2,1)$, two classes in the system lead to $\bar{q}=q(1,2)=q(2,1)$. Consequently,

$$\frac{r(1,2)+r(1,1)S(\boldsymbol{x}_v)}{r(2,2)+r(2,1)S(\boldsymbol{x}_v)}=\frac{\frac{\bar{q}\pi_2+p(1)\pi_1S(\boldsymbol{x}_v)}{p(1)\pi_1+\bar{q}\pi_2}}{\frac{p(2)\pi_2+\bar{q}\pi_1S(\boldsymbol{x}_v)}{\bar{q}\pi_1+p(2)\pi_2}}.$$

Therefore, the test T can be split into three parts:

$$\begin{aligned}T=&\log\left(\frac{\pi_1}{\pi_2}\right)+\log\left(\frac{\bar{q}\pi_1+p(2)\pi_2}{p(1)\pi_1+q\pi_2}\right)\\&+\log\left(S(\boldsymbol{x}_u)\right)\\&+\sum_{v\in\mathcal{N}(u)}\log\left(\frac{\bar{q}\pi_2+p(1)\pi_1S(\boldsymbol{x}_v)}{p(2)\pi_2+\bar{q}\pi_1S(\boldsymbol{x}_v)}\right).\end{aligned}$$

The first part corresponds to a constant term and so is connected to the threshold. The second part is the contribution of the node of interest. The third part, which is the most interesting, corresponds to the contribution of the neighbors in the test. Clearly, in general, this term is not linear with respect to the nodes' features and so the test cannot be seen as a one-layer GNN.

If, in addition, $p(1)=p(2)=\bar{p}$, the test can be written easily with respect to the DoI as follows

$$\begin{aligned}T=&\log\left(\frac{\pi_1}{\pi_2}\right)+\log\left(\frac{\mathrm{DoI}.\pi_1+\pi_2}{\pi_1+\mathrm{DoI}.\pi_2}\right)\\&+\log\left(S(\boldsymbol{x}_u)\right)\\&+\sum_{v\in\mathcal{N}(u)}\log\left(\frac{\mathrm{DoI}.\pi_2+\pi_1S(\boldsymbol{x}_v)}{\pi_2+\mathrm{DoI}.\pi_1S(\boldsymbol{x}_v)}\right).\end{aligned}$$

For instance, we remark that if the DoI is much larger than the pdf ratio between the classes (which may be roughly related to the Kullback-Leibler divergence), then the third term is almost independent of the nodes's feature and the information provided by the graph is not used since it is not reliable.

Consider now that the graph is pure (i.e., $r(1,2)=r(2,1)=0$ or equivalently $q=0$), we obtain

$$\begin{aligned}T=&\log(\frac{\pi_1}{\pi_2})+|\mathcal{N}(u)|\log\left(\frac{r(1,1)}{r(2,2)}\right)\\&+\sum_{v\in\mathcal{N}(u)\cup\{u\}}\log(S(\boldsymbol{x}_v)),\end{aligned}\tag{25}$$

where we consider that $p(1)$ may be different from $p(2)$. Once again the proposed classifier does not boil down to a one-layer GNN. Actually, it can be a GNN if the term $\log_2(S(\boldsymbol{x}_v))$ is a linear combination of $\boldsymbol{x}_v$. This can be achieved if the function $S(\boldsymbol{x}_v)$ is at least a power-function of $\boldsymbol{x}_v$, such as the Gaussian function or Binomial function.

Let us first consider the Gaussian case, i.e. $\boldsymbol{x}_v\sim\mathcal{N}(\boldsymbol{\mu}(v),\boldsymbol{\Sigma})$ where $\boldsymbol{\mu}(v)$ is either $\boldsymbol{\mu}_1$ (if class 1) or $\boldsymbol{\mu}_2$ (if class 2) and the correlation matrix is independent of the class (if not, a second-order polynomial occurs and the GAB is different from a GNN). According to Eq. (25), the first-order GAB test is equal to

$$T=\omega_0+\left(\sum_{v\in\mathcal{N}(u)\cup\{u\}}\boldsymbol{\omega}_1^\top\boldsymbol{x}_v\right),\tag{26}$$

where

$$\begin{aligned}\omega_0\;=&\;\log\left(\frac{\pi_2}{\pi_1}\right)\\&+(|\mathcal{N}(u)|+1)(\boldsymbol{\mu}_2^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_2-\boldsymbol{\mu}_1^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1),\\\boldsymbol{\omega}_1^\top\;:=&\;[\omega_{1,1},\cdots,\omega_{1,F}]=(\boldsymbol{\mu}_2-\boldsymbol{\mu}_1)^\top\boldsymbol{\Sigma}^{-1}.\end{aligned}\tag{27}$$

Consequently this test is a GNN-based test.

Let us now consider the Binomial case. We assume that features $x_{v,f}$ are independent binary random variables with probabilities $P(x_{v,f}=1|y_v=1)=\alpha_f^{(1)}$ and $P(x_{v,f}=1|y_v=2)=\alpha_f^{(2)}$. Eq. 25 can be written as Eq. (26) with

$$\begin{aligned}\omega_0&=\log\left(\frac{\pi_2}{\pi_1}\right)+F\sum_{f=1}^{F}\log\left(\frac{1-\alpha_f^{(2)}}{1-\alpha_f^{(1)}}\right),\\\omega_{1,f}&=-\log\left(\frac{\alpha_f^{(1)}}{\alpha_f^{(2)}}\frac{1-\alpha_f^{(2)}}{1-\alpha_f^{(1)}}\right).\end{aligned}$$

Consequently this test is a GNN-based test as well.

IV. DISCUSSION ON COMPLEXITY ISSUES

In this section, we compare the different versions of the proposed GAB and the GAT in terms of parameters/weights to tune and the number of flops for doing this tuning during the training phase. We will consider only the two-hops case for the GAB and the two-layers case for the GAT.

First of all, we evaluate the number of parameters to be tuned. Let H and $d_{\max}$ be respectively the size of the hidden layer and the maximum degree of the considered graph.

For a GAB classifier, we need to estimate

- K^2 parameters for the transition matrix $\mathbf{R}$ through the terms $\{p(k)\}_k$ and $\{q(k,k')\}_{k\neq k'}$;
- the parameters of the class' distributions (obviously, this value depends on the shape of the assumed distributions):
  - KF parameters when each class of each feature is Bernoulli-distributed
  - $(KF+KF^2)$ parameters when each class is arbitrary Gaussian-distributed. The number of parameters can be reduced if the Gaussian distribution per class is structured. For instance, if the covariance is independent of the class, we only have $(KF+F^2)$ parameters. If in addition, the covariance matrix is diagonal or is assumed as a diagonal matrix for the sake of simplicity, we have $(KF+F)$ parameters;
  - K parameters for the priors $\{\pi_k\}_k$.

When Cora or PubMed dataset are considered (see Table 5 for more details), the distribution is assumed to be Bernoulli

TABLE 2. Number of parameters or weights to be tuned/learnt.

Dataset	N_p (GAB)	N_w (GAT)
Cora	10,087	369,066
PubMed	1,512	129,286

in our numerical evaluations (even if not), which implies that the total number of parameters to estimate, N_p is given by

$$N_p = K(K + F + 1). \tag{28}$$

For GAT, we need to tune/learn the weights and the attention parameters of the neural network. As only two layers are considered we obtain FH weights for the first layer, HK weights for the second layer, and $2H$ (resp. $2F$) weights for the attention mechanism for the first layer (resp. the second one). As a consequence, the number of weights to tune, denoted by N_w is as follows:

$$N_w = FH + HK + 2(H + K). \tag{29}$$

According to the number of classes and features given in Table 5 and by assuming a hidden layer of size $H = 256$, we have the following values N_p and N_w for the Cora and PubMed dataset in Table 2.

We observe that the number of parameters for GAB is much smaller (by more than an order of magnitude) than for GAT. In addition, the parameters in GAB are interpretable. Moreover, since for GAT, parameters are learnt using an iterative process, the computational complexity in terms of the number of epochs may significantly vary with the chosen optimization algorithm.

We here consider two variants for the GAB. The first one (denoted by GAB2) corresponds to the case where $\gamma_1 = \gamma_2 = 1$ while the second one (denoted by GAB2γ) is optimized with respect to the pair $\boldsymbol{\gamma} = (\gamma_1, \gamma_2)$. Let N_t, N_g, and N_v be the number of training nodes, the number of tested pairs $\boldsymbol{\gamma}$, and the number of nodes for validation respectively. To estimate the N_p parameters during the training phase, we need N_pN_t flops since each parameter corresponds to counts or sums over each training node. To select the best pair of hyperparameters $\boldsymbol{\gamma}$, we compute our GAB on all validation and test nodes. A GAB evaluation leads to at most $K^2D_{\max}^2F$ flops by looking at Eq. (14). Indeed we consider that to compute each test for one class, we need $d_{\max}^2KF$ flops where F corresponds to the flops required to compute $D_k(\cdot)$ as it is the case for the Bernoulli case or the diagonal correlation matrix Gaussian case. Finally, we have that

$$N_{\text{flop}|\text{GAB2}\gamma} = N_pN_t + N_gN_vK^2d_{\max}^2F. \tag{30}$$

When we only consider GAB2, the second term in the Right Hand Side (RHS) has to be omitted. Note that the 2D grid for the hyperparameter $\boldsymbol{\gamma}$ is $[0 : 0.1 : 1]^2$ leading to $N_g = 121$.

For GAT2, we just consider the number of flops required for learning the weights (so the hyperparameters such as the learning rate are assumed to be obtained for free). To learn the weights, we apply a gradient-descent like algorithm with N_e epochs. Hereafter, we consider only the neural network's weights which are dominant. So the weights related to the attention mechanism are ignored.

TABLE 3. Number of flops for the training phase.

Dataset	GAB2	GAB2γ	GAT2
Cora-30%	1.01×10^7	1.99×10^{11}	1.14×10^{19}
Cora-5%	1.41×10^6	1.99×10^{11}	1.60×10^{18}
PubMed-5%	1.51×10^6	9.31×10^{10}	1.40×10^{18}

TABLE 4. Number of flops for the inference phase.

Dataset	GAB2/GAB2γ	GAT2
Cora	2.3×10^9	1.60×10^{12}
PubMed	1.3×10^8	2.80×10^{11}

We consider one epoch. For the first layer (resp. second layer), we have FH (resp. HK) weights to update and so FH (resp. HK) sums have to be computed once the gradient is known. For estimating the gradient, we average over N_t nodes, over the sum of the neighbors (at most $d_{\max}$) and a matrix computation of size HF (resp. KH) with the current feature of size F (resp. H). Consequently, we have

$$N_{\text{flop}|\text{GAT2}} = N_eFHN_td_{\max}FH + N_eHKN_td_{\max}HK. \tag{31}$$

In Table 3, we report the rough number of flops for Cora (with a supervision of 30% and 5%) and PubMed (with a supervision of 5%) with three classifiers. We set $N_e = 500$, $N_t = 140$ (Cora-5%) or $N_t = 1000$ (Cora-30% and PubMed-5%). Moreover $d_{\max} = 168$ for Cora and $d_{\max} = 171$ for PubMed.

In the inference phase, applying our classifier (GAB2) leads to $K^2d_{\max}^2F$ flops (which corresponds to the number of flops in the second part of the RHS in Eq. (30) without N_g and N_v). For GAT2, we have $FHd_{\max}FH + HKd_{\max}HK$ flops (actually, we apply Eq. (31) by removing N_e and N_t). In Table 4, we report the rough number of flops during the inference phase for Cora and PubMed.

We remark that the numbers of flops for our GAB classifiers are much smaller (by many orders of magnitude) than for the GAT in both training and inference phases. Consequently, the structure imposed by the derivations of the GAB enable us to have interpretability and less complexity than the black box GAT.

V. NUMERICAL RESULTS

In this section, we conduct experiments with two kinds of datasets: the synthetic ones where we especially analyze the robustness to the degree of impurity; and some real benchmark ones where we compare our classifier to many other approaches based on GNN. The performance of our classifier (and the ordering with respect to standard GNN based approaches) depend on the dataset and the level of supervision.

A. SYNTHETIC DATASETS

For the sake of simplicity, we consider two classes, i.e., $K = 2$. Each class is associated with a different statistical distribution of the node's feature vector $\boldsymbol{x}_v$. We assume here

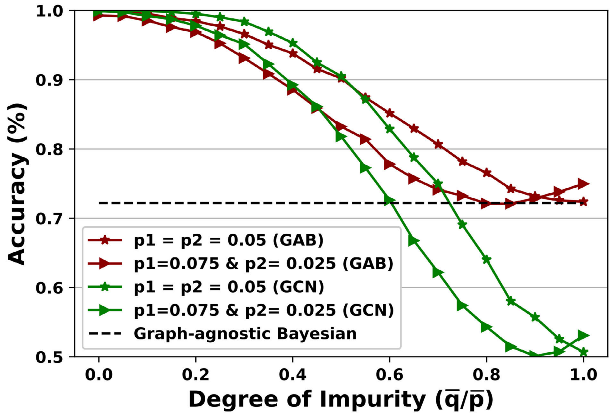

FIGURE 2. GAB and GNN performance on synthetic datasets versus DoI.

that the two distributions are multivariate Gaussian, i.e. $\boldsymbol{x}_v$ follows $\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ in class 1 and $\mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$ in class 2, with unknown mean and covariance. We assume that the features are uncorrelated in the two classes, i.e. $\boldsymbol{\Sigma}_1 = \text{diag}(\boldsymbol{\sigma}_1^2)$ and $\boldsymbol{\Sigma}_2 = \text{diag}(\boldsymbol{\sigma}_2^2)$, where $\boldsymbol{\sigma}_1^2$ and $\boldsymbol{\sigma}_2^2$ are $(F \times 1)$ vectors. To average over different configurations, different distributions are generated by drawing $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ randomly from $\mathcal{U}([0, 1]^F)$ and drawing $\boldsymbol{\sigma}_1$ and $\boldsymbol{\sigma}_2$ randomly from $\mathcal{U}([0.5, 3.5]^F)$. Moreover, the links are generated randomly using different values of the probability of intra-connectivity, $\bar{p}$, and the probability of inter-connectivity, $\bar{q}$ where $\bar{q} \leq \bar{p}$.

In each experimental setting, we evaluate the average node classification accuracy using a Monte Carlo simulation with $1,000$ runs. We set the number of nodes to $N = 5,000$ and the number of features to $F = 500$. For each run, we train each classifier on 500 (already-labeled) nodes and test its accuracy on $2,000$ nodes. We use the remaining nodes for validation.

In Fig. 2, we plot the classification accuracy versus the Degree of Impurity $(\bar{q}/\bar{p})$ for one-order GAB and one-layer GCN. Both approaches perform well when DoI is small. Their performance decreases with increasing DoI. We observe, however, that the GAB is more robust to DoI than the GCN especially beyond a DoI of one half. As expected, the GAB is graph-agnostic when Eq. (18) is satisfied. For instance, when $p(1) = p(2) = 0.05$ then the agnosticity occurs at $\bar{q}/\bar{p} = 1$ while when $p(1) = 0.025$ and $p(2) = 0.075$, the agnosticity is reached at $\bar{q}/\bar{p} = 0.866$. We remark that GNN has similar behavior with respect to the usefulness of the information coming from the graph.

B. REAL DATASETS

Unless otherwise stated, we use the attributed graphs described in Table 5, and we use the training/validation/testing split equal to 30%, 20%, 50%, respectively. To implement the GNN based classifiers, we use Pytorch where we initialize all the GNN weights by Glorot initialization, and we train them to minimize the cross entropy loss using the Adam optimizer with an initial learning rate of 0.005.

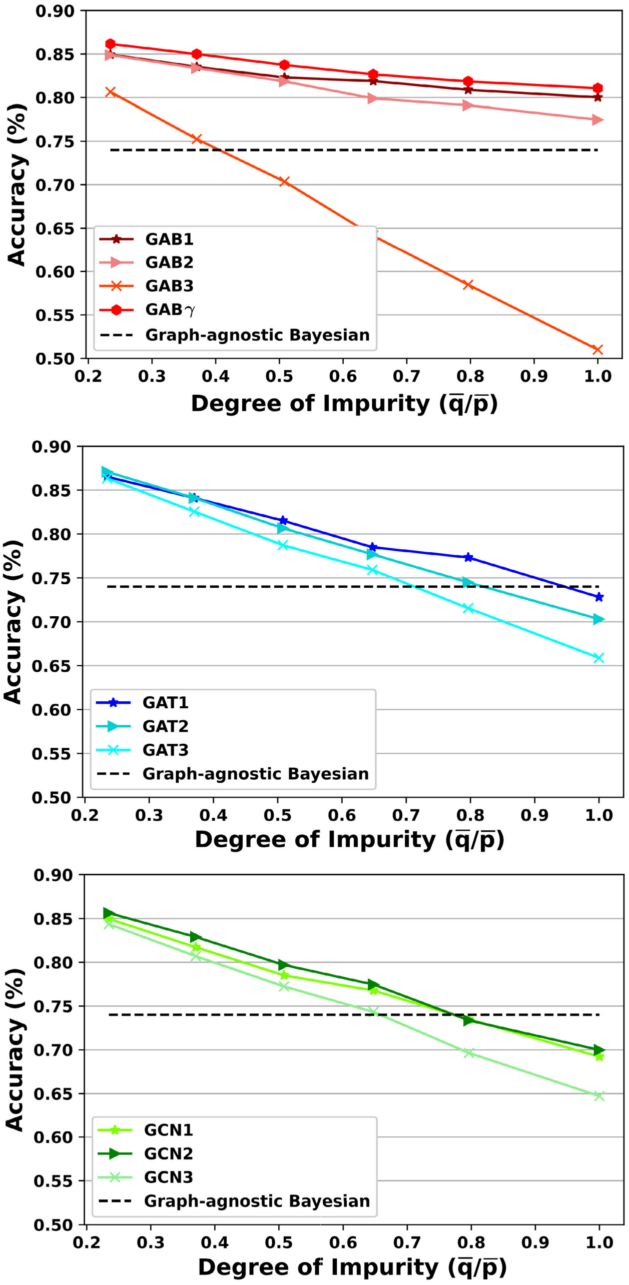

FIGURE 3. Accuracy performance with added noisy edges in Cora.

In Fig. 3, we plot the accuracy versus the DoI: on the top, the GAB at several orders as well as the best combination of powers γ_d in Eq. (14). Here, the training phase enables us to estimate the parameters of the GAB except the γ_d. The powers γ_d are optimized with a grid search approach during the validation phase, on the middle panel, the GAT with several layers, and on the bottom panel the GCN at several layers. The dataset is here always Cora which implies that the distributions are Bernoulli. The x-axis starts with the real value of the DoI and we add links between previously unconnected nodes that belong to different classes with the goal of gradually varying the DoI to 1. The above mentioned

TABLE 5. Description of the real datasets.

Dataset	Classes	Nodes	Edges	Features	Avg.Degree
Cora [24]	7	2,708	5,429	1,433	3.90
CiteSeer [24]	6	3,327	4,732	3,707	2.74
PubMed [24]	3	19,717	44,338	500	4.50
CS [35]	15	18,333	163,788	6,805	8.93
Physics [35]	5	34,493	495,924	8,415	14.38
Sexual [36], [37]	2	1,888	2,096	20	2.21

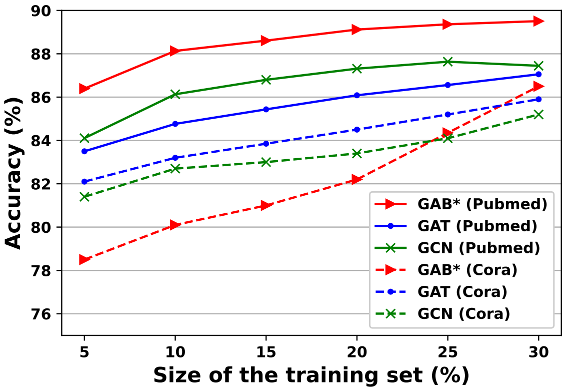

FIGURE 4. GAB and GNN performance on Pubmed versus the size of the training set.

classifiers are trained for each "impure" graph. We remark that the information on the graph is less accurate for any approach when we consider more hops; actually, the confidence on the "far" data is smaller. Nevertheless by averaging properly the hops as in GABγ, the performance are improved slightly. In any case, GABγ is better then the best other approaches since it enables to better adapt to the impurity. Moreover GABγ outperforms the graph-agnostic classifier; so it always manages to take benefit of the graph.

In Table 6, we compare our approaches GABγ and GAB$\star$ up to five hops (in GAB$\star$, we approximate the unknown distributions by two-layer NNs which are learnt during the training phase) with the methods presented and analyzed in [26]. We copy-paste the values given in [26] where the best number of layers and the best version of each approach have been selected. We select the best version of GAB in the sense that we optimized the hyperparameters $\boldsymbol{\gamma}$ by allowing at most the fifth-order case. In GABγ, the shape of the distributions is a priori given. For instance, for PubMed, we considered a Bernoulli one while it is inaccurate. We plot the accuracy rate and the ranking (in brackets) for a 30%-supervised graph. For computing the average accuracy and the ranking (for all the considered datasets) for our GAB approach, we select the best one. We remark that the performance of our approach is close to GBPN and GAT which are the best ones in the literature. When GABγ is bad, it corresponds to the case PubMed where the used distribution does not fit well with the true one. So improvement can be done by choosing a better approximation. In addition to the good performance of our proposed approach, we have interpretability of our algorithms since we wrote them in closed-form and we are able to understand the meaning of each element.

In Fig. 4, we plot the accuracy rate versus the training size (in percentage) for our approach (GAB$\star$) and the GCN and GAT with two-layers/hops. In solid line, we consider PubMed dataset and in dash line, we consider Cora dataset. We observe that for PubMed, our approach outperforms the state of the art regardless of the training size. Actually we should note than as PubMed is a large dataset, a small portion of training still leads to a large amount of data to estimate the statistics required for GAB. In contrast, for Cora, our approach outperforms GCN and GAT only when the training is large enough (and as Cora is a small dataset, large enough means also when the training set in percentage is large enough).

According to all the previous experiments, we remark that our GAB approach is more robust to the degree of impurity and its performance is close to or better than the tested GNN approaches on a number of benchmark graphs.

VI. CONCLUSION

In this work, we tackled the problem of node classification on attributed graphs. We start from the observation that GNN based models' performance depend on the graph topology, and more specifically on the degree of impurity of the graph. We adopted a different path and used the Bayesian theorem to propose a new graph-assisted Bayesian-based node classifier. This classifier is able to take into account the degree of impurity of the graph. It is also shown to significantly outperform some GNN-based classifiers, in addition to providing more interpretability and requiring lower computational complexity (if the model of the nodes' distributions is well approximated in closed-form).

APPENDIX A
DERIVATIONS FOR EQ. (3)

We first focus on the term $Q_u(k)$. We get

$$\begin{aligned}Q_u(k) &= P(\mathcal{X}_u|y_u=k,\mathcal{I}_\mathcal{G})\\ &= \sum_{\{k_v\}_{v\in\mathcal{V}_u}} P(\mathcal{X}_u|y_u=k,\{y_v=k_v\}_{v\in\mathcal{V}_u},\mathcal{I}_\mathcal{G})\\ &\quad \times p(\{y_v=k_v\}_{v\in\mathcal{V}_u}|y_u=k,\mathcal{I}_\mathcal{G})\end{aligned}$$

As the classes of the neighbors are given, the information on the graph becomes redundant and so useless. Therefore $\mathcal{I}_\mathcal{G}$ can be removed from the first term.

$$\begin{aligned}Q_u(k) = &\sum_{\{k_v\}_{v\in\mathcal{V}_u}} P(\mathcal{X}_u|y_u=k,\{y_v=k_v\}_{v\in\mathcal{V}_u})\\ &\times p(\{y_v=k_v\}_{v\in\mathcal{V}_u}|y_u=k,\mathcal{I}_\mathcal{G})\end{aligned}$$

Given the classes, the samples of each node are run independently, so we get

$$\begin{aligned}Q_u(k) &= \sum_{\{k_v\}_{v\in\mathcal{V}_u}} \prod_{v'\in\mathcal{V}_u} P(\boldsymbol{x}_{v'}|y_u=k,\\ &\quad \{y_v=k_v\}_{v\in\mathcal{V}_u})\\ &\quad \times p(\{y_v=k_v\}_{v\in\mathcal{V}_u}|y_u=k,\mathcal{I}_\mathcal{G})\\ &= \sum_{\{k_v\}_{v\in\mathcal{V}_u}} P(\boldsymbol{x}_u|y_u=k)\end{aligned}$$

TABLE 6. 30%-supervised node classification accuracy (%).

Dataset	MLP	GCN	SAGE	GAT	GMNN	DeeperGCN	GBPN	GABγ	GAB$\star$
Cora	72.1±1.3	87.1±0.7	86.9±0.8	87.1±0.9	86.4±0.9	**87.2±0.9**	86.4±0.9	86.9±0.8	86.3±1.1
CiteSeer	71.2±0.9	73.5±1.0	73.2±1.0	73.1±1.2	72.9±1.2	73.9±0.8	74.8±0.8	**75.2±0.9**	74.7±0.8
PubMed	86.5±0.2	87.1±0.3	87.8±0.4	88.1±.3	86.7±0.2	84.7±0.3	88.5±0.3	86.4±0.2	**89.5±0.3**
CS	94.2±0.2	93.2±0.2	93.7±0.2	94.0±0.3	93.3±0.3	94.9±0.2	**95.5±0.3**	94.5±0.3	95.2±0.4
Physics	95.8±0.1	96.1±0.1	96.3±0.1	96.3±0.1	96.1±0.1	96.7±0.1	**96.9±0.1**	96.4±0.1	96.7±0.1
Sexual	74.5±1.4	83.9±1.2	93.3±0.8	93.6±0.9	77.0±1.7	65.0±1.1	**97.4±0.4**	96.5±0.7	97.1±0.5

$$
\begin{aligned}
&\times \prod_{v' \in \mathcal{V}_u} P(\boldsymbol{x}_{v'} | y_{v'} = k_{v'}) \\
&\times p(\{y_v = k_v\}_{v \in \mathcal{V}_u} | y_u = k, \mathcal{I}_{\mathcal{G}}) \\
= D_k(\boldsymbol{x}_u) &\sum_{\{k_v\}_{v \in \mathcal{V}_u}} \prod_{v' \in \mathcal{V}_u \setminus \{u\}} D_{k_{v'}}(\boldsymbol{x}_{v'}) \\
&\times p(\{y_v = k_v\}_{v \in \mathcal{V}_u} | y_u = k, \mathcal{I}_{\mathcal{G}}).
\end{aligned}
$$

which concludes the derivations by doing a re-ordering.

APPENDIX B DERIVATIONS FOR EQ.(12)

- Let us start with $d = 2$:

$$
\begin{aligned}
r^{(2)}(k, k') &= P(y_v = k' | y_u = k, \mathcal{C}_2) \\
&= \sum_{k''=1}^{K} P(y_v = k' | y_u = k, y_w = k'', \mathcal{C}_2) \\
&\quad \times P(y_w = k'' | y_u = k, \mathcal{C}_1)
\end{aligned}
$$

As the class of w is known, the information on the class of u becomes useless and only the fact that v and w are 1-hop neighbor remains important. Therefore, we obtain

$$
\begin{aligned}
r^{(2)}(k, k') = &\sum_{k''=1}^{K} P(y_v = k' | y_w = k'', \mathcal{C}_1) \\
&\times r^{(1)}(k, k'') \qquad (32) \\
= &\sum_{k''=1}^{K} r^{(1)}(k'', k').r^{(1)}(k, k''). \qquad (33)
\end{aligned}
$$

where w is the node connecting u and v. Such a node w exists since u and v are 2-hop connected. According to Eq. (33), we have

$$
\mathbf{R}^{(2)} = \mathbf{R}^2.
$$

- For any d, we have,

$$
\begin{aligned}
r^{(d)}(k, k') &= P(y_v = k' | y_u = k, \mathcal{C}_d) \\
&= \sum_{k''=1}^{K} P(y_v = k' | y_u = k, y_w = k'', \mathcal{C}_1) \\
&\quad \times P(y_w = k'' | y_u = k, \mathcal{C}_{d-1}) \\
&= \sum_{k''=1}^{K} P(y_v = k' | y_w = k'', \mathcal{C}_1) \\
&\quad \times r^{(d-1)}(k, k'') \\
&= \sum_{k''=1}^{K} r^{(1)}(k'', k').r^{(d-1)}(k, k'').
\end{aligned}
$$

Therefore

$$
\mathbf{R}^{(d)} = \mathbf{R}^{(d-1)}\mathbf{R}.
$$

- Finally, by induction, we conclude the derivations.

APPENDIX C ADDITIONAL DETAILS ON IMPLEMENTATIONS

Algorithm 1 The Inference Algorithm for GAB

```
Input: Graph topology $\mathcal{G} = (\mathcal{V}, \mathcal{E})$; nodes' features $\{\boldsymbol{x}_u\}_{u \in \mathcal{V}}$;
  labels of the set of labeled nodes $\{y_u\}_{u \in \mathcal{L}}$; a feature based
  classifier $g$; prior probabilities $\pi = (\pi_1, \ldots, \pi_K)$, transi-
  tion matrix $\boldsymbol{R}$.
Output: prediction for each unlabeled node $\{y_u\}_{u \in \mathcal{U}}$   ▷
  ▷ Step1: Initialize predictions
  for $u \in \mathcal{V}$ do
      for $k \in (1, \ldots, K)$ do
          if $u \in \mathcal{L}$ then
              $D_u(k) \leftarrow \mathbb{1}_{(y_u = k)}$
          else
              $D_u(k) \leftarrow g(\boldsymbol{x_u}; k)$
          end if
      end for
  end for
  ▷ Step2: Update predictions
  for $u \in \mathcal{U}$ do
      for $k \in (1, \ldots, K)$ do
          $P_u(k) \leftarrow Q_u(k)\pi_k$   ▷ $Q_u(k)$ is defined in Eq. (11)
      end for
      $\hat{y}_u \leftarrow argmax_k P_u(k)$
  end for
```

We recall that $\mathcal{V} = \mathcal{L} \cup \mathcal{U}$ where $\mathcal{L}$ and $\mathcal{U}$ are respectively the sets of labeled and unlabeled nodes. The feature based classifier g is obtained either by training a multi-layer perceptron or by estimating parameters of the generating distributions of nodes' features. $g(\boldsymbol{x_u}; k)$ returns the probability that node u belongs to class k based on its intrinsic features. Prior probabilities π and transition matrix $\boldsymbol{R}$ are estimated as explained in section II-B.

REFERENCES

[1] S. P. Borgatti, M. G. Everett, and J. C. Johnson, *Analyzing Social Networks*. London, U.K.: Sage, 2018.

[2] A. Khazane, J. Rider, M. Serpe, A. Gogoglou, K. Hines, C. B. Bruss, and R. Serpe, "DeepTrax: Embedding graphs of financial transactions," in *Proc. 18th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2019, pp. 126–133.

[3] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia, "Graph networks as learnable physics engines for inference and control," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4470–4479.

[4] Y. Li, R. Shafipour, G. Mateos, and Z. Zhang, "Supervised graph representation learning for modeling the relationship between structural and functional brain connectivity," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2020, pp. 9065–9069.

[5] X. Li, H. Chen, J. Li, and Z. Zhang, "Gene function prediction with gene interaction networks: A context graph kernel approach," *IEEE Trans. Inf. Technol. Biomed.*, vol. 14, no. 1, pp. 119–128, Jan. 2010.

[6] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 1–8.

[7] N. I. Kajla, M. M. S. Missen, M. M. Luqman, and M. Coustaty, "Graph neural networks using local descriptions in attributed graphs: An application to symbol recognition and hand written character recognition," *IEEE Access*, vol. 9, pp. 99103–99111, 2021.

[8] T. Gaudelet, B. Day, A. R. Jamasb, J. Soman, C. Regep, G. Liu, J. B. R. Hayter, R. Vickers, C. Roberts, J. Tang, D. Roblin, T. L. Blundell, M. M. Bronstein, and J. P. Taylor-King, "Utilizing graph machine learning within drug discovery and development," *Briefings Bioinf.*, vol. 22, no. 6, pp. 1–22, Nov. 2021.

[9] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 5165–5175.

[10] H. Hafidi, M. Ghogho, P. Ciblat, and A. Swami, "GraphCL: Contrastive self-supervised learning of graph representations," 2020, *arXiv:2007.08025*.

[11] H. Hafidi, M. Ghogho, P. Ciblat, and A. Swami, "Negative sampling strategies for contrastive self-supervised learning of graph representations," *Signal Process.*, vol. 190, Jan. 2022, Art. no. 108310.

[12] M. McPherson, L. Smith-Lovin, and J. M. Cook, "Birds of a feather: Homophily in social networks," *Annu. Rev. Sociol.*, vol. 27, no. 1, pp. 415–444, 2001.

[13] E. Rahm and A. Thor, "Citation analysis of database publications," *ACM SIGMOD Rec.*, vol. 34, no. 4, pp. 48–53, Dec. 2005.

[14] X. Zhu, Z. Ghahramani, and J. D. Lafferty, "Semi-supervised learning using Gaussian fields and harmonic functions," in *Proc. Int. Conf. Mach. Learn.*, 2003, pp. 912–919.

[15] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," 2015, *arXiv:1506.05163*.

[16] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. NIPS*, 2016, pp. 3844–3852.

[17] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.

[18] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–12. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ

[19] K. K. Thekumparampil, C. Wang, S. Oh, and L.-J. Li, "Attention-based graph neural network for semi-supervised learning," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–15.

[20] S. Abu-El-Haija, B. Perozzi, A. Kapoor, N. Alipourfard, K. Lerman, H. Harutyunyan, G. V. Steeg, and A. Galstyan, "MixHop: Higher-order graph convolutional architectures via sparsified neighborhood mixing," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 21–29.

[21] J. Gasteiger, A. Bojchevski, and S. Gunnemann, "Predict then propagate: Graph neural networks meet personalized PageRank," 2018, *arXiv:1810.05997*.

[22] B. Karrer and M. E. J. Newman, "Stochastic blockmodels and community structure in networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 83, no. 1, Jan. 2011, Art. no. 016107.

[23] S. Chakrabarti, B. Dom, and P. Indyk, "Enhanced hypertext categorization using hyperlinks," in *Proc. ACM SIGMOD*, Seattle, WA, USA, vol. 27, Jun. 1998, pp. 307–318.

[24] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI Mag.*, vol. 29, no. 3, p. 93, 2008.

[25] K. Murphy, Y. Weiss, and M. I. Jordan, "Loopy belief propagation for approximate inference: An empirical study," 2013, *arXiv:1301.6725*.

[26] J. Jia, C. Baykal, V. K. Potluru, and A. R. Benson, "Graph belief propagation networks," 2021, *arXiv:2106.03033*.

[27] H. Hafidi, M. Ghogho, P. Ciblat, and A. Swami, "Bayesian node classification for noisy graphs," in *Proc. IEEE Stat. Signal Process. Workshop (SSP)*, Jul. 2021, pp. 246–250.

[28] E. Abbe, "Community detection and stochastic block models: Recent developments," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6446–6531, Jan. 2017.

[29] T. P. Peixoto, "Bayesian stochastic blockmodeling," in *Advances in Network Clustering and Blockmodeling*. Chichester, U.K.: Wiley, 2019, pp. 289–332.

[30] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1024–1034.

[31] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1993–2001.

[32] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and Leman go neural: Higher-order graph neural networks," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 4602–4609.

[33] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–17.

[34] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," 2017, *arXiv:1710.10903*.

[35] O. Shchur, M. Mumme, A. Bojchevski, and S. Gunnemann, "Pitfalls of graph neural network evaluation," 2018, *arXiv:1811.05868*.

[36] M. Morris and R. Rothenberg, "HIV transmission network metastudy project: An archive of data from eight network studies, 1988–2001," Inter-Univ. Consortium Political Social Res., Aug. 2011, doi: 10.3886/ICPSR22140.v1.

[37] J. Jia and A. R. Benson, "Residual correlation in graph neural network regression," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2020, pp. 588–598.

HAKIM HAFIDI (Student Member, IEEE) received the master's degree in big data from the International University of Rabat, Morocco, in 2018. He is currently pursuing the joint Ph.D. degree with the International University of Rabat and the Polytechnic Institute of Paris, France. His research interests include deep learning, signal processing, and machine learning for graphs.

PHILIPPE CIBLAT (Senior Member, IEEE) was born in Paris, France, in 1973. He received the Engineering degree from Telecom Paris (also known as ENST/Telecom ParisTech), the M.Sc. and D.E.A. (French) degrees in automatic control and signal processing from University Paris-Saclay, France, both in 1996, and the Ph.D. and H.D.R. degrees from University Gustave Eiffel, Marne-la-Vallée, France, in 2000 and 2007, respectively. In 2001, he was a Postdoctoral Researcher at the University of Louvain, Belgium. At the end of 2001, he joined the Communications and Electronics Department, Telecom Paris, as an Associate Professor, he has been a (full) Professor, since 2011. His research interests include statistical and distributed signal processing, signal processing for digital communications, and resource allocation. From 2014 to 2019, he was a member of IEEE Technical Committee Signal Processing for Communications and Networking. He served as an Associate Editor for the IEEE Communications Letters, from 2004 to 2007. From 2008 to 2012, he served as an Associate Editor and then a Senior Area Editor for the IEEE Transactions on Signal Processing. From 2018 to 2021, he served as an Associate Editor for the IEEE Transactions on Signal and Information Processing Over Networks.

MOUNIR GHOGHO (Fellow, IEEE) received the M.Sc. and Ph.D. degrees from the National Polytechnic Institute of Toulouse, France, in 1993 and 1997, respectively. He was an EPSRC Research Fellow at the University of Strathclyde (Scotland), from September 1997 to November 2001. In December 2001, he joined the School of Electronic and Electrical Engineering, University of Leeds, U.K., where he was promoted to a full Professor, in 2008. While still affiliated with the University of Leeds, in 2010, he joined the International University of Rabat, Morocco, where he is currently the Dean of the College of Doctoral Studies and the Director of TICLab (ICT Research Laboratory). His research interests include machine learning, signal processing, and wireless communication, on which he has published over 300 papers in journals and conferences. He was a recipient of the 2013 IBM Faculty Award and the 2000 U.K. Royal Academy of Engineering Research Fellowship. He is the Co-Founder and the Co-Director of the CNRS-Associated International Research Laboratory DataNet, in big data and artificial intelligence. He has coordinated around 20 research projects and supervised over 30 Ph.D. students in the U.K. and Morocco. In the past, he served as an Associate Editor of many journals including *IEEE Signal Processing Magazine* and the IEEE TRANSACTIONS ON SIGNAL PROCESSING.

ANANTHRAM SWAMI (Life Fellow, IEEE) received the B.Tech. degree in electrical engineering from IIT-Bombay, the M.S. degree in electrical engineering from Rice University, and the Ph.D. degree in electrical engineering from the University of Southern California (USC). He is currently with the U.S. Army's DEVCOM Army Research Laboratory and the Army's Senior Research Scientist (ST) for Network Science. Prior to joining ARL, he held positions with Unocal Corporation, USC, CS-3, and Malgudi Systems. He was the Statistical Consultant of the California Lottery, developed a MATLAB-based toolbox for non-Gaussian signal processing. He has held visiting faculty positions at INP, Toulouse, and the Imperial College, London. His work is in the broad area of network science, including communication and information networks and cyber security. He is an ARL Fellow. Recent awards include the 2018 IEEE ComSoc MILCOM Technical Achievement Award and the 2017 Presidential Rank Award (Meritorious).