POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Graph Neural Networks on heterophilous graphs: performance analysis and new architectures

Supervisors

Prof. Luca VASSIO

Dr. Claas GROHNFELDT

Michele RUSSO

Dr. Giulio LOVISOTTO

Candidate

Andrea CAVALLO

October 2022

Summary

Graph heterophily, also called disassortativity, is a property of networks that models the likelihood of nodes with different characteristics being connected. Several real-world graphs present high levels of heterophily, such as computer networks, where clients usually connect to servers, and dating applications, where users tend to connect with users of the opposite gender. Despite the relevant contexts in which heterophilous graphs are observed, current approaches fail to map nodes to meaningful low-dimensional embeddings. Indeed, Graph Neural Networks (GNNs), currently the state-of-the-art models for machine learning on graph-structured data, implicitly assume homophily, leading to worse performances on disassortative networks. Notwithstanding several works defining novel GNN architectures for heterophilous settings, there is still insufficient understanding of the relation between GNN performances and graph disassortativity.
In this context, this thesis focuses on two main objectives. First of all, the impact of heterophily on GNN performances is discussed. In particular, it is shown that GNNs can still achieve good performances under heterophily if other conditions are met. Previous works already identify other graph properties beyond heterophily that affect GNN performances, but they rely on the assumption of node features being very informative for node labels, which is not always observed in real scenarios. Therefore, in this thesis, the classification capabilities of a Graph Convolutional Network (GCN), chosen as example given its popular and simple design, are modeled in a context without node features, where only graph structural properties are considered. In this setting, it is shown that a target node is likely to be correctly classified if the majority of its 1-hop neighbors have neighbors that, for the most part, share the same label as the target node. To numerically quantify this property, a new metric is introduced: 2NCS (2-hop Neighborhood Class Similarity), defined as the average over neighbors of the percentages of their neighbors with the same label as the target node. Through extensive experiments on real and synthetic graphs, it is shown that nodes with high 2NCS values are usually correctly classified by a GCN, also in some scenarios where features are present. Therefore, 2NCS appears to be an informative metric to estimate GCN performances.
Secondly, two new GNN models for node classification on heterophilous graphs

are proposed: GATH (GAT for Heterophily) and GCNH (GCN for Heterophily). GATH leverages an extended attention mechanism to learn a weight between any pair of nodes in the graph. This weight defines the importance of the message coming from that node. Additionally, GATH separately computes an embedding for the target node and its neighborhood, merging them only at the latest stage through a learnable coefficient. Moreover, messages for a target node are aggregated also from nodes that are not directly connected but might still be informative in heterophilous settings. In conclusion, structural information, namely the node degrees and the distance between the nodes, is added to the feature embeddings as input to the attention weight computation. GCNH is a simpler model that extends GCN by separately learning embeddings for the target node and the neighbors. Experiments on real and synthetic graphs show that GATH and GCNH can achieve competitive performances compared to state-of-the-art methods on heterophilous graphs.

Acknowledgements

Giunto al termine di questo percorso, è il momento di dedicare un pensiero a tutte le persone che mi hanno accompagnato nel corso di questi anni.
Ringrazio mamma, papà, Elisa e Chiara per tutto il loro supporto, che fosse a migliaia o centinaia di chilometri di distanza o chiusi nella stessa casa per mesi.
Ringrazio gli immancabili Scaturchi Ciotti: un grazie a Rolli, Lollo, Cecio, Will, Tonsio, Chri, Giglio, Marta, Sara, Bibi, Mega, Laurable, Beca, Feca, Gio, Matte. Abbiamo condiviso bene o male qualsiasi cosa in questi anni, partendo dalle famigerate interrogazioni al liceo e arrivando alle varie vacanze, feste, serate, calcetti e mille altre cose e sono molto felice e fiero di poter condividere anche questo momento con tutti voi.
Ringrazio Andre e Gio, l'esperienza negli USA che abbiamo vissuto insieme è stata una parte fondamentale di questi anni e siete state le persone migliori con cui condividerla.
Ringrazio i miei colleghi, compagni e amici del team YouForAll, Davide, Andrea e Karim, con cui ho avuto la fortuna di lavorare in questi ultimi due anni di ASP, tra alti e bassi, da vicino e da lontano, ma arrivando a condividere ben più del semplice lavoro.
I thank the Saturday Sun team, Elena, Marco, Saar, Eszter, Pietro, Theresa and Alba, for having made my 6 months in Munich so much more than just writing a Master Thesis.
Ringrazio il mio relatore, Luca Vassio, per avermi seguito molto efficacemente nonostante la distanza e avermi sempre supportato e aiutato con grande disponibilità.
I thank all the people in the AI4Sec team of the Huawei Munich Research Center, Claas, Michele, Giulio, Matteo, Yuchen, Tu, Nedim, Adrian, Haigang and Federico, who have helped me a lot with countless discussions and brainstorming sessions about my work, but most of all who have managed to make me feel at home during my 6 months in Munich.

Table of Contents

List of Tables

List of Figures

Acronyms

2NCS

2-hop Neighborhood Class Similarity

GAT

Graph Attention Network

GATH

Graph Attention Network for Heterophily

GCN

Graph Convolutional Network

GCNH

Graph Convolutional Network for Heterophily

GNN

Graph Neural Network

kNN

k-Nearest Neighbor

LSTM

Long-Short Term Memory

MLP

Multi Layer Perceptron

NLP

Natural Language Processing

ReLU
: Rectified Linear Unit

SVD
: Singular Value Decomposition

Chapter 1

Introduction

This chapter introduces the problems of Graph Neural Networks on heterophilous graphs and defines the research questions that this thesis aims at answering. We then present the contributions of the work and we describe how the thesis is organized in its various chapters.

1.1 Scenario

In the latest years, Graph Neural Networks (GNNs) have achieved state-of-the-art performance on graph-structured data. The main reason behind their success consists in their ability to effectively represent information related to the graph structure, instead of just considering each node as a single instance. However, despite their success, experimental evidence has shown that GNN performance is severely hindered on some graphs presenting high levels of heterophily. Heterophily is a property of networks describing the extent to which nodes are connected to dissimilar nodes. Similarity can be measured in terms of node features or labels. Heterophilous graphs are present in many real-world scenarios. One example are computer networks, where clients tend to connect to servers and vice-versa, but it is unlikely for a client to directly communicate to another client. Similarly, in dating applications users tend to connect to users of the opposite gender. The large diffusion of heterophilous networks in real-world settings motivates the need to further investigate and improve GNN performance on these graphs.

1.2 Research questions

Within the context of GNN performance on heterophilous graphs, this thesis aims at answering two main questions.

RQ1. Do GNNs perform badly on all heterophilous graphs? Are there other metrics beyond heterophily that can be useful to characterize GNN performance on graphs?

RQ2. Is it possible to extend the design of GNNs in order to improve performance on heterophilous graphs?

The first question is relevant to gain a deeper understanding about the relation between GNN performance and graph properties. On the one hand, this allows to better characterize the behavior of existing GNNs and predict which scenarios are problematic for them. On the other hand, clarifying the limitations of current GNNs is helpful to improve them.
The second question, instead, has a more practical focus. The goal is to create new GNN models that improve performance on graphs that currently represent difficult benchmarks for GNNs. This allows to extend the scenarios on which GNNs can be successful and enlarge their impact.

1.3 Contributions

This thesis provides two main contributions. First of all, an intuitive analysis of the learning process of a GCN shows that, in a scenario where features are not considered, the capability to correctly classify a node is affected by the labels of the 1-hop and 2-hop neighbors. To numerically quantify this property, we define a new metric, 2-hop Neighborhood Class Similarity (2NCS). We show that this metric provides useful insights about how favorable the graph structure is for a GCN, improving on homophily ratio and other existing measures.
Secondly, the thesis contains the definition of two GNN-based models for heterophilous graphs: GATH (GAT for Heterophily) and GCNH (GCN for Heterophily). GATH relies on an extension of the attention mechanism to learn weights among pairs of nodes that indicate how much the two nodes are relevant to each other. This allows to assign more importance to the messages of the nodes that are classified as more relevant and filter out the information carried by nodes that are not important. Moreover, GATH enlarges the scope of the neighborhood to aggregate information also from nodes that are not directly connected. Indeed, in heterophilous graphs useful information can be located in different portions of the graph. In conclusion, GATH separately computes an embedding for the target node and for the neighborhood and merges them only as a final stage, learning a coefficient that balances the contributions of the two. GCNH is, instead, a simpler model that extends the design of GCN by separately learning the embedding for the neighborhood and for the target node. These are then merged at the end using a learnable weight, similarly to GATH. Experiments on real and synthetic

graphs show that GATH and GCNH achieve competitive results compared to state-of-the-art models.

1.4 Thesis overview

The thesis is organized as follows. In Chapter 2, we present the basics of machine learning on graphs and we describe the architectures of the most common graph neural networks. Moreover, we introduce the problem of graph neural network performance on heterophilous graphs and we present several metrics to measure heterophily.
In Chapter 3, we report an overview of the existing approaches to improve GNN performance on heterophilous graphs. Then, we explain Cross-Class Neighborhood Similarity (CCNS), a metric that attempts to characterize GCN performance on heterophilous graphs, and we discuss its strength and limitations.
Chapter 4 introduces the real-world and synthetic datasets used in the thesis. The presented graphs are commonly used in works dealing with heterophilous networks and cover a wide range of heterophily values.
In Chapter 5, we propose a new metric to characterize a graph property that is relevant for GCN performance: 2-hop Neighborhood Class Similarity (2NCS). We introduce the metric through an analysis of the learning process of a simplified GCN model and we evaluate it by observing how informative it is for GCN performance on several graphs.
Chapter 6 describes the two GNN models introduced in this thesis: GATH (GAT for Heterophily) and GCNH (GCN for Heterophily). We then analyze their performance in detail in Chapter 7. We report comparisons with state-of-the-art models and we perform ablation studies to show the importance of the different design choices.
Chapter 8, in conclusion, summarizes the main findings of the thesis, points out the most relevant limitations of the work and proposes future directions to expand the study and the experiments.

Chapter 2

Background

In this chapter, we define the notation used throughout the thesis and the problem of supervised node classification that this work deals with. Moreover, we introduce Graph Neural Networks from a general perspective and we describe the most common GNN architectures. To conclude, we present the notion of heterophily in graphs, we justify why it is relevant to analyze its relation with GNN performance and we provide an overview on the different existing metrics to measure heterophily and, more in general, to gain information about the connectivity patterns in a graph.

2.1 Problem statement and notation

Let $G = (V, E)$ be an unweighted and undirected graph, where V is the set of nodes and E is the set of edges. The connectivity information in the graph can also be represented by means of the adjacency matrix $A \in \mathbb{R}^{n \times n}$, where $n = |V|$ is the number of nodes and each element of the matrix A_{ij} is equal to 1 if nodes i and j are adjacent to each other, equal to 0 otherwise. Each node is associated to a feature vector x_i of size f, and the complete set of features in the graph is denoted as $X \in \mathbb{R}^{n \times f}$. Each node is also associated with a label $y_i \in C$ representing the class the node belongs to, where C represents the set of labels. The set of nodes with label $c \in C$ is denoted as V_c. The set of adjacent nodes to a node i (also called *neighbors*) is denoted as $N(i)$, and the degree of the node $|N(i)|$ is also denoted as d_i. The symbol $N'(i)$ is used to denote the set of nodes connected to i plus the node i itself. The symbol $N_k(i)$ is used to identify the set of nodes up to k hops away from node i (i.e. nodes up to the k-hop neighborhood). The number of nodes adjacent to node i that have label $c \in C$ is denoted as $d_i^{(c)}$. I indicates the identity matrix.
The task considered in this work is *supervised node classification*, which corresponds

to learning a mapping $\mathcal{F} : V \to C$ exploiting the information of the graph G, the features X and the labels such that the loss $\mathcal{L}(y_i, \hat{y_i})$ is minimized, on average, for the nodes in the training set, where $\hat{y_i} = \mathcal{F}(G, X, i)$ is the predicted label for node i and the loss function $\mathcal{L}(\cdot, \cdot)$ is used to describe the distance between the true label y_i and the predicted label.

2.2 Dealing with graph-structured data

Deep learning has witnessed an outstanding expansion in numerous fields thanks to its ability to extract relevant features from data and automatically recognize relevant patterns that help solve a wide variety of tasks. However, most of the common deep learning approaches are designed to operate on data with a specific structure. In particular, the two most developed and popular areas in which deep learning thrives are computer vision and natural language processing, which deal respectively with images, an example of grid-structured data, and with text, naturally represented as a sequence. However, real-world data can assume much more complex structures that cannot be represented as grids or sequences without a significant loss of relevant information.

One class of data that would suffer from being simplified into a grid or a sequence is graph-structured data, where different entities in the dataset are characterized both by their own features and by their interactions with the other entities. Graphs can be used to effectively model several real-world data, such as social networks, computer networks and protein structures. To understand why adapting graph-structured data to images or sequences may be problematic, it is important to underline the differences between these data structures.

One important property that distinguishes graphs from images or sequences is the notion of order. As a matter of fact, both images and sequences intrinsically define an ordering between the different elements in the data (for example, each pixel in an image is surrounded by other pixels with a specific spatial structure, and, similarly, words in a sentence follow a defined ordering and it is straightforward to understand which word comes before another one). This property, however, does not hold with graphs, where there is no notion of order between the neighbors of a node. The deep learning architectures dealing with images or sequences, in most cases, assume an ordering among the inputs. Therefore, when applied to graphs, they introduce an assumption that may bias the results.

Moreover, another relevant difference is the fact that in images the number of neighbors for a center pixel is fixed and always the same, whereas in graphs each node can have a different number of neighbors.

Taking these observations into consideration, it is clear that the approaches that target images or sequences cannot be directly applied to graphs without the introduction of wrong assumptions in the data and loss of relevant information. This motivates the development of Graph Neural Networks (GNNs), a more flexible neural network paradigm that extends the aforementioned architectures to graph-structured data.

2.2.1 Graph Neural Networks

Graph Neural Networks are a class of neural networks specifically designed to target graph-structured data. They were originally introduced by [1] and [2] but they gained popularity in more recent years. The basic idea behind GNNs is to compute an embedding $h_u \in \mathbb{R}^e$ for each node u in a latent space of size e based on a computational graph that differs depending on the position of the node in the graph. In particular, as depicted in Figure 2.1, the computational graph for each node is composed of the 1-hop neighbors at the penultimate layer. Then, each node in this layer is connected to its 1-hop neighbors, and this structure can be expanded for any number of layers.

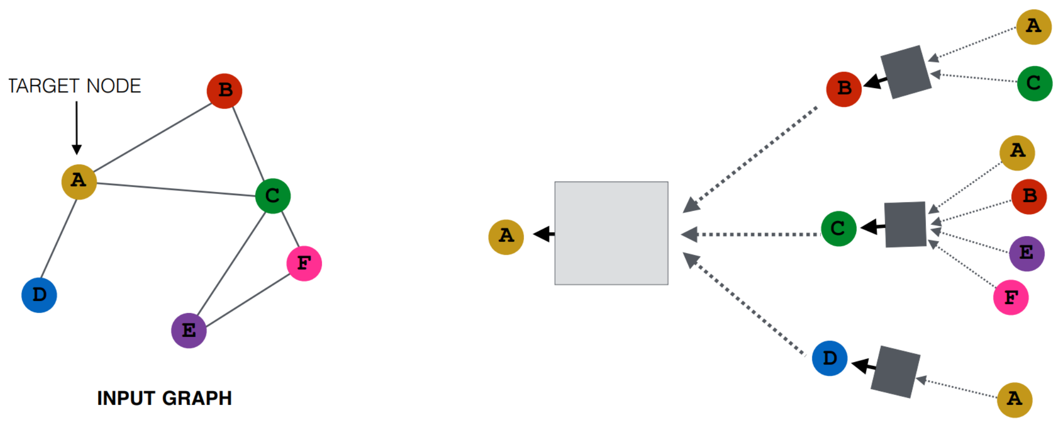


Figure 2.1: Example of computational graph for the target node A given the graph structure. Each node at each layer is connected to all its neighbors in the original graph. The image is taken from [3].

We can observe that this formulation defines a flexible framework that can represent any structure around a node.
The computational graph is then exploited to generate an embedding for the target node. In particular, each node at each layer is represented by an embedding in a latent space, and these embeddings are manipulated according to the following two steps:

- *message transformation*: at each layer l, the embeddings (also called *messages*) of the nodes u in the previous layer h_u^{l-1} are transformed:

$$m_u^l = \mathrm{MSG}^l(h_u^{l-1}) \tag{2.1}$$

- *message aggregation*: to generate the message for the target node, the transformed messages of its neighbors and of the node itself are aggregated:

$$h_v^l = \mathrm{AGG}^l(\{m_u^l, u \in N'(v)\}) \tag{2.2}$$

The messages at the final layer correspond to the node embeddings and can be used to perform several downstream tasks. For node-level tasks such as node classification, we can use the single node embeddings, whereas for link-level tasks (e.g. link prediction) or graph-level tasks (e.g. graph classification) we can exploit the combination of the embeddings of different nodes.
The definition of the MSG and AGG functions mentioned above changes depending on the GNN design. We introduce two of the most common GNN architectures in the following.

2.2.2 Graph Convolutional Network

Graph Convolutional Networks (GCNs) are a particular class of GNNs introduced by [4]. They are inspired by Convolutional Neural Networks (CNNs), which achieved great success in the field of computer vision thanks to their ability to capture repeating patterns in images through the convolution operation, i.e. the multiplication between a fixed-size filter and different patches of the image. However, this notion of convolution strongly relies on two specific characteristics of images: the number of neighbors of a central pixel is fixed for all pixels and there is a notion of ordering among the neighbors, so changing the order of the pixels will change the results. These two assumptions are not true in graphs, and therefore the notion of convolution must be extended for this specific case.
To this end, [4] define the message transformation MSG as a learnable linear layer and the aggregation function AGG as a weighted average of the messages from the neighboring nodes, where the weighing factors are the target nodes' degrees. Overall, the layer-wise propagation rule of GCN for a node v can be written as

$$h_v^l = \sigma\Big(\sum_{u \in N'(v)} W^l \frac{h_u^{l-1}}{|N'(v)|}\Big) \tag{2.3}$$

where σ is a non-linearity such as ReLU and $W^l \in \mathbb{R}^{e^l \times e^{l-1}}$ is a learnable matrix for the l-th layer. The size of the embeddings at each layer can vary. We can also express the same formulation in matrix form:

$$H^l = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{l-1} (W^l)^T) \tag{2.4}$$

where $H^l \in \mathbb{R}^{n \times e^l}$ is the matrix containing the node embeddings at layer l, $\tilde{A} = A+I$ is the adjacency matrix with added self-connections (note that adding self-loops corresponds to adding the identity matrix to the adjacency) and $\tilde{D} \in \mathbb{R}^{n \times n}$ is a diagonal matrix whose values on the diagonal are defined as $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij} = d_i$ (i.e. $\tilde{D}_{ii}$ is the degree of node i). At the first layer $l = 0$, the node embeddings are the node features, i.e. $H^0 = X$ and $e^0 = f$.

2.2.3 Graph Attention Network

The Graph Attention Network (GAT), introduced by [5], is another design of graph neural network that extends the information representation capabilities of the GCN. In particular, it introduces in the graph neural network domain the concept of attention, popular in NLP and computer vision. Thanks to the attention mechanism, it is possible for a model to assign different levels of importance to different elements in the input, based on how relevant these elements are considered for the final prediction. This allows to deal with large and redundant inputs more effectively, since the attention mechanism is able to discard useless information and focus on the most relevant parts. To encode this mechanism into GNNs, [5] define a graph attention layer with the following formulation. The MSG function is encoded by a learnable linear transformation, represented by a matrix W^l for each layer l. Then, attention scores are computed on the transformed messages and normalized with a softmax function:

$$\alpha^l_{uv} = \text{softmax}_v(\text{LeakyReLU}((z^l)^T[W^l h^{l-1}_u || W^l h^{l-1}_v])) \tag{2.5}$$

where $||$ represents the concatenation operation and $z^l \in \mathbb{R}^{2e^{l-1}}$ and $W^l \in \mathbb{R}^{e^l \times e^{l-1}}$ are learnable parameters at layer l. The attention scores obtained in this way are used to compute a linear combination of the features of the neighboring nodes, to which a non-linearity is then applied to obtain the final embedding for the target node, according to the following formulation:

$$h^l_u = \sigma(\sum_{v \in N'(u)} \alpha^l_{uv} W^l h^{l-1}_v) \tag{2.6}$$

Also in this case, at the first layer $l = 0$, the node embeddings are the node features. Following this approach, we can also define multi-head attention, which corresponds to computing different sets of attention scores (one per head) and using each of them to generate a different embedding for the target node, which is then concatenated to those of the other heads. This allows for different heads to focus on different aspects of the input, therefore capturing more information. For K heads, we can formulate multi-head attention as

$$h^l_u = ||^K_{k=1}(\sigma(\sum_{v \in N'(u)} \alpha^{l,k}_{uv} W^{l,k} h^{l-1}_v)) \tag{2.7}$$

2.2.4 GATv2

As pointed out in [6], the attention mechanism of GAT, reported in Equation (2.5), is not expressive enough to solve specific problems. In particular, [6] define two classes of attention: *static attention*, the one of GAT, and *dynamic attention*, the original idea of attention ([7]) and the one implemented by GATv2, an extension of GAT described in [6]. Going deeper into details, static attention employs a function that, for any query node, is monotonic with respect to the neighbor scores, i.e. the ranking of attention coefficients is shared across all nodes in the graph and is unconditioned on the query node. Dynamic attention, conversely, does not suffer from this. The limitations of static attention prevent it from being able to solve specific problems. One example designed by [6] is the *dictionary-lookup* problem represented in Figure 2.2.

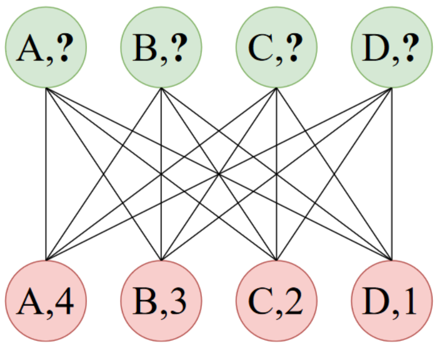


Figure 2.2: Example of *dictionary-lookup* problem with 4 key nodes. Letters are attributes (i.e. node features) and numbers are labels. Each feature corresponds to one label. The nodes in red are the keys, used as training set, and those in green are the queries, used as test set. The goal, therefore, is to predict the labels of the green nodes using their attribute. The image is taken from [6].

Experiments show that the simple GAT cannot solve this problem, being unable to fit the training set. This confirms the limited expressiveness of this formulation of attention. To solve this limitation, [6] propose a new formulation of attention, which, in this thesis, is referred to as *GATv2 attention* or just *attv2*:

$$\alpha_{uv}^{l} = \text{softmax}_v((z^l)^T \text{LeakyReLU}(W^l[h_u^{l-1}||h_v^{l-1}])) \tag{2.8}$$

where the symbols are the same as in Equation (2.5). With respect to the standard GAT attention, the multiplication with the vector z^l is performed after the non-linearity. Theoretical proofs and experiments in [6] prove that this simple modification improves the model expressiveness. Indeed, GATv2 easily solves the *dictionary-lookup* problem and improves GAT performance on some real-world datasets.

2.3 Heterophily in graphs

Graph homophily is a property of graphs defined as the extent to which nodes that are connected with each other share the same label or similar features. The concept is inspired by social sciences, where it is observed that people tend to homogeneously aggregate according to common characteristics or, in other words, *birds of a feather flock together* [8]. On graphs, node characteristics are represented by either the label or the features. A graph with a low level of homophily is characterized by connections among nodes belonging to different classes or with different attributes and it can be called heterophilous or disassortative. Although homophily is a phenomenon observed in many social networks, also heterophilous graphs are common in several real-world scenarios. We can find an example in dating networks, in which most of the users (nodes) tend to connect with nodes of the opposite gender. We can observe another relevant example in computer networks, in which clients tend to connect to servers and not to other clients.
Despite the importance of real-world applications where data can be encoded into heterophilous graphs, most of the datasets commonly used for GNN evaluation present high homophily, and only recent works have observed empirically that GNN performance on heterophilous graphs suffers from a significant drop [9, 10, 11, 12]. To provide an intuitive explanation of this behavior, we analyze two main characteristics of the uniform message passing framework [13].
First of all, standard GNNs define the neighbors of a node through graph topology, i.e. they generate the embedding of the target node by aggregating from the nodes that are connected to it. Although this works well on homophilous graphs, where connected nodes tend to share the same label and therefore carry relevant information, in heterophilous networks this formulation fails to capture long-distance relationships between nodes of the same class. As a matter of fact, in heterophilous settings nodes sharing the same label are likely not to be connected, and therefore a single GNN layer cannot model their interaction. One possible solution to this problem consists in stacking multiple GNN layers, such that node embeddings depend on nodes that are located up to the L-th neighborhood, where L is the number of layers. However, this approach has two main drawbacks. Firstly, the information coming from nodes at further neighborhoods needs to be propagated through several layers before reaching the target node, therefore losing expressive power. Moreover, stacking multiple layers might cause the oversmoothing problem, i.e. a degradation of performance related to the fact that, when GNNs become very deep, the computational graphs for most nodes become similar and, therefore, nodes are encoded into similar embeddings in the latent space [14].
In addition to this, other issues are related to the message aggregation strategy adopted in GNNs. As a matter of fact, GNNs aggregate information from the 1-hop neighbors of a node, which therefore play a very important role in the computation

of the embedding for the central node. For example, in GCN each neighbor provides the same contribution to the target node embedding as the target node itself. This is advantageous when the neighbors are informative for the target node, but hugely detrimental when there are relevant differences between a node and its neighbors. Indeed, aggregating information from nodes belonging to different classes might lead to noisy latent representations, making it impossible to correctly recognize the class each node belongs to.

2.4 Measuring heterophily

Although graph homophily is a relevant property to characterize real-world networks, it is not straightforward to define an effective metric to measure the level of homophily of a graph, since this may be related to different characteristics of the graph (e.g. labels or features) and might vary in different areas of the graph.

2.4.1 Global measures

The most common measure for heterophily is the edge homophily ratio:

$$h = \frac{|\{(u,v) : (u,v) \in E \wedge y_u = y_v\}|}{|E|} \tag{2.9}$$

i.e. the fraction of edges in the graph that connect nodes belonging to the same class. For homophilous networks, this measure tends to 1, whereas it assumes values close to 0 for networks characterized by the heterophily property. Although it is an intuitive measure, [15] point out some weaknesses of the edge homophily ratio. As a matter of fact, they underline how the value of this measure is dependent on the number of classes in the graph, since, in a graph in which topology is independent on the labels, the edge homophily ratio of a node u with label y_u is $d_u^{(y_u)}/d_u \approx |V_{y_u}|/|V|$. Moreover, the homophily ratio for graphs with strongly unbalanced classes may be misleading, since it mostly depends on the most common class. Therefore, [15] introduce a new measure for homophily in graphs that addresses these limitations:

$$\hat{h} = \frac{1}{|C|-1} \sum_{c=0}^{|C|-1} [h_c - \frac{|V_c|}{n}]_+ \tag{2.10}$$

where $|C|$ is the number of classes, V_c is the set of nodes belonging to class c, $[a]_+ = \max(a,0)$ and h_c is the class-wise homophily metric defined as

$$h_c = \frac{\sum_{u \in V_c} d_u^{(c_u)}}{\sum_{u \in V_c} d_u} \tag{2.11}$$

where d_u is the number of neighbors of node u (node degree) and $d_u^{(c_u)}$ is the number of neighbors of node u that have the same label. With respect to the edge homophily ratio h, $\hat{h}$ does not depend on the number of classes and handles more effectively the case of unbalanced classes. Some examples of graphs with various label-topology relationships and the respective values for h and $\hat{h}$ are reported in Figure 2.3. The depicted examples show how $\hat{h}$ is independent of the number of classes and is not affected by class imbalance.

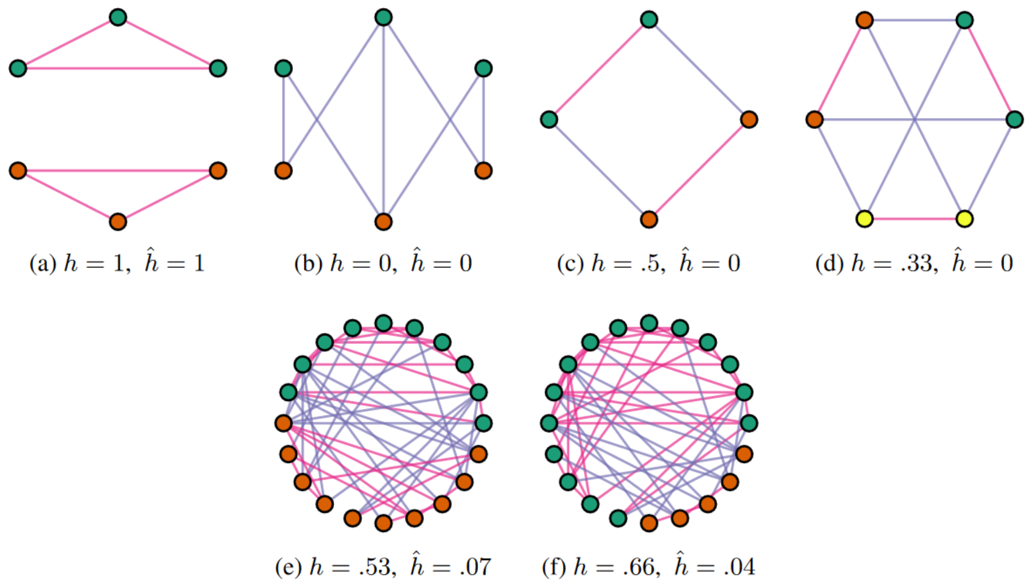

(a) $h = 1,\ \hat{h} = 1$
(b) $h = 0,\ \hat{h} = 0$
(c) $h = .5,\ \hat{h} = 0$
(d) $h = .33,\ \hat{h} = 0$
(e) $h = .53,\ \hat{h} = .07$
(f) $h = .66,\ \hat{h} = .04$

Figure 2.3: Comparison between the homophily measures h (2.9) and $\hat{h}$ (2.10). Colors indicate labels, pink edges connect nodes of the same class, while purple edges connect nodes of different classes. (a) and (b) show pure homophily and pure heterophily, for which both measures are respectively 1 and 0. (c) and (d) are graphs in which each node is connected to one node of each class. h depends on the number of classes, whereas $\hat{h}$ does not. (e) and (f) are random Erdős-Rényi graphs where edges are independent of labels. h is sensitive to class imbalance, whereas $\hat{h}$ is not. The image is taken from [15].

Another global measure of the tendency of nodes with similar attributes or labels to be connected to each other is the assortativity coefficient introduced by [16], defined as

$$r_{global} = \frac{\sum_g M_{gg} - \sum_g a_g b_g}{1 - \sum_g a_g b_g} \tag{2.12}$$

where M is the *mixing matrix*, in which each cell M_{gh} is the fraction of edges in the network that connect a node with label g to one of label h. Then, a_g and b_g are the number of outgoing and incoming edges of all nodes of label g. The assortativity coefficient ranges from -1 (fully disassortative network) to 1 (fully assortative network).

Table 2.1 reports the values of the different heterophily measures for eight different real-world datasets. These datasets are used throughout the thesis and we provide a detailed description of their characteristics in Section 4. According to all measures, `Cora` and `Citeseer` are homophilous graphs, whereas the others can be considered heterophilous. In particular, `Texas` is the most heterophilous among all, whereas the order of the others slightly changes depending on the metric used.

	`Cornell`	`Texas`	`Wisconsin`	`Film`	`Chameleon`	`Squirrel`	`Cora`	`Citeseer`
h (Eq. (2.9))	0.30	0.11	0.21	0.22	0.23	0.22	0.81	0.74
$\hat{h}$ (Eq. (2.10))	0.06	0.00	0.09	0.01	0.06	0.03	0.77	0.63
r_{global} (Eq. (2.12))	-0.07	-0.26	-0.15	0.00	0.03	0.01	0.77	0.67
Rank	4.33	1	3.33	3	5	3.67	8	7

Table 2.1: Homophily measures for eight real-world datasets. The row Rank reports the average ranks of the datasets according to the different measures, in increasing order of homophily.

2.4.2 Local measures

The measures described so far provide a value to estimate the homophily level of the graph as a whole. However, one single value might not be informative enough to describe the connectivity patterns present in the network, as different nodes might show different assortativity behaviors [17, 15]. Therefore, it is useful to define also local assortativity measures that describe the homophily of a single node. One first measure in this sense is node homophily [18], defined as

$$h_u = \frac{1}{|V|} \sum_{u \in V} \frac{d_u^{(y_u)}}{d_u} \tag{2.13}$$

i.e. the fraction of edges starting from one node that connect it to a node with the same label.
It is also possible to define a local mixing measure [19], inspired by the global assortativity coefficient (Equation (2.12)), as

$$r_{local}(l) = \frac{\sum_g M_{gg}(l) - \sum_g a_g^2}{1 - \sum_g a_g^2} \tag{2.14}$$

where $M_{gh}(l)$ is the weighted sum of edges that connect nodes of class g to nodes of class h and the weights encode how local the edge is to the current target node l. In particular, this information is encoded through the personalized PageRank vector.
Further information about the connectivity patterns of the networks can be observed through the compatibility matrix H, where each value is defined as

$$H_{kl} = \frac{|(u,v) \in E : y_u = k, y_v = l|}{|(u,v) \in E : y_u = k|} \tag{2.15}$$

Note that the compatibility matrix H is very similar to the mixing matrix defined in Equation (2.12), with the only difference of the normalization factors.

Figure 2.4 shows the compatibility matrixes for the presented datasets, thanks to which we can observe some more complex patterns related to connectivity among classes. Homophilous networks (`Cora` and `Citeseer`) have higher values on the diagonal, as nodes of the same class tend to be connected. In `Film`, the compatibility matrix shows different patterns for different columns, indicating that nodes belonging to different classes have, on average, different numbers of incoming edges from nodes of specific classes. In `Cornell` and `Texas`, only one node belongs to class 1, leading to unreliable results in the matrix. In `Chameleon`, it is possible to observe stronger connectivity among nodes of classes 2, 3 and 4 with respect to the other two classes. In `Squirrel`, nodes of class 4 have on average more connections than the nodes of the other classes.

We can make further considerations by observing the distribution of the local node homophily ratio across nodes in the network in Figure 2.5. In particular, we can notice that heterophilous graphs like `Film` and `Cornell` have a relevant fraction of nodes with high homophily level, although the majority of nodes are heterophilous. Moreover, it can be observed that `Chameleon` and `Squirrel` have a more uniform distribution of homophily level across nodes, whereas the distributions for `Film`, `Texas`, `Wisconsin` and `Cornell` have a stronger concentration of nodes with homophily ratio around 0. Distributions for the homophilous graphs, `Cora` and `Citeseer`, are mostly centered around high homophily values (almost 1), but there is a fraction of nodes with strongly heterophilous connections.

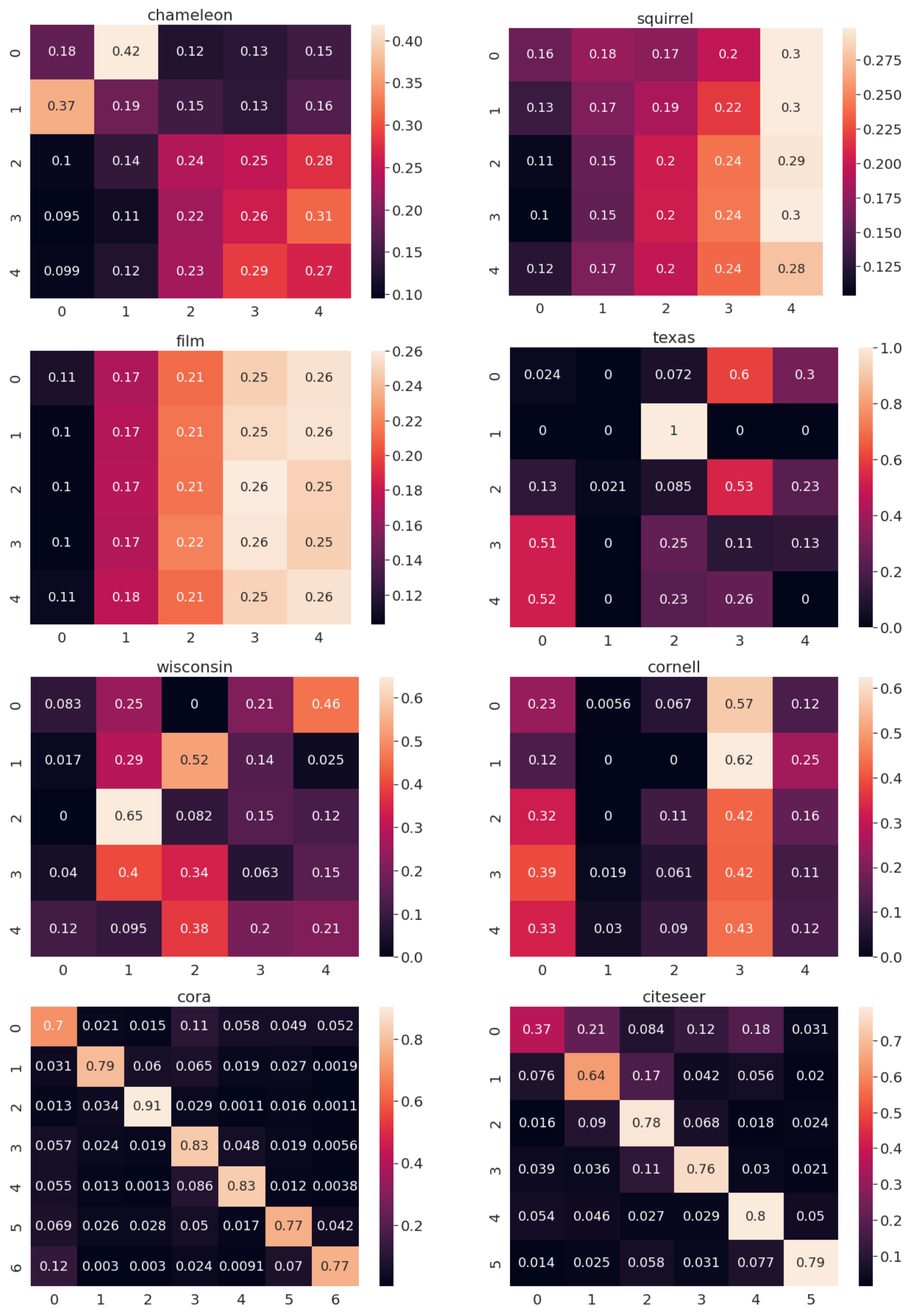


Figure 2.4: Compatibility matrixes of different datasets

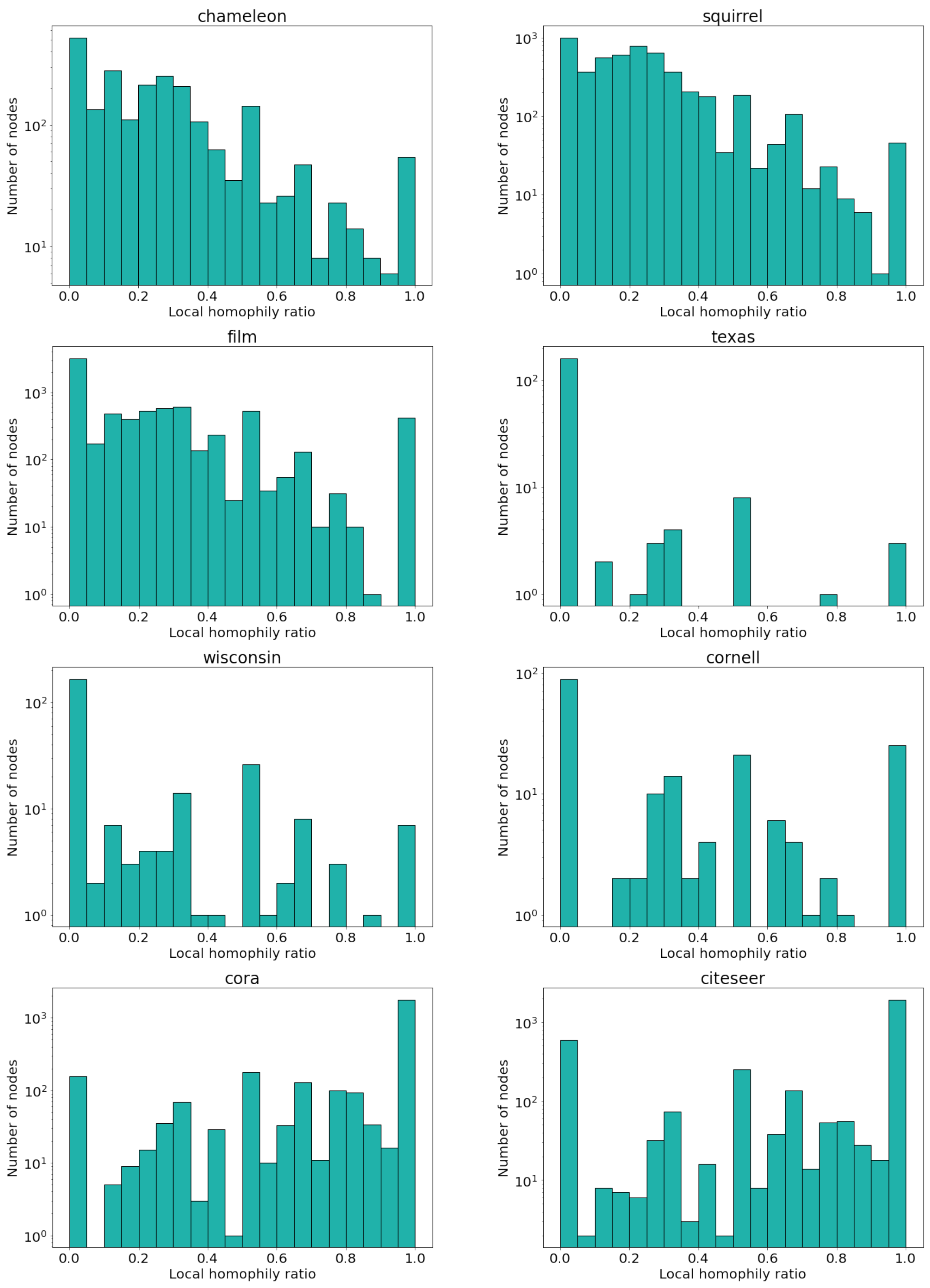


Figure 2.5: Local homophily distribution of different datasets

Chapter 3

Related work

This chapter presents an overview on the existing GNN models for heterophilous graphs. We categorize the different approaches based on the main design choices and we compare them to the models proposed in this thesis. Then, we present another related work that shows that GNNs can still perform well on heterophilous graphs, if some conditions are met, and we explain the measure they introduce, Cross-Class Neighborhood Similarity (CCNS). This metric is related to GNN performance on graphs, but we also point out some limitations that motivate the analysis performed in this thesis.

3.1 GNNs to deal with heterophily

Experimental evidence has shown a worsening in classification performance for GNNs when dealing with heterophilous graphs [9, 11]. This is commonly attributed to the uniform message passing framework of GNNs [13], which introduces noise in node representations and does not allow for effective aggregation of information from further nodes. Although it has been pointed out that heterophily is not a sufficient condition for bad GNN performance [20], it is still true that there exist specific categories of graphs on which GNNs do not manage to successfully represent information and achieve good performance, and heterophilous graphs are a superset of them. Therefore, several works tried to tackle the problem by introducing new designs in the GNN architecture or modifying the underlying structure of the graph to make it more suitable for the message passing framework. It is therefore relevant to study the problem and analyze the works that provided some contributions to improve representation capabilities on heterophilous networks. According to [13], the main approaches rely on two categories of improvements:

- *Non-local Neighbor Extension*: some methods try to expand the scope of GNNs to reach potentially important nodes located outside of the close neighborhood.

This approaches rely on the assumption that, since it cannot be assumed that nodes with the same label are connected, useful information (usually carried by nodes with the same label) can be discovered also in nodes that belong to higher-order neighborhoods with respect to a target node.

- *GNN Architecture Refinement*: other methods modify the GNN architecture introducing specific designs in order to improve representation capabilities on heterophilous graphs.

In the following, we present and briefly analyze the main approaches in these directions.

3.1.1 Non-local Neighbor Extension

The standard design of GNNs aggregates information from 1-hop neighbors to create an embedding for the target node. Information from nodes located at further hops can be reached by stacking more GNN layers, but this leads to an indirect connection that does not allow for effective message passing between the two nodes. However, in heterophilous networks useful information might be found in nodes that are not connected to the target node, and these situations are harder to capture for a GNN. At the same time, the strong influence of the 1-hop neighborhood on the embedding of the target node can have a negative impact if the connected nodes are different and carry noisy information. Therefore, a category of approaches targeting disassortative graphs tries to extend the neighborhood to reach nodes located further away in the graph. In particular, in [13] this class of approaches are further categorized into two subclasses: *high-order neighbor mixing* and *potential neighbor discovery*.

High-order neighbor mixing

Methods belonging to the first class create latent representations of nodes using neighbors at different hops separately and then combine them in a second stage. For example, MixHop [21] concatenates the GCN layers corresponding to multi-hop neighbors (specifically 1 and 2-hop neighbors). A similar approach is adopted in H2GCN [9], where at each message passing step information is aggregated from higher-order neighborhoods. UGCN [11] proposes to aggregate information from three distinct neighborhoods (1-hop, 2-hop and top k most similar nodes based on features) and leverages the attention mechanism to allow the network to flexibly choose which neighborhood is more relevant for the current graph. Moreover, TDGNN [22] extends this idea even further by generating different latent embeddings for the nodes using, separately, different hop neighborhoods up to the k-hop, and then merges the representations through a simple sum or a weighted sum with learnable attention coefficients. Another approach that belongs to this category is FSGNN [23], where latent embeddings for the nodes are computed using different hop neighborhoods and by adding or removing self-loops, and then representations are weighted by learnable coefficients and concatenated into a final representation.

Potential neighbor discovery

Methods belonging to this class also rely on the assumption that nodes carrying relevant information might not be connected to the target node and therefore should be looked for also in other portions of the graph. However, they do not consider the entire k-hop neighborhood for node representations, but try to understand which nodes in the graph might be more relevant for the current target node, regardless of their location. In this context, Geom-GCN [18] maps the input nodes in a latent space and then defines the relative importance of nodes based on a distance metric in the latent space, which is able to capture deeper information than the plain graph structure. Two other approaches, NL-GNN [24] and GPNN [25], after sampling a subgraph centered on a target node, define an order between nodes in the subgraph based, respectively, on attention and on pointer networks (a variation of a bidirectional LSTM in a sequence-to-sequence fashion). Then, messages from ordered nodes are aggregated using a one-dimensional convolutional layer. In the same context, HOG-GCN [26] tries to model the relative importance of two nodes based on the estimation of a homophily degree matrix, that aims at capturing how likely it is for the two nodes to belong to the same class. The matrix is estimated from the features and is integrated in the label propagation mechanism. Also UGCN [11] can be included in this category, since one of its neighborhood aggregations is based on feature similarity, which is then considered as a metric to estimate node relative importance. SimP-GCN [27] uses feature similarity to build a new graph based on kNN and merges representations of this new graph with the original graph to exploit both sources of information. WRGNN [17] proposes to modify the input graph by creating new connections among nodes based on two main factors: structural similarity of nodes and proximity in the original graph. The structural similarity is encoded by defining the ordered sequence of degrees of the neighbors and computing the similarity between sequences of different nodes. Edges in the original graph are also taken into consideration when building the connections for the transformed graph. Through this transformation, the modified graph has an increased homophily ratio and simple GNNs can achieve good performance on node classification. Similarly, BM-GCN [28] leverages an MLP to estimate the block matrix, representing the likelihood of interconnections between nodes of two classes. This block matrix is then integrated in the graph convolution operation to increase the strength of the connection between nodes that are likely relevant to each other. In conclusion, GloGNN [29] estimates a coefficient matrix to model the relative importance of nodes. With respect to a simple attention mechanism, this formulation allows for negative values as coefficients (useful for connections among different nodes) and is more efficient thanks to a change in the order of matrix multiplications. Moreover, the optimization problem is formulated such that it has a closed-form solution.

3.1.2 GNN architecture refinement

The second macrocategory of approaches modifies directly the GNN architecture by adopting design choices that can account for more complex relationships than the standard GNN framework. This category can be further subdivided into three main approaches: *adaptive message aggregation*, *ego-neighbor separation* and *inter-layer combination*.

Adaptive message aggregation

The idea of the approaches belonging to this class is to assign different weights to messages coming from different nodes in the graph based on an estimation of how relevant the node is to the current target. The weight can be assigned following diverse methods. For example, in [30], the authors describe GCNs from a spectral point of view and observes that their standard formulation makes use of a low-pass filter (represented by the average of the node messages), which decreases the differences between center node and neighborhood and, if these two are very different, generates a noisy representation. On the other hand, high-pass filters amplify the differences and therefore can exploit the information carried by dissimilar nodes in the neighborhood. In practice, this is implemented in their model, FAGCN, with the introduction of learnable weights between nodes that can also assume a negative value. Similarly, ACM [31] adds to the GCN formulation also an identity filter, consisting of a linear combination of low-pass and high-pass filters. GBK-GNN [12] is a GCN with two kernels, one tailored to deal with homophilous node pairs, the other for heterophilous pairs. A gate (encoded through a MLP) is trained to understand which of the two kernels to apply to each node pair. Moreover, [32] analyze the redundancy of information in heterophilous graphs by training a model, SGAT, able to discard useless edges in the network and perform attention-weighted message passing based on the remaining edges. On some heterophilous benchmarks, up to 80% of the edges are found to be redundant or not impactful for model performance. DMP [33] extends the standard label-based assortativity measure to an attribute-based assortativity measure by providing insights into the connectivity among nodes that share the same features. Motivated by the observation that different features show different levels of heterophily, the authors propose to learn a weight for each feature in the node embeddings and aggregate messages using these attribute-wise weights. On a different note, WRGNN [17] models the diversity of relations among nodes by leveraging a multi-relational graph with different types of edges that are managed by different filters in the neural network. CPGNN [34] estimates a compatibility matrix (modeling how likely it is for nodes belonging to one class to be connected to nodes belonging to another class) and exploits it for message propagation. Also GGCN [10] moves in a similar direction. Indeed, starting from the analysis of how node representations change in a multi-layer GCN when heterophily and node degree vary, the authors propose two key design choices to improve performance on heterophilous benchmarks. First, they employ degree corrections, as they realize that low degree nodes are more prone to oversmoothing. Secondly, they employ signed messages, i.e. based on cosine similarity between node representations, they multiply the message by a negative sign if the nodes are interpreted as very different in the embedding space. The different variations of sheaf diffusion-based models proposed in [35] improve the performance by employing non-symmetric restriction maps, which can be interpreted as the possibility for nodes to be connected by an edge with a negative weight. In LWGNN [36], the authors observe that label-wise propagation is more distinctive on heterophilous graphs. Since labels are not known, propagation is driven by pseudo-labels estimated for each node. MWGNN [37] proposes separate graph convolutions, each focusing on a specific feature of the graph. In particular, three convolutions are learnt: one to encode node features, another one for topological structure and the last one to represent positional identity. These three matrixes are then adaptively merged with an attention mechanism, since their contribution to the representation learning might differ depending on the characteristics of the graph.

Ego-neighbor separation

Another class of methods elaborates on the observation that if a node is significantly different from its neighborhood, using the average as aggregation strategy might lead to significant information loss, since the differences are smoothed into a unique noisy embedding. This can be seen also from a spectral point of view [9], since average aggregation is a low-pass filter that cuts high-frequency information that could, however, be fundamental in heterophilous settings. Therefore, a design choice that has been proved to help in this sense is to separately create an embedding for the target node and for the neighborhood and then concatenate them to generate a final embedding in which information coming from both sources is effectively preserved. This design choice is exploited in similar forms by several works [9, 17, 10, 31, 23].

Inter-layer combination

A final class of approaches aims at exploiting information collected by the GCN at different layers. In particular, starting from the observation that lower layers capture local interactions whereas deeper layers capture global information, the embeddings at different layers are concatenated at the end and exploited to perform predictions. This can be achieved in practice through different implementations. H2GCN [17] concatenates the embeddings of different layers to create the final embedding. GCNII [38] adds skip-connections with the first layer at each layer and manages to create deep architecture without incurring in the oversmoothing issue. GPR-GNN [39] leverages the generalized PageRank algorithm to allow for inter-layer combinations of nodes. HLP [40] revisits GPR-GNN by using truncated SVD of the features.

3.1.3 Categorization of present work

The models proposed in this work extend some of the approaches described above and they can be framed within the proposed classification of existing methods. For GATH, the choice to leverage the attention mechanism is related to its ability to assign variable weights to different node pairs depending on how relevant they are to each other. This idea is an implementation of the adaptive message aggregation framework described in Section 3.1.2, since message aggregation is not uniformly distributed across all neighboring nodes. Moreover, GATH extends the neighborhood to neighbors up to the k-hop, thus applying the potential neighbor discovery principle described in Section 3.1.1. However, with respect to existing methods, GATH exploits both structural and attribute information in the calculation of the attention weights, thus allowing for more flexibility. In conclusion, the skip-connections introduced in both GATH and GCNH can be considered as an implementation of the ego-neighbor separation framework described in Section 3.1.2, as the embeddings for the neighborhood and the center node are computed separately and aggregated in a later stage.

3.2 Is heterophily the real problem?

After discussing about possible GNN improvements to increase performance on heterophilous graphs, we now discuss the relation between graph heterophily and GNN performance.

3.2.1 GNN performance on heterophilous graphs

The fact that GNN performance is hampered by graph heterophily is supported by experimental evidence. As a motivating example, Figure 3.1 shows the classification accuracy of three simple baselines (MLP, GCN and GAT) on synthetic datasets with variable homophily ratio. Further details on the datasets are available in Section 4 and Appendix A.1 reports information about the experiments for the baselines. We can notice that GCN and GAT performance is clearly correlated with the homophily ratio of the graphs, and, specifically, classification accuracies on heterophilous graphs are significantly worse than those of MLP, which does not take any structural information into account and only relies on node features.

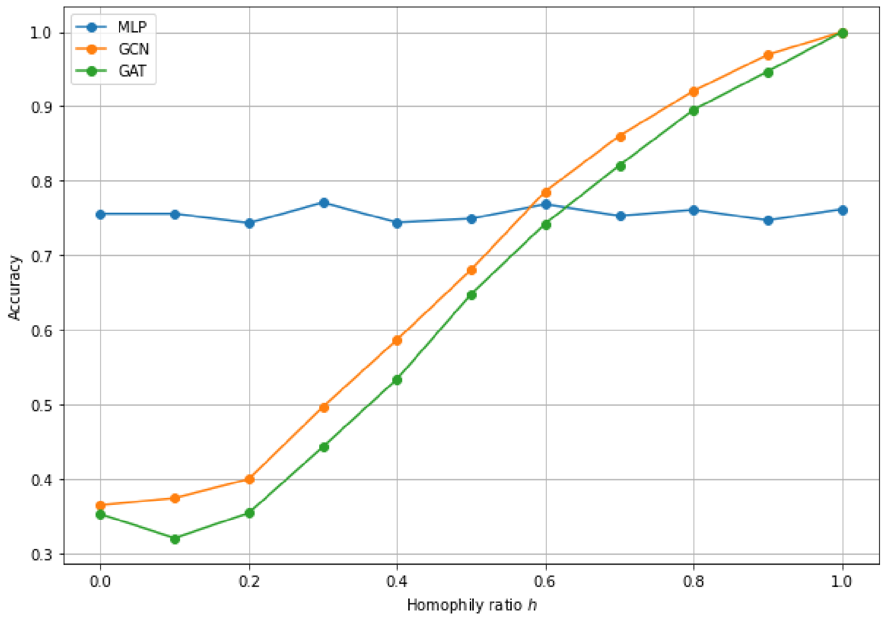


Figure 3.1: Node classification accuracy for three baseline models (MLP, GCN and GAT) on synthetic datasets with variable homophily ratio. More details about the hyperparameters used for the experiments are available in Appendix A.1.

Nevertheless, the situation is different on real-world datasets. Table 3.1 reports the classification accuracy for the same three baseline models (MLP, GCN and GAT) on the real-world datasets analyzed in the work. For `Cornell`, `Texas`, `Wisconsin` and `Film`, MLP outperforms GCN and GAT, as expected since they show strongly heterophilous behaviors. Also the results on `Cora` and `Citeseer` reflect the expectations, as they are homophilous graphs and GCN and GAT can take advantage of the useful graph information to outperform the MLP. However, results for `Chameleon` and `Squirrel` are not consistent with the previous observations: despite the low homophily ratio, the best classification accuracy is achieved by the GCN, which significantly outperforms the MLP. This behavior contradicts the assumption that heterophily necessarily leads to poor GNN performance and motivates the analysis reported in this thesis.

	Cornell	Texas	Wisconsin	Film	Chameleon	Squirrel	Cora	Citeseer
h	0.30	0.11	0.21	0.22	0.23	0.22	0.81	0.74
GCN	50.27 ± 7.57	57.03 ± 5.05	50.98 ± 4.88	23.27 ± 0.94	**67.43±1.95**	**50.49±1.54**	84.29 ± 0.98	73.25 ± 1.42
GAT	61.89 ± 5.05	52.16 ± 6.63	49.41 ± 4.09	27.44 ± 0.89	60.26 ± 2.50	40.72 ± 1.55	**87.30±1.10**	**76.55±1.23**
MLP	**81.89±6.40**	**80.81±4.75**	**85.29±3.31**	**36.53±0.70**	46.21 ± 2.99	28.77 ± 1.56	75.69 ± 2.00	74.02 ± 1.90

Table 3.1: Node classification accuracy for baseline models on datasets with various edge homophily ratio h. Best results are in **bold**. Results for MLP and GAT are taken from [35], results for GCN are obtained with the hyperparameter configuration described in Appendix A.2

3.2.2 Cross-Class Neighborhood Similarity

Motivated by the good performance of the GCN on real-world heterophilous graphs, [20] analyze which graph properties affect GCN performance beyond homophily, finding out that there is a specific category of heterophilous graphs on which the GCN can perform well. A toy example of this class of graphs is depicted in Figure 3.2, where nodes belong to two classes (*blue* or *orange*) and have a single feature (0 or 1). We can notice that the graph is perfectly heterophilous, since every node is connected only to nodes of the other class (homophily ratio $h = 0$). However, a GCN is still able to perfectly classify the nodes in the graph, since the neighborhood representation is the same for every node belonging to the same class (i.e. every node of class *orange* has 4 neighbors with feature 0 and every node of class *blue* has 4 neighbors with feature 1). Therefore, it can be concluded that the homophily ratio by itself is not enough to predict whether a simple GCN performs well on the graph or not.

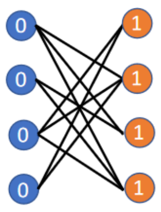


Figure 3.2: Toy example of heterophilous graph on which a GCN can achieve perfect performance. Colors are classes, numbers are features. The image is taken from [20].

More in general, thanks to its formulation, a GCN is able to capture the distribution of features in the neighborhood of a target node. Therefore, under the assumption that features of nodes belonging to the same class are sampled from the same distribution, a GCN performs well on nodes for which the neighborhood distribution is informative for their label. As an example, Figure 3.3 shows two nodes with the same neighborhood label distribution (i.e. the percentage of neighbors belonging to each class is the same for the two nodes). Although these two nodes are perfectly heterophilous, a GCN is still able to recognize the class *blue* as characterized by the specific neighborhood distribution $[orange : \frac{1}{3}, yellow : \frac{1}{3}, green : \frac{1}{3}]$.

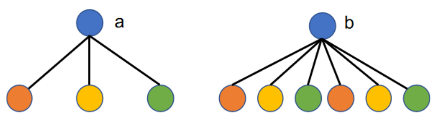


Figure 3.3: Two nodes with the same neighborhood distribution. Classes are colors. The image is taken from [20].

According to this observation, [20] provide a theoretical proof that, under some assumptions such as a relevant correlation between node labels and node attributes, a GCN performs well in graphs where the neighborhood label distribution is informative for the class of a node. It can be noticed that this formulation is in line with the empirical results that show good GCN performance on homophilous graphs. Indeed, in homophilous graphs the neighborhood label distribution can be interpreted as a skewed distribution centered on the class of the target node, since nodes belonging to the same class tend to connect. Therefore, it is safe to assume that a GCN performs well on homophilous graphs, but heterophily is not a sufficient condition to assume bad performance.
To characterize neighborhood distributions in real graphs, and therefore provide an additional explanation for GCN performance, [20] introduce the notion of *Cross-Class Neighborhood Similarity* (CCNS), defined as follows:

$$s(c, c') = \frac{1}{|V_c||V_{c'}|} \sum_{i \in V_c, j \in V_{c'}} \cos(d(i), d(j)) \tag{3.1}$$

where $c, c' \in C$ are classes, V_c is the set of nodes in class c, $d(i)$ is the empirical histogram over $|C|$ classes of node i's neighbors' labels and $\cos(\cdot, \cdot)$ denotes the cosine similarity function.

Figure 3.4 shows the values of the CCNS for all class pairs on the analyzed datasets. It should be noticed that distinctive per-class neighborhood label distributions correspond to high values on the diagonals of the matrixes. Therefore, it is

expected to observe high values on the diagonal for the datasets on which GCN achieves good performance.
By observing Figure 3.4, some considerations can be made. First of all, homophilous graphs (`Cora` and `Citeseer`) have higher values on the diagonal of the table, since nodes of one class tend to be connected to nodes of the same class, and this represents a distinctive neighborhood distribution pattern in the graph, thus explaining the good GCN performance. For heterophilous graphs, different situations can occur. In `Chameleon`, neighbor distribution patterns differentiate classes 0 and 1 from classes 2, 3 and 4, but the distinctions within these groups are less evident. In `Squirrel` and `Film`, intra-class similarity (i.e. values on the diagonal) and inter-class similarity assume similar values, which would lead to the expectation of bad GCN performance. Tables for `Cornell`, `Texas` and `Wisconsin` are not informative given the reduced sizes of the graphs (distributions for classes depend only on a small number of nodes).

3.2.3 Limitations of CCNS

Although CCNS is proved to be a useful metric to model good GCN performance, there are still some limitations that prevent it from effectively explaining some results on real-world datasets.

Node degrees

[41] identify one limitation of CCNS in the lack of an analysis of the node degrees. As a matter of fact, they show that the similarity among labels in the neighborhood can be reliably estimated only if node degrees are reasonably high, which occurs in the experimental setting proposed in [20], but might not happen in real graphs. Through experiments on synthetic datasets, [41] show how graphs with low homophily and low-degree nodes are still subject to bad GCN performance, regardless of the value of CCNS. Moreover, they point out that the issues related to low-degree nodes and label neighborhood similarity are relevant only in heterophilous settings, since under homophily the negative effects are strongly mitigated. Thus, they claim that "heterophily is still a challenging problem for GNN models, including GCN" ([41]).

GCN performance on `Chameleon` and `Squirrel`

We point out another concern about CCNS by observing the classification performance of different models on real graphs. Indeed, from Table 3.1 it is possible to observe that GCN largely outperforms MLP on `Chameleon` and `Squirrel` despite these datasets being highly heterophilous. The measure of CCNS does not explain this behavior for two reasons. First of all, good GCN performance due to distinctive neighborhood patterns should correspond to larger CCNS values on the diagonal of the matrixes, but this is not observed in Figure 3.4. Moreover, one important assumption behind the theoretical proofs in [20] is that features for each class are sampled from the same distribution, which implies a correlation between node features and class labels. However, this assumption does not hold for `Chameleon` and, especially, `Squirrel`, where the performance of MLP (only based on features) is bad (for `Squirrel`, features allow a very small performance improvement with respect to random guessing). Therefore, further investigations are required to deeply understand what other graph properties allow the GCN to achieve good performance on these graphs.

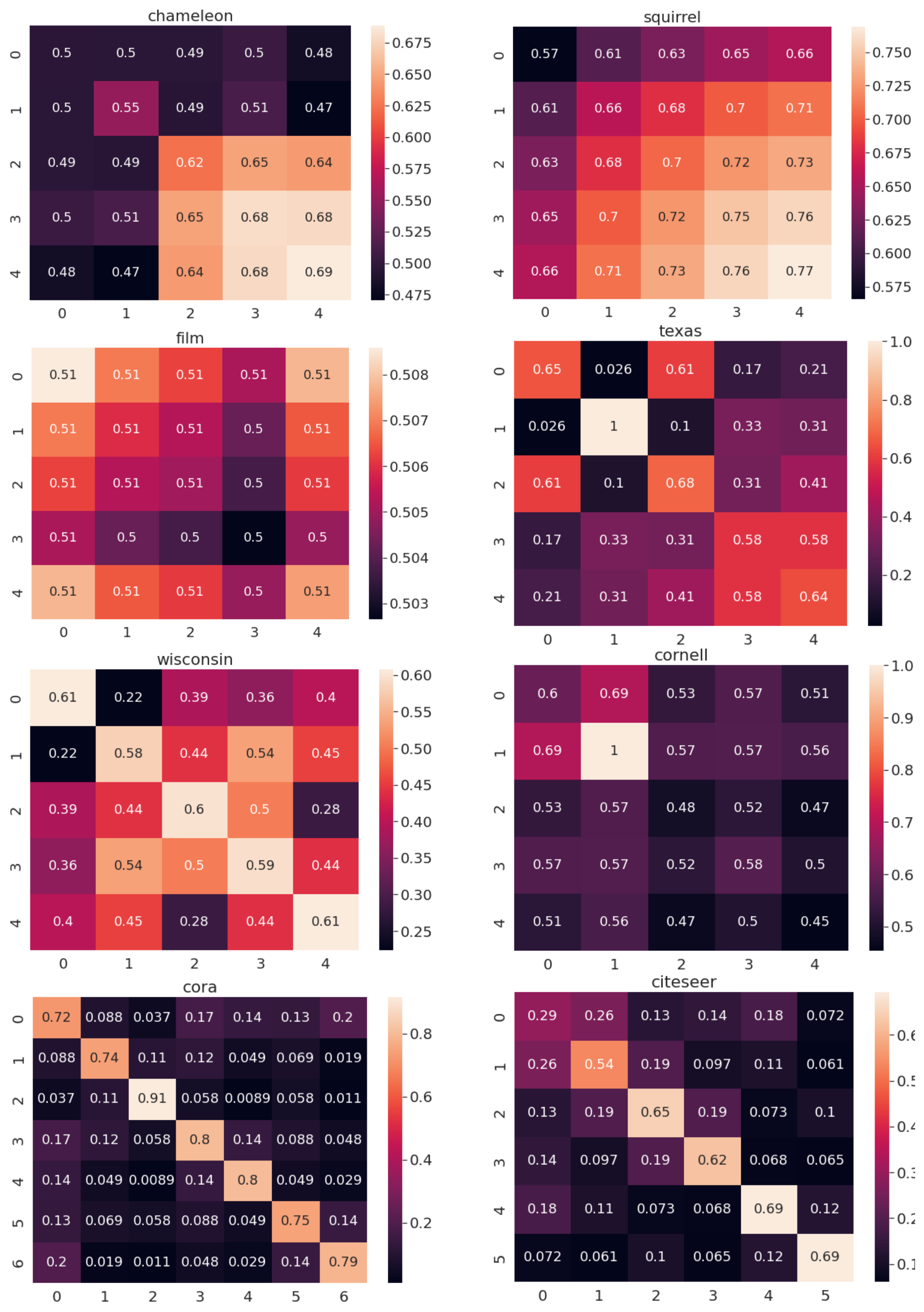


Figure 3.4: Cross-Class Neighborhood Similarity matrixes of different datasets

Chapter 4

Datasets

In this chapter we present the real-world and synthetic datasets used for evaluation throughout the thesis.

4.1 Real-world datasets

Most of the experimental evaluations in this work are based on 8 real-world datasets coming from different scenarios and showing various levels of heterophily. These datasets are commonly used in most of the works dealing with heterophilous graphs. In this section, we describe their origin and we present their main characteristics. Table 4.1 shows some statistics. We can point out that the graphs are in general small. Indeed, `Cornell`, `Texas` and `Wisconsin` only have about 200 nodes and the biggest one, `Film`, has 7600 nodes, which is still a low number compared to the million of nodes of large-scale graphs used in other settings. `Chameleon` and `Squirrel` are dense graphs, whereas all the others have on average low node degrees. `Cora` and `Citeseer` are representative of homophilous settings, since their homophily ratio is high, whereas all the others are strongly heterophilous. All graphs have large feature dimensionality.

Benchmark	`Cornell`	`Texas`	`Wisconsin`	`Film`	`Chameleon`	`Squirrel`	`Cora`	`Citeseer`
#Nodes	183	183	251	7,600	2,277	5,201	2,708	3,327
#Edges	280	295	466	26,752	31,421	198,493	1,433	3,703
#Classes	5	5	5	5	5	5	7	6
#Features	1,703	1,703	1,703	931	2,325	2,089	1,433	3,703
Homophily h	0.30	0.11	0.21	0.22	0.23	0.22	0.81	0.74

Table 4.1: Statistics of real-world datasets.

We now present into more details what each dataset represents and where it comes from.

- **Texas, Wisconsin and Cornell** are webpage datasets collected from the computer science departments of different universities by Carnegie Mellon University within the WebKB project[1]. Nodes represent web pages and edges are hyperlinks between them. Node features are bag-of-words representations of the web pages, which are manually classified into five categories: student, project, course, staff, faculty. The datasets are taken from the public code of [18].
- **Film**, also referred to as **Actor**, is the actor-only induced subgraph of the film-director-actor-writer network [42]. Nodes correspond to Wikipedia pages of actors and edges denote the co-occurrence of two actors on the same page. Node features are some keywords in the Wikipedia pages and labels are assigned by [18] based on words of the actors' Wikipedia pages. This dataset is taken from the public code of [18].
- **Chameleon and Squirrel** are Wikipedia pages on the specific topics of chameleons and squirrels. They were collected by [43] and pre-processed by [18]. Nodes are Wikipedia pages and edges are mutual links between them. Node features indicate the presence of informative nouns in the Wikipedia pages. Nodes are classified into five categories based on the average monthly traffic on the web page. The datasets are taken from the public code of [23].
- **Cora and Citeseer** are standard citation networks where nodes represent papers and edges represent citations of one paper by another [44, 45]. Node features are bag-of-words representations of papers and labels are the academic topics of the papers. These datasets are treated as undirected and they are taken from the public code of [18].

4.2 Synthetic datasets

Some experiments in this work are also based on synthetically generated datasets with variable homophily level. These are taken from [9], that define a graph generation strategy similar to [21]. After choosing in advance the number of classes $|C|$, the number of nodes $|V|$ and the class compatibility matrix H, a small initial graph is taken as starting point. Then, new nodes are added one by one until the desired size of the graph is reached, and for each newly added node i the probability

[1]http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb/

Benchmark	`syn-cora`
#Nodes	1490
#Edges	2965 to 2968
#Classes	5
Features	`Cora`
Homophily h	[0, 0.1, ..., 1]
Degree Range	1 to 94
Average Degree	3.98

Table 4.2: Statistics for the synthetic dataset.

of an edge p_{ij} between i and any other node in the graph j is proportional to both the degree of the existing node d_j and the class compatibility $H_{y_i y_j}$. In this way, the degree distribution of the generated graph follows a power law and heterophily can be tuned through the compatibility matrix H. Nodes are randomly assigned to a class during generation and the features for each node are sampled from feature vectors of nodes of the corresponding class in the real benchmark (`Cora`). Class sizes of the synthetic graphs are smaller than the class size of the real graphs. Table 4.2 shows some statistics for the generated synthetic datasets. For each of the specified homophily levels, three different graphs are generated.

Chapter 5

2NCS: modeling GCN performance

As discussed in Section 3.2, it has been shown that often lower levels of homophily are correlated with worse GNN performance. However, it has been pointed out that this correlation does not hold for all graphs with high heterophily. As a first effort towards a better description of which graph properties affect GCN performance, [20] introduce a new measure to characterize the distribution of labels in the neighborhood of a node. This is shown to be related to the ability of a GCN to distinguish node classes. Indeed, if the distribution is similar for nodes belonging to the same class, a GCN is able to correctly classify the nodes of that class, regardless of their homophily level. To quantify the similarity of the distributions, they introduce a new metric: Cross-Class Neighborhood Similarity (CCNS). However, also CCNS has some limitations, as discussed in Section 3.2.3. This chapter presents a new metric, 2-hop Class Neighborhood Similarity (2NCS), to measure the average over neighbors of the percentages of their neighbors with the same label as the target node. An intuitive theoretical analysis of how a GCN learns and experimental evaluations show that 2NCS is related to the ability of a GCN to correctly classify a node in a graph.

5.1 Motivation

Analyzing GCN performance in relation to different graph properties represents an important step in several directions. First of all, it is helpful to gain a deeper understanding of how a GCN works, what kind of structural information it can successfully encode in a graph and what properties affect its performance. Moreover, defining graph properties that allow for good GCN performance can be helpful to design new GNN architectures that solve existing limitations.

5.2 2NCS: 2-hop Neighbor Class Similarity

This section introduces 2-hop Neighbor Class Similarity (2NCS), a metric to characterize a graph property that is relevant to GCN representation capabilities. The metric is developed by making observation on a simplified GCN model and analysing its learning process.

5.2.1 The simplified GCN model

One of the main limitations of CCNS is the assumption that the features for nodes belonging to the same class are drawn from the same distribution. Although this seems reasonable in theory, it is not observed in some real-world graphs, such as `Squirrel`, on which the graph-unaware MLP is hardly able to improve on random guessing for node classification. Moreover, in several real settings node features might not be available [46], and the GNN models can be extended to handle also these situations [34].
This motivates the introduction of a simplified GCN model, in which the feature matrix is equal to the identity matrix (i.e. $X = I$, so node features are not considered). Moreover, the simplified GCN model contains only one layer, it does not have a non-linearity and the adjacency matrix is not row-normalized, since normalization does not affect the class probability distribution with just one layer. Overall, the simplified GCN model can be expressed as

$$H = \text{softmax}(\tilde{A}XW) = \text{softmax}(\tilde{A}IW) = \text{softmax}(\tilde{A}W) \tag{5.1}$$

where $H \in \mathbb{R}^{n\times|C|}$ are the class probabilities for each node, $\tilde{A} = A + I$ is the adjacency matrix with added self-loops and $W \in \mathbb{R}^{n\times|C|}$ is a learnable weight matrix. These modifications allow for a more thorough analysis of the model representation capabilities and a characterization of what graph structural information affect the model performance.

5.2.2 Learning process of the simplified GCN model

To understand which properties of the graph structure affect the learning process of the simplified GCN model, it is important to understand how node embeddings are learned during model training. We use the graph represented in Figure 5.1 as an example.
In this graph, each node u is characterized by a row in the adjacency matrix A_u that indicates its neighbors. For the matrixes W and H in Equation (5.1), each row W_u can be interpreted as an embedding of u before aggregating over the neighbors, whereas H_u is the embedding of u after aggregation, also corresponding to the class

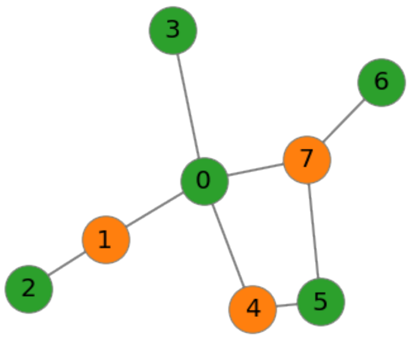


Figure 5.1: Example of a graph to model simplified GCN learning. Different colors indicate different node labels.

probabilities for the node. Taking node 0 as example, its final embedding can be written as

$$H_0 = \text{softmax}(W_0 + W_1 + W_3 + W_4 + W_7) \tag{5.2}$$

Therefore, the class probabilities for node 0 depend on the embeddings W of the node itself and of its neighbors.
Then, it is important to understand which terms of the loss affect each W_u. W_7, for example, contributes to the final embeddings H of nodes 0, 5 and 6, plus node 7 itself. Therefore, it is trained on the labels of the nodes whose embeddings it contributes to, so, for this example, one *orange* label and three *green* labels. Since the training objective is the minimization of the loss, embeddings are updated such that the probability of the observed label is increased and the probabilities of the non-observed labels are decreased. Therefore, it can be assumed that, after training, W_7 will have a higher value on the cell corresponding to class *green*, since *green* labels are more frequent in the terms of the loss containing W_7. Thus, for a generic node u, we can state that, in expectation, its embedding W_u after training will contain a higher value for the class it has observed more often during training, i.e. the most common class among the node itself and its neighbors.
Given these considerations, we can now analyze what elements in the graph affect the class probabilities of node 0. As a matter of fact, each term in Equation (5.2) describes the class distribution of the neighbors of the nodes each embedding corresponds to. Therefore, we can observe that H_0 depends not only on node 0's neighbors, but also on the neighbors of its neighbors, i.e. its 2-hop neighbors.

However, just considering the most common class between 1 and 2-hop neighbors might not be informative enough. Indeed, by observing the graph in Figure 5.2, we can notice that, for node 0, the most common class among nodes of the first and second hop is *orange*, which would lead to the hypothesis that a simplified GCN

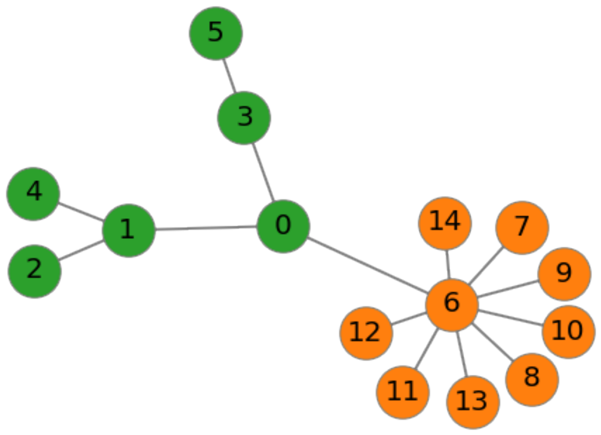


Figure 5.2: Example of a graph where the most common label in the 1 and 2-hop neighborhood is not informative. Different colors indicate different node labels.

cannot correctly classify node 0. However, we can write the expression for the class probabilities of node 0 as

$$H_0 = \text{softmax}(W_0 + W_1 + W_3 + W_6) \tag{5.3}$$

where W_0, W_1 and W_3 mostly observe class *green* during training, whereas W_6 favors class *orange*. Therefore, H_0 will have higher probability for class *green*. Hence, we can assume that the simplified GCN is able to correctly classify it. More in general, we can claim that the simplified GCN can correctly classify a node u if the majority of its neighbors have, for the most part, neighbors with the same label as u.

5.2.3 2NCS: a metric for graph structural properties

Given the above considerations, we introduce a new metric to quantify the described property of a node. The metric is called 2-hop Neighbor Class Similarity (2NCS) and, for a single node u, we define it as

$$2NCS_u = \frac{1}{|N'(u)|} \sum_{v \in N'(u)} \frac{|\{z : z \in N'(v) \setminus \{u\} \wedge y_z = y_u\}|}{|N'(v)| - 1} \tag{5.4}$$

where $N'(u)$ denotes the set of neighbors of u and the node u itself. Note that node u is removed from the count of neighbors with the same label since the goal is to understand whether its label can be correctly predicted given the rest of the graph structure. We can also interpret this problem as the training of a simplified GCN on the whole graph but node 0 and the evaluation on node 0 as test set. In this

case, the label of node 0 is not available during training, and therefore it would be inaccurate to include it in the 2NCS computation. The possible values of 2NCS lie in the interval [0,1], where 2NCS=0 means that all 1-hop and 2-hop neighbors of a node have different labels from it and 2NCS=1 means that all 1-hop and 2-hop neighbors of a node have its same label.
We can also define the graph-level 2NCS as

$$2NCS = \frac{1}{|V|} \sum_{u \in V} 2NCS_u \tag{5.5}$$

Considering the graph property it represents, we can claim that a simplified GCN can correctly classify nodes with high 2NCS, and the graph-level 2NCS is informative of how well a simplified GCN performs. Moreover, in most cases the performance of the simplified GCN is comparable with the performance of the standard GCN, and, therefore, 2NCS can be informative also for the standard GCN.

5.3 Analysis and evaluation

This section evaluates how informative 2NCS is in understanding whether a graph contains structural information that is useful for a GCN to perform classification. To this end, we present one example of artificial graph on which 2NCS is more informative than homophily and CCNS to predict GCN performance. Then, we report the values of 2NCS on real-world datasets, both at node level and graph level, and we analyze the relation with GCN performance.

5.3.1 2NCS on artificial graphs

To begin with, we analyze 2NCS on artificially designed graphs in order to understand the edge cases of the metric and provide examples of graphs where 2NCS is more informative about GCN performance than the other existing metrics, namely homophily ratio and CCNS. We use as an example the graph in Figure 5.3. In this graph, node features are not available, thus we set the feature matrix to identity matrix ($X = I$). The goal of the example is to understand whether a GCN that is trained on all the other labels except for the one of node 0 can correctly classify node 0.

Table 5.1 reports the values of homophily ratio, CCNS and 2NCS for node 0 in the proposed graph. We can observe that the values for homophily and CCNS are low, indicating that, according to them, the graph structure does not help a GCN to correctly classify node 0. However, we performed experiments to prove that a GCN succeeds in the proposed task. Remarkably, the high value of 2NCS for node 0 explains this behavior. Indeed, a GCN correctly classifies node 0 because it is

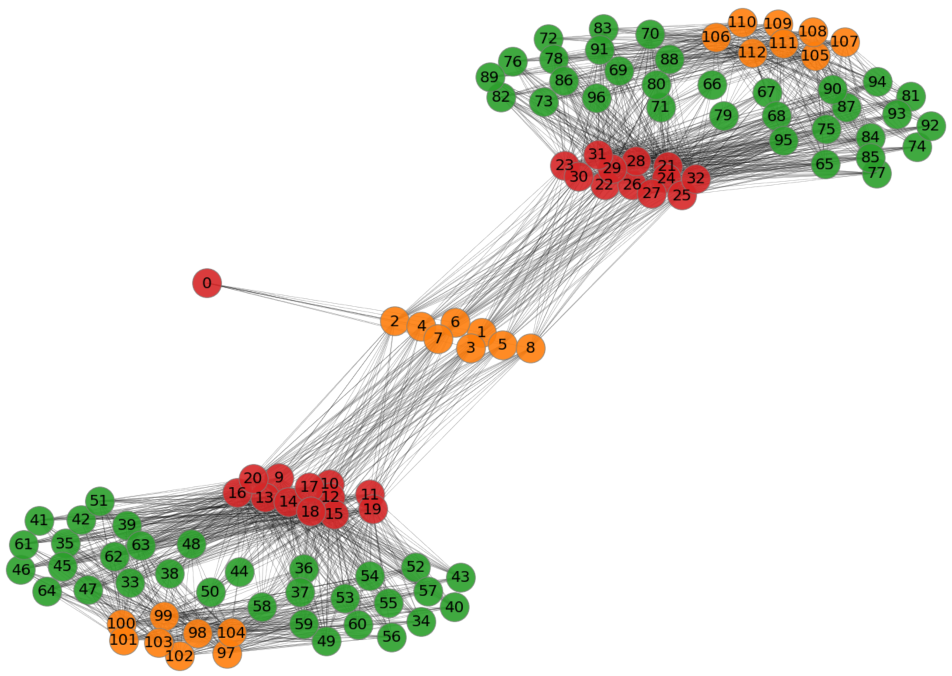


Figure 5.3: Example of a graph where 2NCS is more informative than homophily and CCNS for GCN performance. Different colors indicate different node labels. Node 0 is connected to the orange nodes 1-8, nodes 1-8 are densely connected to the red nodes 9-32, nodes 9-20 are densely connected to the green nodes 33-64 which are also densely connected to the orange nodes 97-104, nodes 21-32 are densely connected to the green nodes 65-96 which are also densely connected to the orange nodes 105-112. Nodes have no features.

Metric	Node 0
h (Eq. (2.9))	0
$CCNS$ (Eq. (3.1))	0.24
$2NCS$ (Eq. (5.5))	0.85

Table 5.1: Homophily, CCNS and 2NCS for node 0.

connected to nodes 1-8, which are connected to nodes of the same class as node 0. This pattern allows for good GCN performance, but it is not described by the other two presented metrics.

	Cornell	Texas	Wisconsin	Film	Chameleon	Squirrel	Cora	Citeseer
h (2.9)	0.30	0.11	0.21	0.22	0.23	0.22	0.81	0.74
$2NCS$ (5.5)	0.35	0.28	0.30	0.21	0.36	0.26	0.79	0.70
GCN	50.27 ± 7.57	57.03 ± 5.05	50.98 ± 4.88	23.27 ± 0.94	67.43 ± 1.95	50.49 ± 1.54	84.29 ± 0.98	73.25 ± 1.42

Table 5.2: 2NCS measures for eight real-world datasets. GCN performance on the graphs is reported for comparison.

5.3.2 2NCS on real-world graphs

Graph-level 2NCS

Table 5.2 reports the values for 2NCS for the eight real-world datasets used in the work. We can observe that high values of homophily correspond to high values of 2NCS, since nodes tend to be connected to nodes with the same label and, therefore, nodes tend to share their label also with their 2-hop neighbors. However, 2NCS is able to better discriminate among graphs with low homophily ratio. Indeed, `Film` has almost the same homophily ratio as `Wisconsin`, `Chameleon` and `Squirrel`, even though GCN performance is much worse on it than on the other graphs. 2NCS, conversely, is able to better describe this behavior, since `Film` has a lower 2NCS value than the other mentioned datasets.

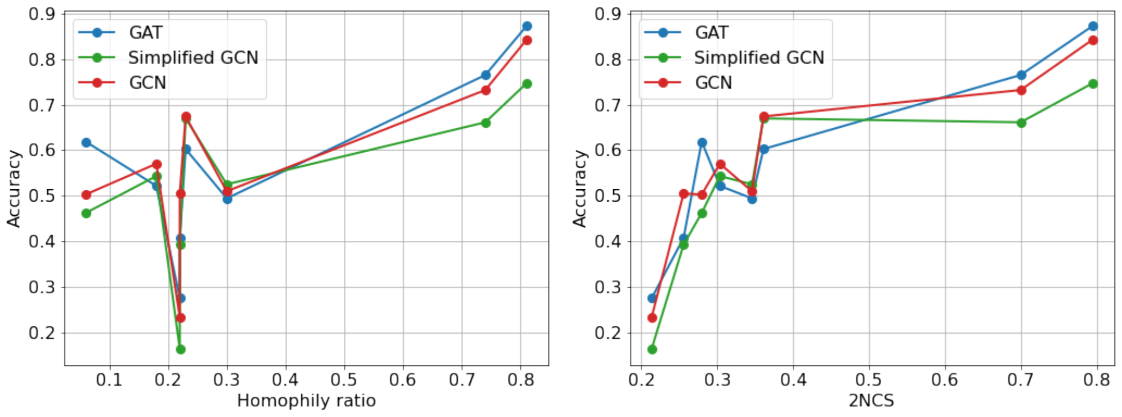
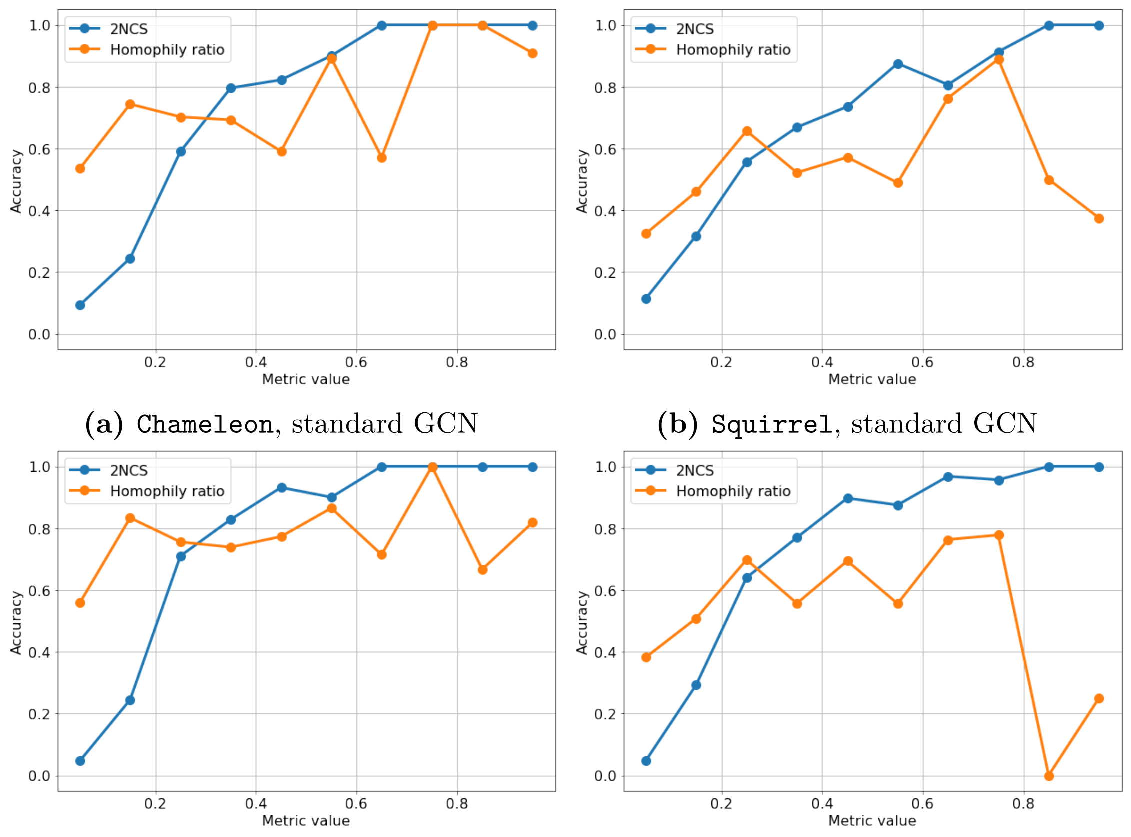


Figure 5.4: Comparison between GCN performance versus homophily ratio and versus 2NCS. The simplified GCN is the one described in Section 5.2.1, while for the standard GCN the results are the same as in Table 5.2.

Figure 5.4 reports the performance of different GNNs on eight real-world datasets ranked by increasing value of, respectively, homophily ratio and 2NCS. There is no clear correlation between GNN performance and homophily ratio h, since the worst performance does not correspond to the lowest value of h. The correlation among

GNN performance and 2NCS, on the other hand, appears to be stronger, although not perfect. Moreover, we can notice that the performance of simplified GCN and standard GCN is comparable, and this justifies the extension of the considerations made on the simplified GCN to the standard GCN.



(a) `Chameleon`, standard GCN **(b)** `Squirrel`, standard GCN

(c) `Chameleon`, simplified GCN **(d)** `Squirrel`, simplified GCN

Figure 5.5: GCN accuracy versus node-level homophily ratio and node-level 2NCS.

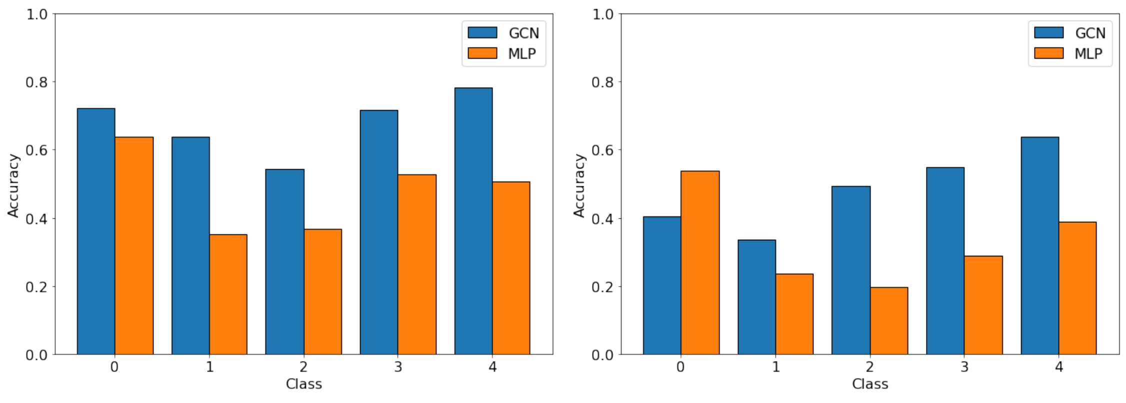
Node-level 2NCS

As for the homophily measures, it is hard to encode the complex and various structure of a graph in a single value of 2NCS. Hence, looking at node-level values for this metric can provide more insights about local patterns. Figure 5.7 shows the distributions of node-level 2NCS for the different graphs. We can observe that homophilous graphs (`Cora` and `Citeseer`) have high 2NCS for most of their nodes. For heterophilous graphs, behaviors are more various: `Chameleon` shows a similar distribution of 2NCS values between 0 and 0.7, whereas `Film` and `Squirrel` have

most of their nodes with 2NCS values between 0 and 0.4.
It is also valuable to compare node-level 2NCS values with GCN classification accuracy. In particular, in Section 5.2.3 we claim that a GCN can correctly classify nodes with high 2NCS values. To show that this claim holds experimentally, Figure 5.5 reports the GCN classification performance with respect to different local 2NCS measures. The figure shows that, for the simplified GCN, there is a clear correlation between classification accuracy and node-level 2NCS, whereas this correlation is much less evident with the homophily ratio of the nodes. We can make a similar consideration for the standard GCN: also in this case, indeed, 2NCS is a better indicator of GCN performance than the homophily ratio. Therefore, these experimental results support the claim that a GCN can correctly classify nodes with high 2NCS value.

Class-level 2NCS

We can observe the ability of 2NCS to capture graph structural properties that are useful to GCN also at class-level.



(a) `Chameleon` **(b)** `Squirrel`

Figure 5.6: Per-class accuracy for MLP and GCN on `Chameleon` and `Squirrel`.

Figure 5.6 shows the accuracies of two different models (MLP and GCN) for nodes belonging to different classes on two datasets, `Chameleon` and `Squirrel`. The MLP performance is used as a measure of how informative the node features are for classification, and the comparison between MLP and GCN performance provides insights into how useful the graph structure is.
For example, Figure 5.6a shows that on `Chameleon` GCN significantly improves MLP performance on classes 1 and 4. This behavior is consistent with the per-class average 2NCS reported in Table 5.3. Indeed, 2NCS values for classes 1 and 4 are particularly high, indicating that nodes with these labels are well characterized by

the surrounding graph structure.
We can make similar considerations on `Squirrel`. In particular, Figure 5.6b shows that for class 0 MLP outperforms GCN, which is consistent with the low value of 2NCS for class 0 reported in Table 5.3 (indeed, the structural information for nodes of this class is probably misleading and introduces noise that makes GCN perform worse than MLP). Class 4, on the other hand, witnesses a large performance improvement for GCN with respect to MLP, and this is also in agreement with the high 2NCS value for the class, indicating that the graph structure is informative for nodes belonging to class 4.

Class	0	1	2	3	4
`Chameleon`	0.34	0.39	0.29	0.40	0.43
`Squirrel`	0.15	0.19	0.24	0.28	0.40

Table 5.3: Per-class 2NCS values for `Chameleon` and `Squirrel`.

5.4 Limitations

Although 2NCS has been shown to be useful to characterize specific graph structural properties that affect GCN performance, we can identify also some limitations. The main weakness is that it completely neglects the impact of node features, since it only considers the label distribution in the graph. Even though this is advantageous in scenarios where features are not informative or are not present at all, it becomes detrimental in graphs where features play an important role. Moreover, the examples and the experimental evidence provided in this chapter confirm that high values of 2NCS generally correspond to good GCN performance, whereas the opposite cannot be shown. Hence, this leads to the conclusion that 2NCS represents a *sufficient but not necessary condition* for good GCN performance.

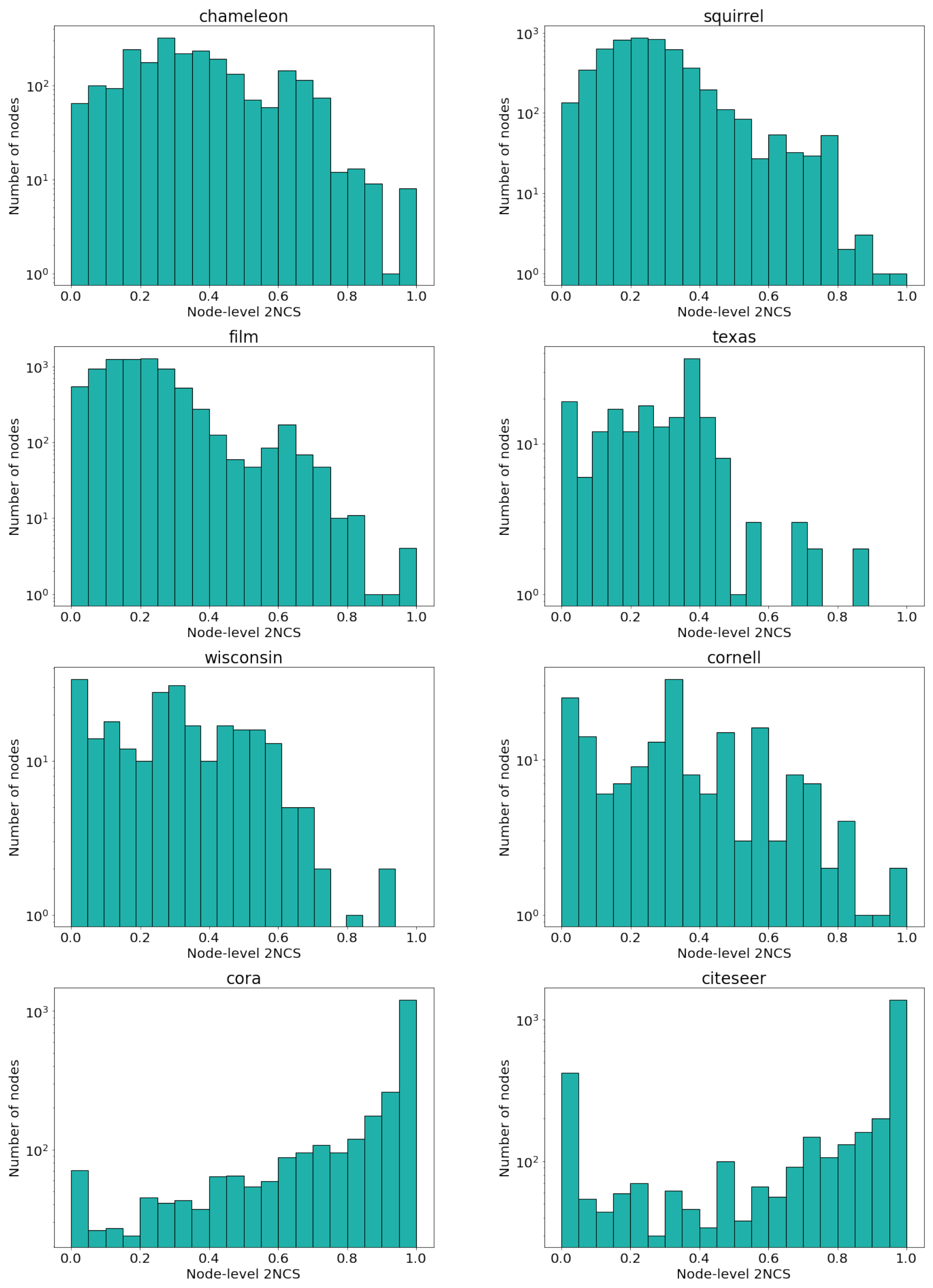


Figure 5.7: Node-level 2NCS distribution of different datasets

Chapter 6

GATH and GCNH: GAT and GCN for Heterophily

Despite the fact that heterophilous networks are present in several real-world applications, standard GNN architectures fail to achieve satisfactory performance on different heterophilous benchmarks, both real and synthetic, as discussed in Section 3.2. Although high levels of heterophily do not necessarily imply bad GNN performance, as pointed out in Section 5, it is still relevant to improve the GNN architectures in order to extend their representation capabilities also on the heterophilous graphs that turn out to be problematic. To this end, in this chapter we present two new GNN-based models, GATH and GCNH. Both models incorporate intuitive designs that improve the performance of basic GNNs on heterophilous graphs. This chapter describes the architecture of the two models, whereas Chapter 7 reports the experimental evaluation of their performance on real and synthetic datasets, showing their capability to achieve competitive results with respect to state-of-the-art models.

6.1 GATH

The first GNN-based model proposed in this thesis is GATH (GAT for Heterophily). Similarly to GAT, GATH leverages an attention mechanism to adaptively learn a weight between two nodes that is used as coefficient during the message aggregation phase. However, we adopt additional design choices that allow for much more flexibility than GAT, thus resulting in large performance improvements on heterophilous settings. In particular, three main characteristics impact on the model behavior. First of all, we extend the attention formulation with respect to GAT to improve expressiveness. Secondly, we expand the neighborhood also to non-connected nodes to aggregate useful information that might be located in other portions of the graph. Third, we employ skip-connections to separate the computation of the embeddings of self-node and neighborhood, which are merged only as last stage. Each of these components is better described and analyzed in the following paragraphs. Moreover, Figure 6.1 depicts a visual representation of the different components of GATH.

6.1.1 Overall formulation

Overall, the GATH layer can be expressed as follows. First, features are projected into a lower-dimensional space of size e_1 through a linear transformation:

$$h_u = \text{LeakyReLU}(Wx_u + c) \tag{6.1}$$

where $x_u \in \mathbb{R}^f$ are node u's features, $W \in \mathbb{R}^{e_1 \times f}$ is a learnable matrix and $c \in \mathbb{R}^{e_1}$ is a learnable bias term. This step is helpful because features are generally high-dimensional (in the order of thousands) and it would therefore be cumbersome to deal with all of them in the later stages.
Then, attention weights a are computed between a target node and each node up to its k-th neighborhood, excluding the node itself. Attention weights are computed by a 2-layer perceptron and are normalized with a softmax function. The information used as input for attention computation is constituted by the transformed features, the node degrees and the distance between the nodes. Overall, attention weight computation can be expressed as

$$a_{uv} = \text{softmax}_v(z_1^T \text{LeakyReLU}(Z_2[h_u||h_v||d_u||d_v||dist_{uv}])) \tag{6.2}$$

where $z_1 \in \mathbb{R}^{e_2}$ and $Z_2 \in \mathbb{R}^{e_2 \times (2e_1+3)}$ are learnable parameters, d_u is the node degree of u and $dist_{uv}$ is the distance between nodes u and v in terms of number of hops (i.e. length of shortest path between the nodes).
Subsequently, node embeddings are computed by aggregating information from the neighbors up to the k-hop using the attention weights a. Moreover, the contribution of the messages from the neighbors and the message from the node itself are balanced through a learnable coefficient. Overall, the final embeddings can be expressed as

$$h'_u = (1 - b) \cdot \sum_{v \in N_k(u)} a_{uv} h_v + b \cdot h_u \tag{6.3}$$

where b is a learnable scalar normalized between 0 and 1 with a sigmoid function, $N_k(u)$ is the set of nodes up to the k-hop neighborhood of node u and h'_u is the final embedding for node u. This formulation can be identified with the term *skip-connection*, since it introduces a direct path for the self-node embeddings to be

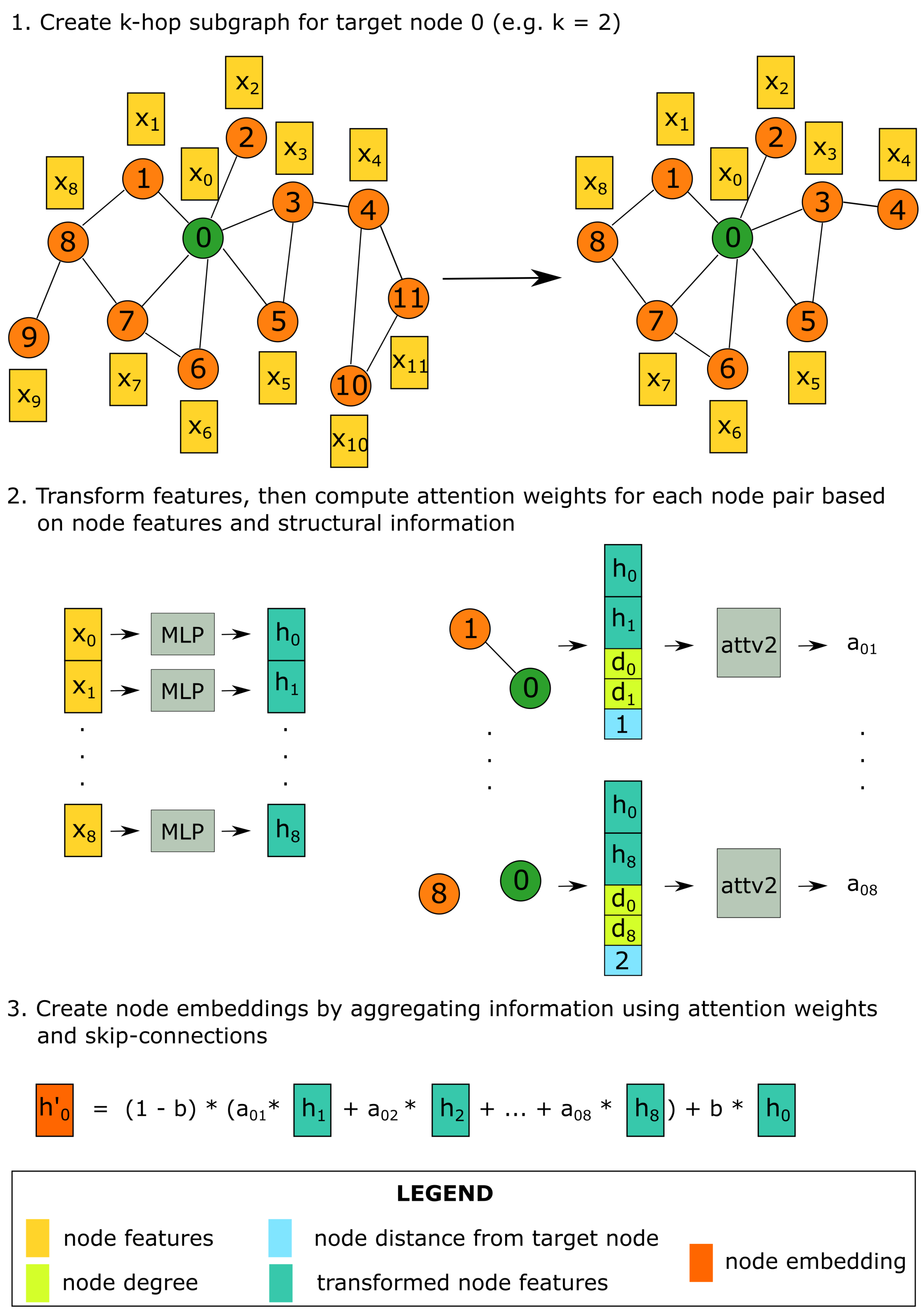

1. Create k-hop subgraph for target node 0 (e.g. k = 2)
2. Transform features, then compute attention weights for each node pair based on node features and structural information
MLP
attv2
a_{01}
a_{08}
3. Create node embeddings by aggregating information using attention weights and skip-connections
$h'_0 = (1 - b) * (a_{01} * h_1 + a_{02} * h_2 + \dots + a_{08} * h_8) + b * h_0$
LEGEND
node features
node degree
node distance from target node
transformed node features
node embedding

Figure 6.1: Proposed GATH architecture.

incorporated in the final embedding, skipping the aggregation. The final embedding h'_u is then forwarded through a linear layer to perform classification:

$$\tilde{y}_u = \text{LogSoftmax}(W_{cl} h'_u + c_{cl}) \tag{6.4}$$

where $W_{cl} \in \mathbb{R}^{|C| \times e_1}$ is a learnable matrix, $c_{cl} \in \mathbb{R}^{|C|}$ is a learnable bias term and $\text{LogSoftmax}(\cdot) = \log(\text{softmax}(\cdot))$ is chosen over standard $\text{softmax}(\cdot)$ to improve numerical performance. Note that $\tilde{y}_u \in \mathbb{R}^{|C|}$ is the probability distribution over classes for node u. During inference, GATH assigns to the node the class with maximum likelihood, i.e.

$$\hat{y}_u = \text{argmax}(\tilde{y}_u) \tag{6.5}$$

The model is trained to minimize the negative log-likelihood on the training set, i.e. the normalized sum of the negative log of the probabilities assigned by the model to the true class. In formulas:

$$\mathcal{L}(\tilde{y}, y) = -\frac{1}{|V_{train}|} \sum_{u \in V_{train}} \log(\tilde{y}_{u y_u}) \tag{6.6}$$

where V_{train} is the set of nodes used for training.

6.1.2 Attention formulation

As observed in [6] and discussed in Section 2.2.4, the standard formulation of attention adopted in GAT (Equation (2.5)) is not expressive enough to handle some specific types of problems. An improved version, attv2 (Equation (2.8)), solves this limitation by introducing a transformation after the non-linearity. To better capture the relations among nodes, the attention formulation of GATH follows this approach. Indeed, as shown in 6.2, the vector z_1 multiplies the embeddings after the non-linearity, providing increased expressiveness.

6.1.3 Neighborhood extension

It has been discussed that, in heterophilous graphs, important information for node classification might not depend only on the graph structure, that is, some neighboring nodes might have irrelevant or misleading information whereas non-connected nodes might carry useful messages. To allow for a more flexible interaction among nodes in the graph, GATH exploits a variation of the attention mechanism of GAT [5], generating a weight between two nodes based on how relevant they are to each other. However, the formulation of GAT has some limitations that hamper its performance on heterophilous settings.

As a matter of fact, GAT only aggregates information from connected nodes[1]. To incorporate also potentially useful information from further nodes, GATH performs message aggregation considering nodes up to the k-th neighborhood, where k is a hyperparameter. However, just aggregating messages from nodes at different distances might cause the loss of relevant information conveyed by the graph structure. To address this problem, we take into consideration relevant structural properties of the nodes when computing the attention weights, namely the node degrees and the hop-distance between a node and the current target node. We select node degrees as they are a relevant indicator of the importance of a node in the graph [17, 47], whereas node distance is meaningful to understand how strongly two nodes are related.

6.1.4 Skip-connections

By comparing GAT and MLP performance on heterophilous benchmarks (e.g. Figure 3.1 and Table 3.1), it is possible to observe that in some scenarios GAT can be largely outperformed by MLP. Since MLP can be viewed as a GAT where the attention weight matrix corresponds to the identity matrix, it can be hypothesized that GAT struggles to learn high self-attention weights (i.e. assign high attention values to the node itself). To facilitate this, skip-connections can be used. Skip-connections were introduced in the field of Computer Vision [48, 49] and they achieved great success thanks to their ability to facilitate the learning of the identity mapping, which consists in keeping a layer equal to the previous one, thus avoiding the potentially negative impact of more complex transformation. In GATH, skip-connections play the same role. In particular, through a learnable gate b, the network can balance the contributions of the messages aggregated from the neighbors and the message of the target node itself, as expressed in Equation (6.3). Moreover, this can be interpreted as an implementation of the ego-neighbor separation principle presented in 3.1.2, since representations for the neighborhood and for the self-node are merged only at the end of the layer.

6.1.5 Time complexity

We can divide the computation of the time complexity of GATH into the different steps that characterize the architecture.
First of all, features are projected into lower-dimensional embeddings of size e_1 through a matrix multiplication and a non-linear transformation, as expressed

[1]The original paper ([5]) defines a general formulation of attention that considers also non-connected nodes, but the provided implementation aggregates information only from 1-hop neighbors.

in Equation (6.1). The time required by this step is dominated by the matrix multiplication, which takes $\mathcal{O}(ne_1 f)$, where n is the number of nodes and f the feature size.
Then, attention weights are computed according to the formula described in Equation (6.2). For a node pair (u, v), the computation of the corresponding attention weight takes $\mathcal{O}(e_2 e_1)$ time, since the dominant operation is the multiplication with the matrix Z_2. This computation is performed once for every node pair, i.e. $|E_k|$ times, where E_k is the set of edges of the graph created by connecting each node to every other node located up to its k-hop neighborhood. Therefore, the overall time complexity of the second step is $\mathcal{O}(|E_k| e_2 e_1)$.
In conclusion, messages are aggregated from the neighbors using the learnable parameter b as weight between neighborhood embeddings and self-node embedding, as shown in Equation (6.3). In this step, the aggregation over neighbors takes $\mathcal{O}(|E_k| e_1)$ time, whereas the sum of the contribution of the self-node embedding takes $\mathcal{O}(ne_1)$ time. It can be noticed that both these terms are dominated by the previous ones. Indeed, the aggregation complexity is dominated by the time complexity derived at the second step, i.e. $\mathcal{O}(|E_k| e_2 e_1 + |E_k| e_1) = \mathcal{O}(|E_k| e_2 e_1)$, and the sum of self-node embeddings is dominated by the feature transformation, i.e. $\mathcal{O}(ne_1 f + ne_1) = \mathcal{O}(ne_1 f)$.
Summing everything up, we can write the overall time complexity of GATH as $\mathcal{O}(ne_1 f + |E_k| e_2 e_1)$. We can also compare the time complexity of GATH with that of GAT [5] and GATv2 [6]: $\mathcal{O}(ne_1 f + |E| e_1)$. It can be noticed that there are two differences between the complexities of the two models. First of all, GATH employes a 2-layer perceptron to compute attention weights, which leads to a complexity of $\mathcal{O}(e_2 e_1)$ for each node pair, whereas GAT and GATv2 have only one layer, lowering the complexity to $\mathcal{O}(e_1)$. However, the most significant difference is the number of weights to compute: $|E|$ for GAT, $|E_k|$ for GATH. Indeed, $|E_k|$ can be much larger than $|E|$, since the number of edges grows, in general, exponentially every time the neighborhood is enlarged by one hop.

6.2 GCNH

The second model proposed in the scope of this work is GCNH (GCN for Heterophily). Despite the flexibility introduced by GATH in learning weights between different node pairs in the graph, without being limited by the graph connectivity, in Section 5 we observe that even a very simple model like GCN can achieve good performance on some heterophilous graphs, namely `Chameleon` and `Squirrel`. Although GATH can in principle learn attention weights that resemble the adjacency weights used by GCN, this does not easily occur in practice, given the small size of the datasets used and the limited dimensionality of the network layers to avoid

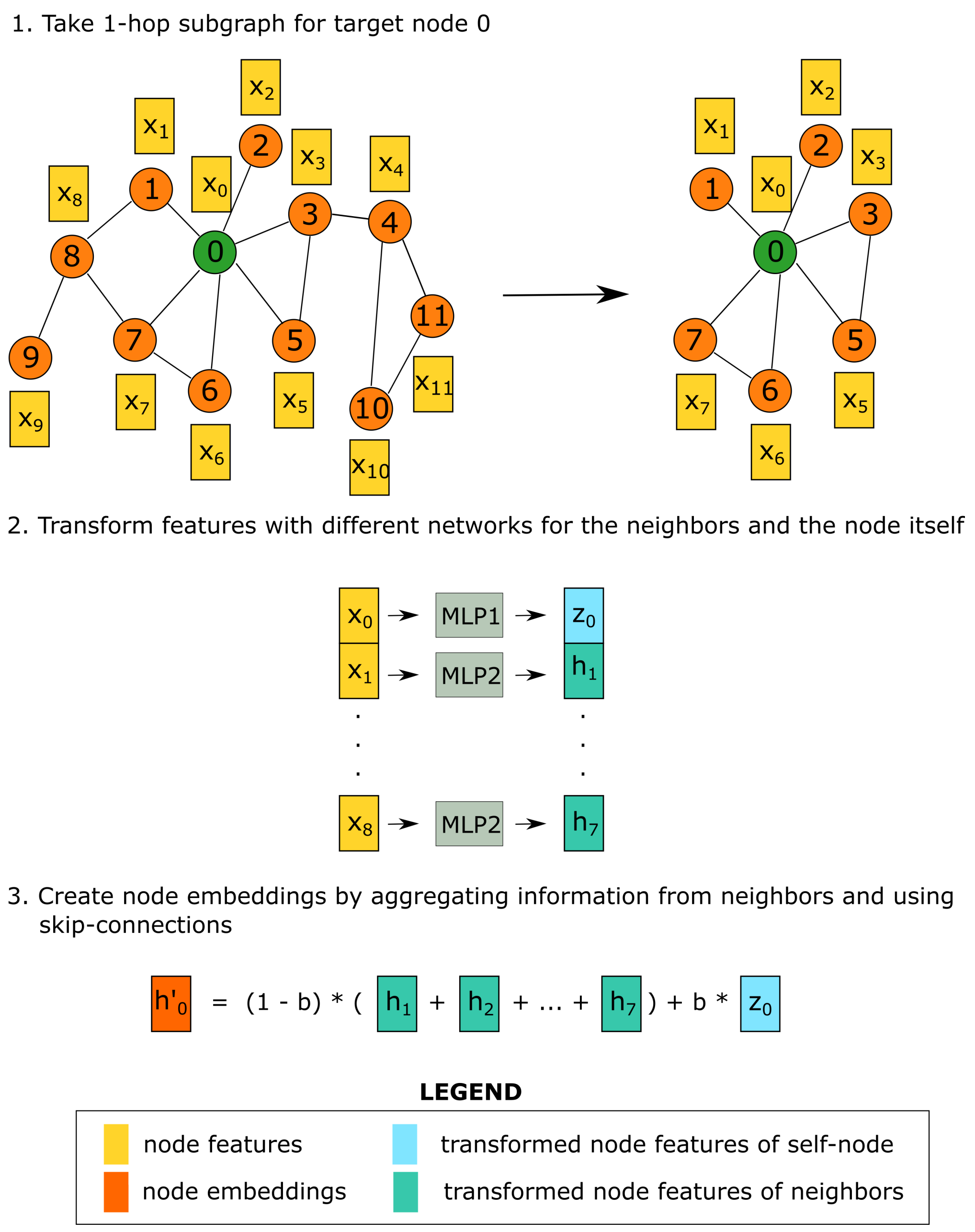

1. Take 1-hop subgraph for target node 0
2. Transform features with different networks for the neighbors and the node itself
MLP1
MLP2
MLP2
3. Create node embeddings by aggregating information from neighbors and using skip-connections
h'0 = (1 - b) * (h1 + h2 + ... + h7) + b * z0
LEGEND
node features
node embeddings
transformed node features of self-node
transformed node features of neighbors

Figure 6.2: Proposed GCNH architecture.

overfitting. To exploit the capability of GCN to capture specific graph properties and extend its performance to heterophilous graphs, we introduce GCNH. GCNH is a simpler model compared to GATH, as it does not involve learnable weights between nodes. However, it allows for an effective exploitation of the information provided by the adjacency matrix, which may be lost in the more complex formulation of GATH. Figure 6.2 provides a visual representation of the behavior of GCNH.

6.2.1 Overall formulation

As first step, GCNH performs feature transformation through two separate 1-layer MLPs in order to reduce the feature dimensionality. The two sets of feature embeddings obtained in this way are used for different purposes: one is used when nodes act as neighbors, the other when the node is the target. For a node u, this step can be expressed as follows:

$$h_u = \text{LeakyReLU}(W_1 x_u + c_1) \tag{6.7}$$

$$z_u = \text{LeakyReLU}(W_2 x_u + c_2) \tag{6.8}$$

where $W_1 \in \mathbb{R}^{e \times f}$ and $W_2 \in \mathbb{R}^{e \times f}$ are learnable matrixes and $c_1 \in \mathbb{R}^e$ and $c_2 \in \mathbb{R}^e$ are learnable bias terms. Then, message aggregation is performed using a learnable scalar b, normalized between 0 and 1, to balance the contributions of neighborhood embeddings and self-features. This mechanism can be viewed as a skip-connection and is analogous to the one employed by GATH. In formulas, the computation of the node embedding h'_u for a node u can be expressed as

$$h'_u = (1 - b) \cdot \sum_{v \in N(u)} h_v + b \cdot z_u \tag{6.9}$$

A linear transformation is then applied to the embedding h'_u to get per-class probabilities:

$$\tilde{y}_u = \text{LogSoftmax}(W_{cl} h'_u + c_{cl}) \tag{6.10}$$

where the symbols are the same as Equation (6.4). The label assigned to the nodes and the loss used for training are the same described in Equations (6.5) and (6.6).

6.2.2 Time complexity

We can compute the time complexity of GCNH by analyzing separately the different steps involved in the model.
To begin with, features are transformed into two different sets of embeddings of lower dimensionality e, as described in Equations (6.7) and (6.8). This step takes $\mathcal{O}(nef)$ time.
Then, messages are aggregated from 1-hop neighbors and merged with the self-node embedding according to Equation (6.9). The aggregation from neighbors takes

$\mathcal{O}\left(|E|e\right)$ time, whereas the weighted sum with the self-node embedding takes $\mathcal{O}\left(ne\right)$ time. This last term is dominated by the feature transformation complexity. Overall, the time complexity of GCNH can be written as $\mathcal{O}\left(nef + |E|e\right)$. With respect to GATH, the complexity is significantly lower since there is no attention weight computation and aggregation is performed only from 1-hop neighbors. This leads to a relevant increase in efficiency for GCNH compared to GATH.

Chapter 7

Experiments

This chapter provides experimental evidence of the performance of the two models introduced in this thesis, GATH and GCNH. First of all, we report results for node classification on real and synthetic graphs and we compare them with other state-of-the-art models. Then, we perform ablation studies to evaluate the impact of the different design choices on the final architecture and the influence of the different hyperparameters tested. In conclusion, we analyze the parameters learned by the model for different datasets to understand how different graph properties affect the model behavior.

7.1 Node classification performance

We evaluate the representation capabilities of GATH and GCNH on the task of supervised node classification, which is a common choice for models dealing with heterophilous graphs. We perform the evaluation both on synthetic datasets with different levels of homophily ratio and on the real-world graphs commonly used in works dealing with assortative networks. Further information about the datasets is reported in Section 4.

7.1.1 Baselines

The baselines used for comparison can be divided into two main categories. The first category includes basic methods, with respect to which we expect large improvements especially on heterophilous benchmarks given their limited flexibility. In the following, we introduce the methods belonging to this class.

- **MLP**: 2-layer perceptron. It is a graph-agnostic model and it is reported to understand how much information is conveyed by the plain node features.

- **GCN** [4]: Graph Convolutional Network
- **GAT** [5]: Graph Attention Network

The second category of reported baselines is composed of methods that are specifically designed for heterophilous graphs. It is fundamental to compare GATH and GCNH to these approaches since they are implemented for the same task. In particular, we present in the following the approaches used for comparison.

- **Geom-GCN** [18]: method for heterophily that maps nodes to a latent space and defines a new graph based on embedding similarity
- **H2GCN** [9]: method that introduces three specific designs to boost performance on heterophilous graphs: ego and neighbor-embedding separation, higher-order neighborhoods and combination of intermediate representations
- **GPRGNN** [39]: method that uses PageRank to determine relations between nodes
- **GGCN** [10]: method that applies two designs to extend GNNs: degree correction and signed messages
- **O(d)-SD** [35]: method based on sheaf diffusion

7.1.2 Experimental setting

We evaluate model performance in terms of classification accuracy, i.e. the percentage of nodes in the test set that are assigned to the correct class. Accuracy is a standard metric for classification performance and it is commonly used in related works. Moreover, the classes in the considered datasets are quite balanced, thus accuracy is a reliable metric for this setting.
We test different values for hyperparameters and we select the best ones. We report further information about the grids for the hyperparameters in Appendix A.4.
We run experiments on a NVIDIA Tesla V100 PCIE with 32 GB. The code is implemented in Python and deep learning models are defined and trained using PyTorch.
We compute classification accuracies on 10 different train/validation/test splits for each dataset, provided by [18], and we report average values and standard deviation for each dataset. The sizes of the splits are 48%/32%/20%. We train the models on the training set and the best performing model on the validation set is used on the test set, on which we compute the accuracy.

	Cornell	Texas	Wisconsin	Film	Chameleon	Squirrel	Cora	Citeseer
h	0.30	0.11	0.21	0.22	0.23	0.22	0.81	0.74
MLP	81.89 ± 6.40	80.81 ± 4.75	85.29 ± 3.31	**36.53±0.70**	46.21 ± 2.99	28.77 ± 1.56	75.69 ± 2.00	74.02 ± 1.90
GCN	50.27 ± 7.57	57.03 ± 5.05	50.98 ± 4.88	23.27 ± 0.94	67.43 ± 1.95	50.49 ± 1.54	84.29 ± 0.98	73.25 ± 1.42
GAT	61.89 ± 5.05	52.16 ± 6.63	49.41 ± 4.09	27.44 ± 0.89	60.26 ± 2.50	40.72 ± 1.55	87.30 ± 1.10	76.55 ± 1.23
Geom-GCN	60.54 ± 3.67	66.76 ± 2.72	64.51 ± 3.66	31.59 ± 1.15	60.00 ± 2.81	43.80 ± 1.48	85.35 ± 1.57	**78.02±1.15**
H2GCN	82.70 ± 5.28	84.86 ± 7.23	**87.65±4.98**	35.70 ± 1.00	60.11 ± 2.15	36.48 ± 1.86	**87.87±1.20**	77.11 ± 1.57
GPRGNN	80.27 ± 8.11	78.38 ± 4.36	82.94 ± 4.21	34.63 ± 1.22	46.58 ± 1.71	31.61 ± 1.24	**87.95±1.18**	**77.13±1.67**
GGCN	**85.68±6.63**	**84.86±4.55**	**86.86±3.29**	**37.54±1.56**	**71.14±1.84**	**55.17±1.58**	**87.95±1.05**	**77.14±1.45**
O(d)-SD	**84.86±4.71**	**85.95±5.51**	**89.41±4.74**	**37.81±1.15**	**68.04±1.58**	**56.34±1.32**	86.90 ± 1.13	76.70 ± 1.57
GATH	**83.78±6.28**	84.33 ± 5.24	85.88 ± 4.54	35.75 ± 1.43	49.06 ± 1.55	35.22 ± 1.46	85.98 ± 1.51	76.18 ± 1.59
GCNH	83.51 ± 7.50	**86.76±3.71**	86.27 ± 2.48	35.93 ± 1.09	**71.29±1.51**	**58.44±2.27**	85.05 ± 0.83	75.63 ± 1.10

Table 7.1: Mean classification accuracy and standard deviation for GATH and GCNH on real-world datasets. Best results are reported in **red**, second best results in **blue** and third best in **violet**. The results for the baselines are taken from [35], with the exception of GCN which was implemented and tested as described in Appendix A.2. Experiments for each dataset are run on the 10 different splits taken from [18].

7.1.3 Results on real-world datasets

To assess the effectiveness of the proposed models, Table 7.1 reports classification accuracies for GATH, GCNH and baselines on the real-world datasets.
We can observe that, on `Cornell`, `Texas` and `Wisconsin`, GATH outperforms the simple graph-unaware MLP and especially improves the simple GNN models (GCN and GAT). This behavior can be explained by observing that GATH extends GAT and GCN allowing for a more flexible importance of the features of the target node, which, on these datasets, turn out to be very relevant. However, the performance improvement with respect to MLP shows that GATH is also able to extract useful information from the graph structure. Moreover, GATH also outperforms Geom-GCN and GPRGNN, specifically designed for heterophily, and achieves competitive performance with H2GCN, improving the accuracy on one of the three graphs. GGCN and O(d)-SD outperform GATH on these three graphs. These observations confirm the effectiveness of the design choices implemented in GATH, even though they appear not to be sufficient to outperform state-of-the-art models.
On `Film`, GATH performs similarly to MLP. Since most models perform worse than MLP on this dataset and the improvement brought by O(d)-SD is quite small, it can be assumed that, as observed also in other works [20], the graph structure is not particularly relevant in this case and it mostly introduces misleading information.
On `Chameleon` and `Squirrel`, GATH does not provide satisfactory performance. This might be explained by observing that features in these two datasets are not particularly relevant, as proved by the low performance of MLP, whereas the graph structure by itself is very informative, since GCN provides very good results. GATH

struggles to learn attention weights that reflect the connections in the graph, and therefore is not able to perform satisfactory classification.
On the two homophilous graphs `Cora` and `Citeseer`, GATH performs quite closely to GCN and GAT, which are models that thrive in homophilous settings. Therefore, it can be stated that the increased model complexity of GATH does not hinder the performance on homophilous graphs.

We can also make some considerations based on the results for GCNH. Quite surprisingly given the simplicity of the model, GCNH outperforms GATH on 5 out of the 8 datasets, achieving state-of-the-art performance on three of them. The very good performance on `Chameleon` and `Squirrel` can be justified by observing that GCNH performs an adjacency matrix-based message aggregation, similarly to GCN, and this approach is shown to be very successful on these two graphs. The skip-connection is also proved to be useful given the improved performance compared to the simple GCN.
On the other heterophilous graphs, GCNH achieves very satisfactory performance, outperforming all the other models on `Texas`. This behavior can be motivated by noticing that node features are informative for these graphs, as proved by the performance of MLP compared to basic GNNs. In GCN and GAT, indeed, the information of the features is mixed with information coming from the neighborhood, which may be misleading. The skip-connection of GCNH solves this problem, since node features can contribute more to the final embeddings. Moreover, the performance improvements of GCNH with respect to MLP show that also aggregating information from the neighborhood is useful, as long as its contribution does not overwhelm the self-features.
On the homophilous graphs `Cora` and `Citeseer`, GCNH performs worse than GATH and the simple GNNs, but the gap is not large. This proves that, also in this case, the added complexity does not worsen performance significantly on homophilous graphs.

7.1.4 Results on synthetic datasets

Beyond real-world graphs, it is also interesting to analyze GATH and GCNH performance on synthetic graphs. Indeed, real-world graphs may incorporate more complex patterns that do not allow for immediate comparison between different graphs with similar homophily ratio. On the other hand, the synthetic graphs with variable homophily ratio described in Section 4 can be used to observe the change in performance related to different homophily levels in graphs with similar characteristics. To this end, Figure 7.1 shows the classification results of GATH and GCNH on these datasets.

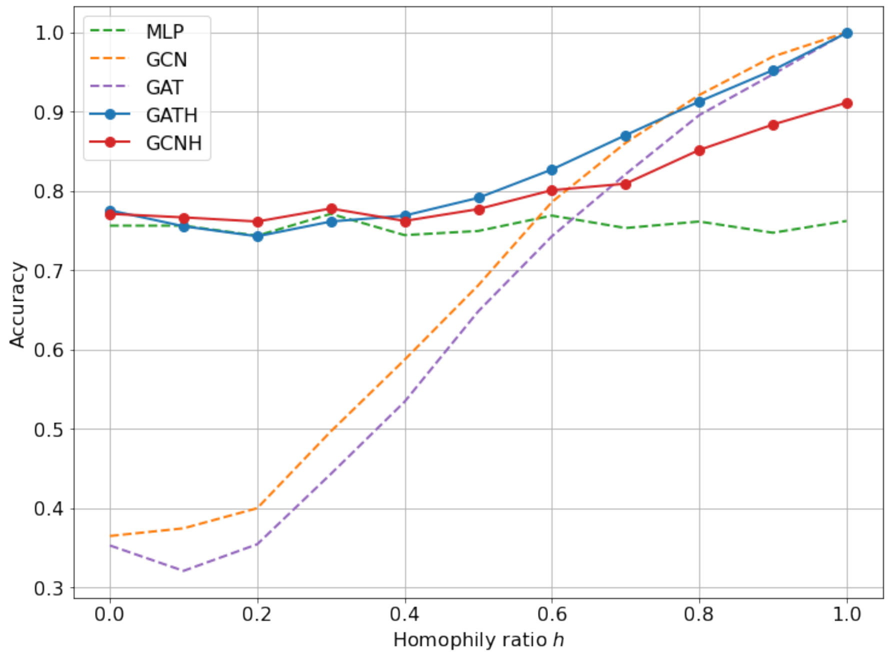


Figure 7.1: GATH and GCNH classification accuracies on synthetic datasets. Results show the average accuracy on 3 datasets for the different values of homophily ratio.

We can notice that GATH performance is still slightly dependent on the homophily level of the graph, but the accuracies also on heterophilous graphs are largely improved with respect to GCN and GAT. Moreover, on the graphs with high homophily, where the graph structure provides useful information, GATH is able to positively exploit it, achieving a performance that is very close to GCN and GAT and perfect accuracy on perfectly homophilous graphs ($h = 1$).

Similarly, GCNH performance is also less dependent on the homophily level of the graph with respect to GAT and GCN. In general, GCNH performs similarly to GATH on heterophilous graphs, which confirms that the skip-connection mechanism is effective in these settings. On homophilous graphs, GCNH performs slightly worse than GATH, which can be due to the lower flexibility of GCNH which does not involve the attention mechanism.

7.2 Ablation studies

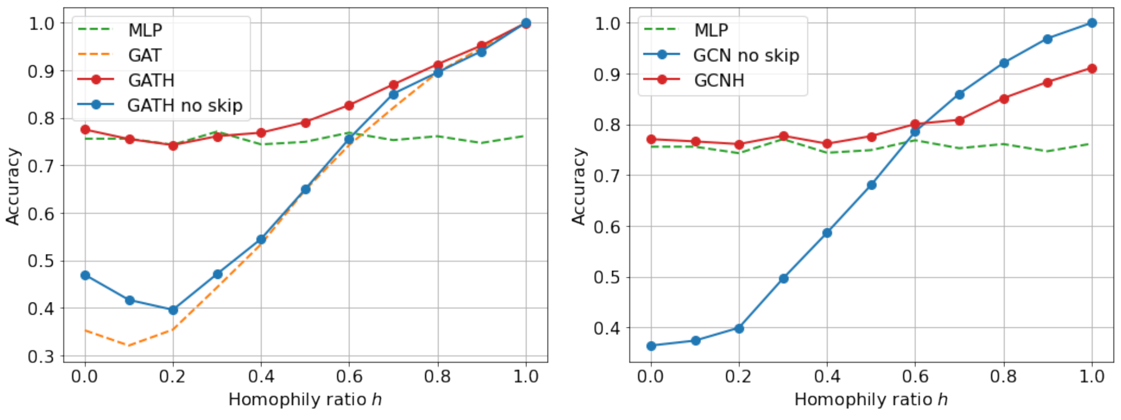
To thoroughly understand the behavior of GATH and GCNH it is necessary to delve into the performance and characterize the impact of each component of the models. To this end, this section analyzes the influence of the different design choices of GATH and GCNH by showing the variation in performance when different components are added or removed. Moreover, we also describe the influence of the different hyperparameters on the performance.

7.2.1 Impact of design choices

This section motivates the design choices of GATH and GCNH by analyzing how their presence affects the model performance.

Skip-connections

The first design choice to be considered are the skip-connections, described in Section 6.1.4. These allow for the separate computation of an embedding for the target node and for the neighborhood, which are then mixed as last stage using a learnable coefficient as weight. This design is present in both GATH and GCNH.



(a) GATH **(b)** GCNH

Figure 7.2: Comparison between performance of GATH and GCNH with and without skip-connections on synthetic graphs. For GATH, removing skip-connections means adding self-loops to the adjacency matrix and learning an attention weight also for the target node. For GCNH, instead, removing the skip-connections is equivalent to using a plain GCN model.

Figure 7.2a shows the difference in performance between GATH and a modified version of GATH where skip-connections are removed and self-loops are added to the adjacency matrix (i.e. the importance of self-features can only be learned through the attention weights). It is very clear that skip-connections widely affect the performance, especially on graphs with low homophily. As a matter of fact, without skip-connections GATH only provides a marginal improvement with respect to GAT, and performs largely worse than the MLP on disassortative graphs. This experiment confirms the hypothesis that it is difficult for the attention mechanism to learn to give high values to self-attention weights, and skip-connections help overcome this issue.

Figure 7.2b shows a similar plot for GCNH. In this case, the model without skip-connections is a simple GCN where message aggregation is performed from all neighbors and the target node itself. Also in this case, it can be noticed that the introduction of skip-connections is fundamental to boost performance on heterophilous graphs, resulting in a 30% improvement. However, in this case, skip-connections are slightly detrimental for homophilous graphs, leading to a decrease in accuracy for GCNH with respect to the standard GCN.

Attention of GATv2

Figure 7.3 shows the difference in performance when using the formulation of attention defined in GATv2 [6] instead of the standard definition of attention for GAT [5], as described in Section 6.1.2.

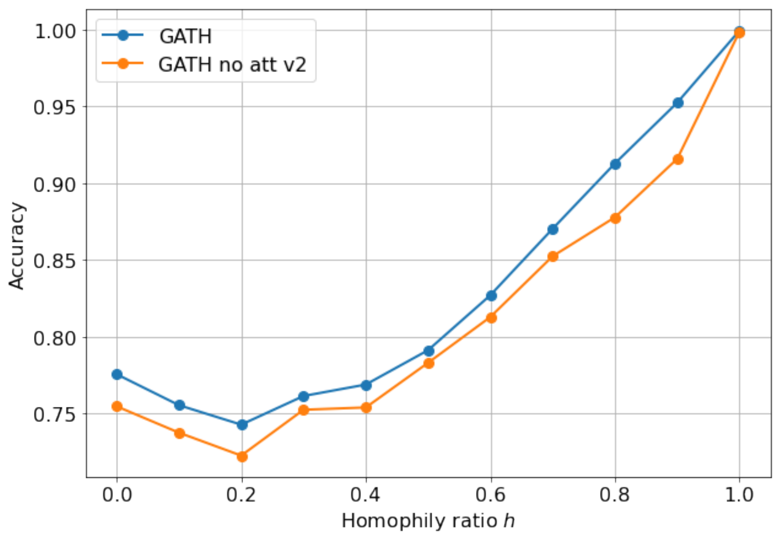


Figure 7.3: Comparison between performance of GATH with attv2 and with standard GAT attention on synthetic graphs.

We can observe that GATH with extended attention consistently outperforms the version with the standard attention formulation for all homophily levels. Despite the improvements being small, this confirms that the increased expressivity of attv2 compared to the standard attention translates into an improvement in performance.

Structural information

Figure 7.4 shows the impact of the addition of structural information to the feature embeddings when computing attention weights, as described in Section 6.1.3. This design choice is believed to be important because, when aggregating from nodes up to the k-hop neighborhood, the structural information of the graph might be lost if not properly encoded. In particular, the proposed model uses the node degrees of both nodes and the number of hops between them, i.e. the length of the shortest path.

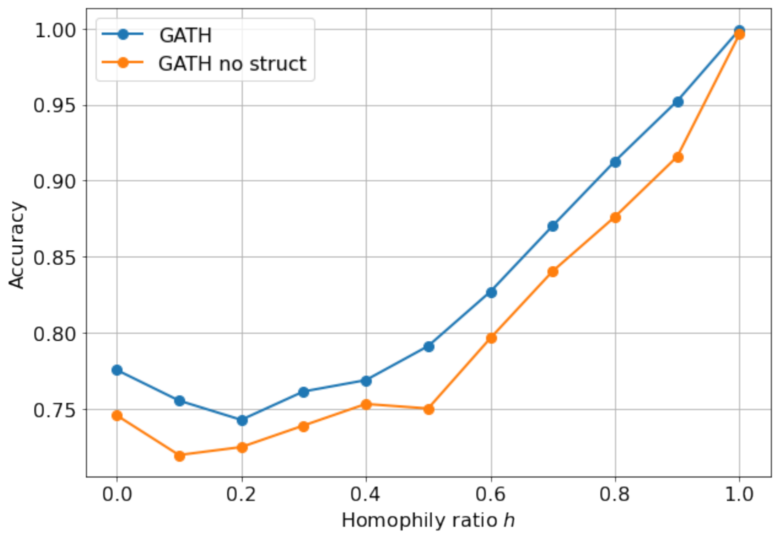


Figure 7.4: Comparison between performance of GATH with and without the use of structural information for attention computation on synthetic graphs.

Also in this case, we can observe that this addition allows for a more comprehensive attention weight computation that, eventually, improves the performance. This confirms the expectation that the structural information can be relevant for node classification and it is therefore important to retain it when aggregating from larger neighborhoods.

7.2.2 Impact of hyperparameters

In this section, we analyze the differences in performance related to variations of the relevant hyperparameters of the model.

Neighborhood size

The first hyperparameter considered is k, i.e. the number of hops used to define the extended neighborhood containing the nodes whose messages are aggregated to create the final embedding for a target node. Figure 7.5 shows how different values of this parameter affect the model performance.

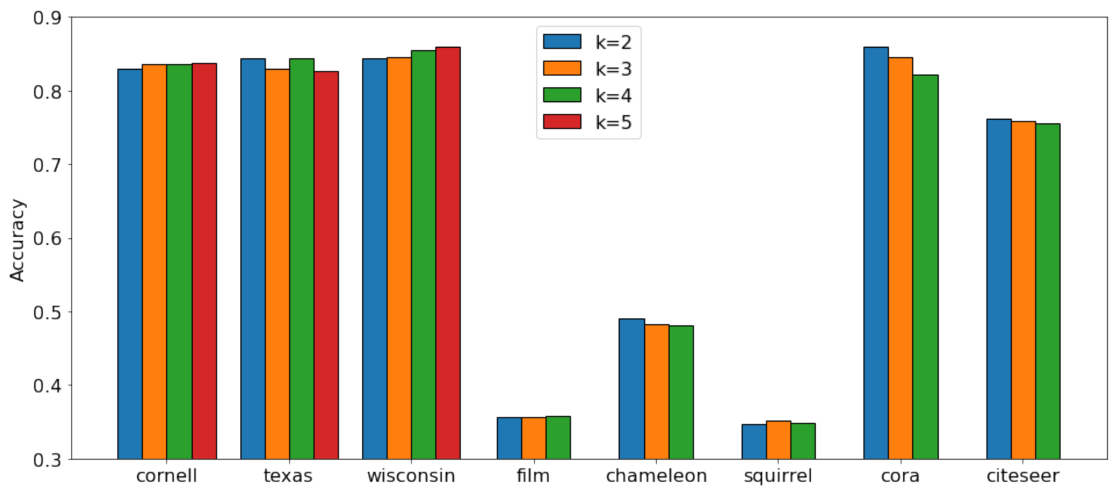
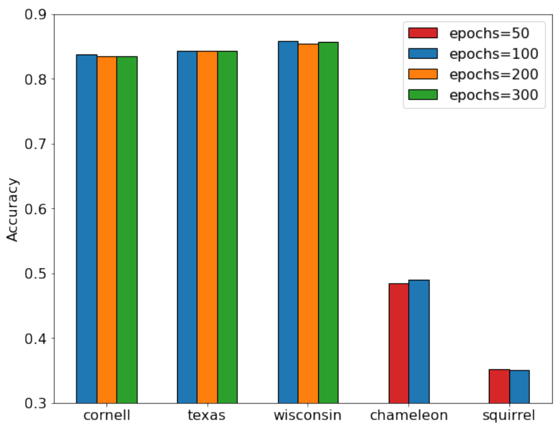


Figure 7.5: GATH performance for different neighborhood sizes k on real-world graphs.

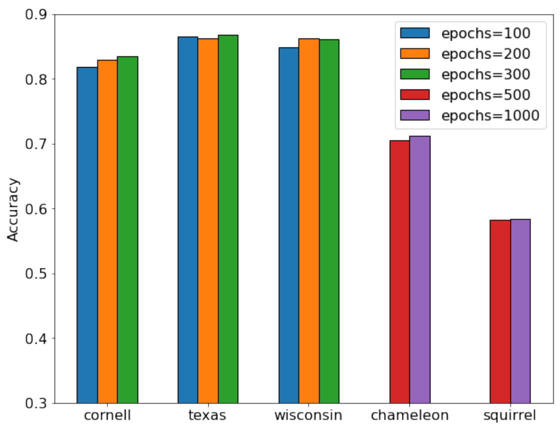
We can observe that for `Cornell` and `Wisconsin` enlarging the neighborhood size improves performance and the best results are obtained with the highest value tested, i.e. 5. For `Texas`, instead, similar results are obtained with k equal to 2 or 4, whereas enlarging it to 5 is detrimental.
On `Chameleon`, the best value for k is 2, which is likely due to the fact that enlarging the neighborhood further introduces many more nodes in the neighborhood since the graph is very dense and it becomes harder for the attention mechanism to distinguish which nodes are relevant and which are not. For `Squirrel` and `Film`, the differences in performance for different values of k are very small. In general, however, it can be stated that larger values of k bring small improvements.
For the homophilous graphs `Cora` and `Citeseer`, the best results are achieved with k equal to 2. This is intuitively justified by observing that enlarging the neighborhood means increasing the risk to include nodes with different labels in the message aggregation, since the 1-hop neighbors are likely to share the same label.

Number of epochs

Another relevant hyperparameter is the number of training epochs. Indeed, training for too many epochs may lead to overfitting, i.e. learning non-relevant patterns in the dataset that hinder generalization capabilities. On the other hand, the number of epochs should be large enough for the model to get as close as possible to a local optimum for loss minimization. An effective approach to tune the number of epochs consists in monitoring the variations of the training and validation losses and accuracies during training. When the training and validation accuracies are both increasing, the training process should continue. However, when the validation accuracy starts decreasing, the training should be stopped, since overfitting is occurring. Due to this, the model used for testing in the experiments of the work is the model that performs best on the validation set.



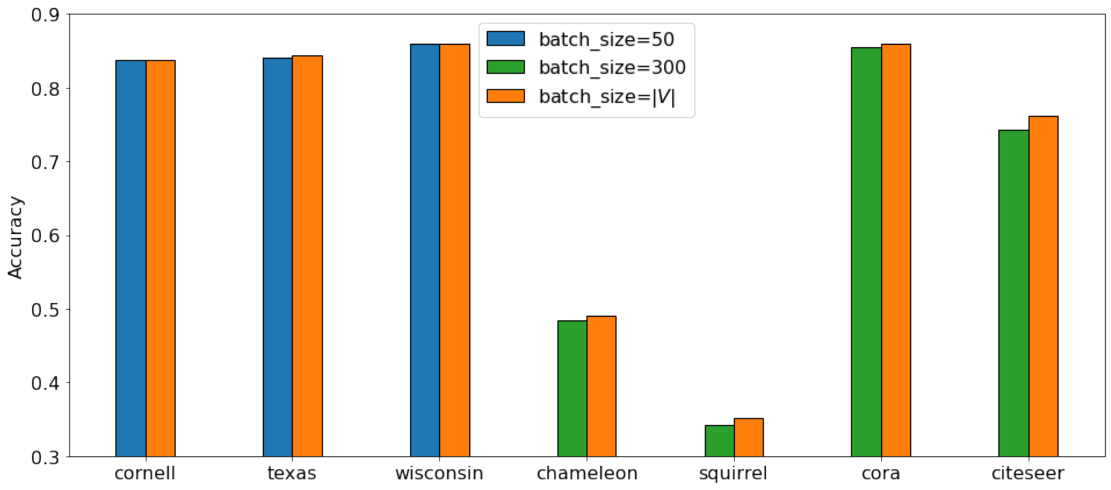
(a) GATH



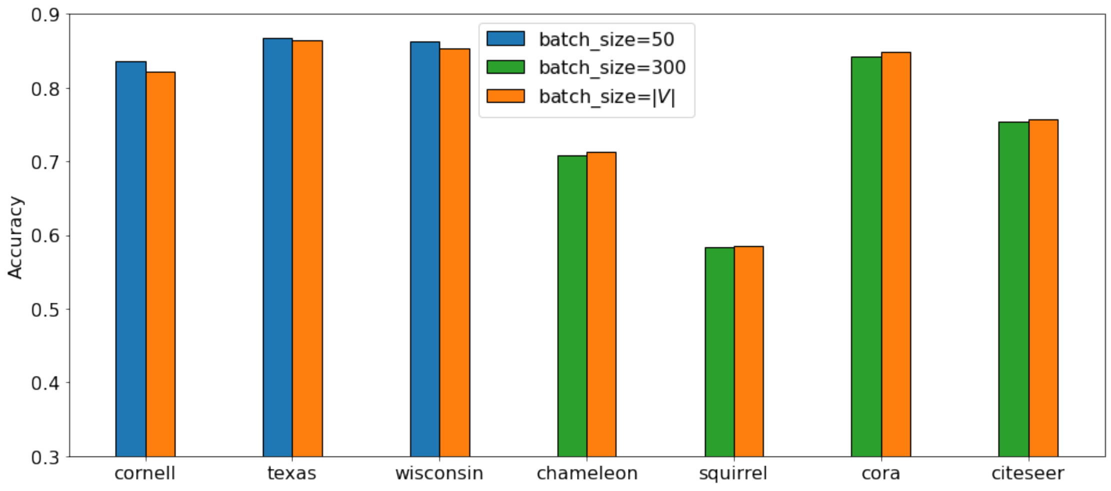
(b) GCNH

Figure 7.6: GATH and GCNH performance with different numbers of training epochs. Results are only reported for the graphs for which different values were tested.

Figure 7.6 shows how changing the number of training epochs affects the performance. Results are not reported for `Film`, `Cora` and `Citeseer`, as only one value for the number of epochs was tested for them. For GATH, the number of training epochs does not significantly affect the results. For GCNH, instead, increasing the number of epochs is generally helpful. Moreover, it is relevant to notice how the optimal number of epochs for `Chameleon` and `Squirrel` is much larger than the one used for the other graphs. The same behavior is observed with simple GCN, whose performance is optimal on these two graphs when training for about 1000 epochs, whereas on the other datasets the optimal number of epochs around 100.



(a) GATH

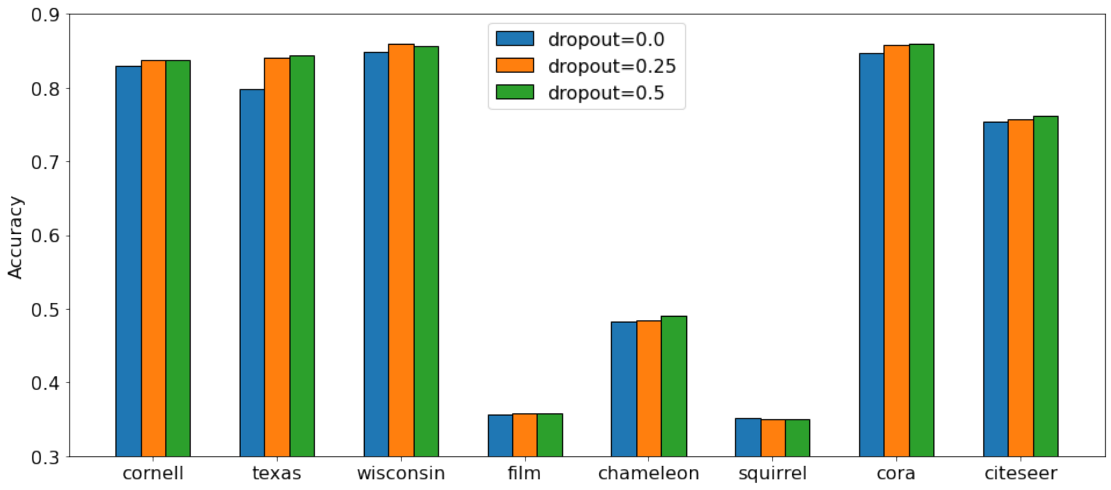


(b) GCNH

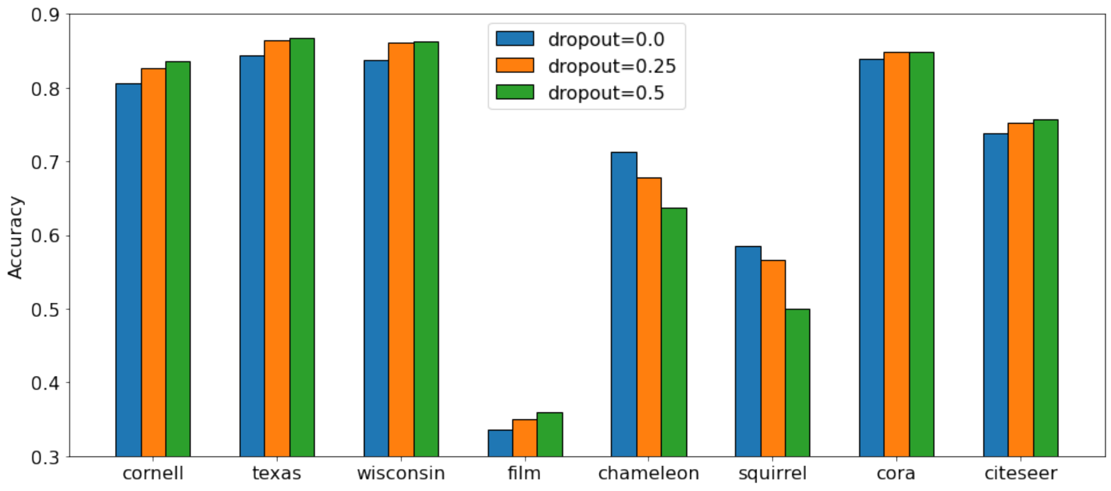
Figure 7.7: GATH and GCNH performance for different batch sizes. Setting the batch size equal to $|V|$ is equivalent to having just one batch, i.e. forwarding the complete training dataset through the model at once. Results are not reported for `Film` since only one value was tested for it.

Batch size

Splitting the training dataset into different batches can be helpful for several reasons. First of all, it allows to handle larger datasets even with limited computing resources, since the training data do not need to be processed all at once. Moreover, it acts as a regularization technique, since each batch provides a different contribution to the gradient descent algorithm and batches are different at every epoch. Therefore,



(a) GATH



(b) GCNH

Figure 7.8: GATH and GCNH performance for different values of the dropout ratio, i.e. the percentage of attention weights that are randomly set to zero at each training epoch. Note that dropout = 0.0 is equivalent to not using dropout at all.

we test different batch sizes to improve model performance. Figure 7.7 shows the impact of different batch sizes on the model performance. Results are not reported for `Film` since the only tested value for the batch size is 300. Indeed, it was not possible to use $|V|$ as batch size due to memory issues.
Despite the expectations, it can be observed that having several batches does not improve performance on most graphs, with the only exception of `Cornell`, `Texas` and `Wisconsin` for GCNH. This can be explained by observing that the datasets

are generally quite small and therefore batches might not contain enough data to be representative for the complete graph and incorporate relevant patterns.

Dropout

Dropout is another regularization technique that aims at improving the model generalization capabilities. In particular, it consists of randomly setting some parameters to zero during one epoch, such that the model learns alternative representations to still classify the nodes correctly, without relying to much on a single pattern. In GATH, dropout is implemented by zeroing some attention weights, whereas in GCNH it is implemented by setting to zero some values of the embeddings used for classification.

Figure 7.8 shows how different values of the dropout ratio affect model performance. We can point out that dropout is generally helpful for both GATH and GCNH, and usually a higher value of dropout ratio corresponds to higher performance. The only exception is represented by GCNH on `Chameleon` and `Squirrel`, since in this case the best results are achieved without dropout.

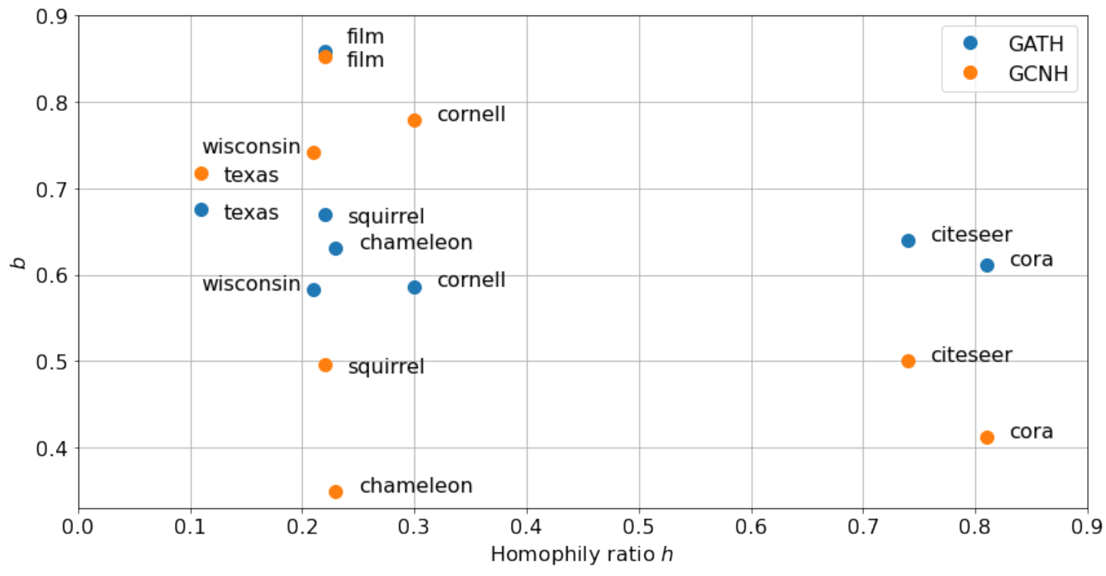
7.3 Performance analysis

This section provides deeper insights into model performance by analyzing the values of the different parameters learned by GATH and GCNH on different graphs.

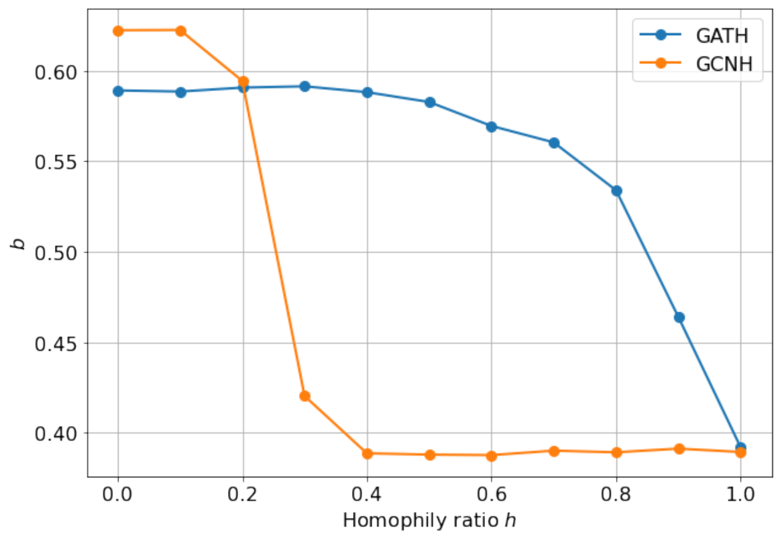
Coefficient b

The first term to be analyzed is the coefficient b that balances the contribution of the messages of the self-node and the neighbors in the aggregation phase, as described in Equations (6.3) and (6.9). This parameter is present in both GATH and GCNH.

Figure 7.9 shows the values of the parameter b on different graphs. On synthetic graphs (Figure 7.9b), it can be observed that the value for both GATH and GCNH significantly decreases for graphs with high homophily, in which the neighbors are likely similar to the target node and therefore are useful to improve classification. For heterophilous benchmarks, instead, b is higher, as better results are achieved if the main contribution to the node embeddings is given by the features of the node itself. It is also worth noticing that the value of b starts decreasing earlier for GCNH than for GATH as homophily increases.
On real graphs (Figure 7.9a), however, the values of b are less related to the homophily ratio. For GATH on `Film`, b has a large value, which is consistent with the observation that the graph structure of the dataset is not particularly relevant, as confirmed also by the largely better performance of MLP on the graph with



(a) Real graphs



(b) Synthetic graphs

Figure 7.9: Values of parameter b for GATH and GCNH on real and synthetic datasets.

respect to GCN and GAT. For the other datasets, however, the value of b does not appear to be related to the graph characteristics and is always between 0.6 and 0.7. For GCNH, it is relevant to notice that the value of b is low for `Chameleon`, where it has already been observed that the structural information derived from the adjacency matrix is particularly relevant. However, b assumes a higher value on

`Squirrel`, where the same observation holds, and node features are even less informative. Nevertheless, the value of b on `Squirrel` is lower than on the graphs with similar homophily ratio, which may indicate that the graph structure is more relevant in this case. On `Film`, similarly to GATH, the value of b is high since the graph structure is not informative for the dataset. Moreover, it can be noticed that the values of b for `Cora` and `Citeseer`, homophilous graphs, are lower than the values for `Cornell`, `Texas` and `Wisconsin`, heterophilous graphs, which is consistent with the observations made on the synthetic graphs and with the intuition that higher homophily corresponds to more relevant graph structure that plays a more important role in the aggregation for the final embeddings.

Weight distribution

We can make further considerations by observing the attention weights learned by GATH. This analysis is not applicable to GCNH.

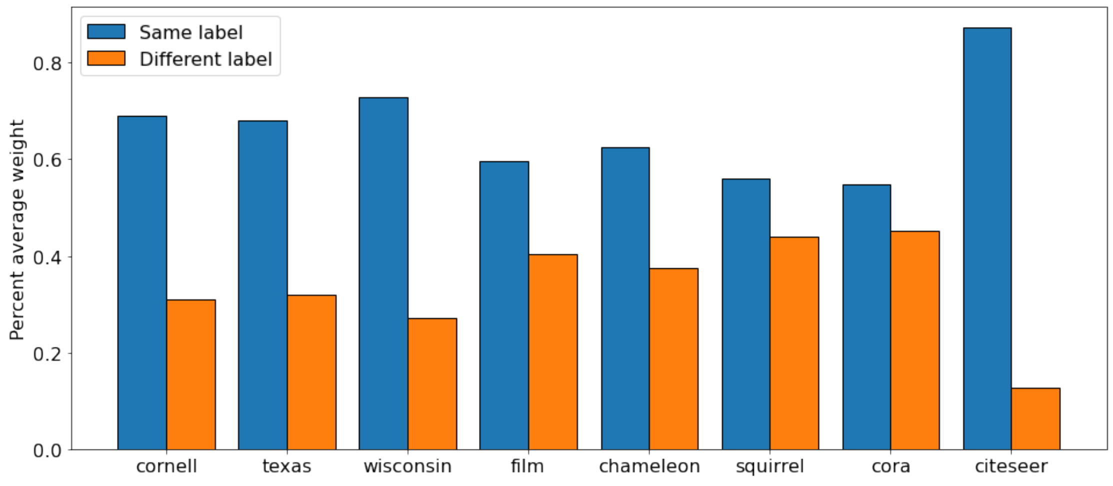


Figure 7.10: Normalized average weight for nodes with same or different labels with respect to the target node on real graphs. We compute the average weight for nodes with same and different label and we normalize columns such that, for each dataset, they sum to one. This is done for visualization purposes, since denser dataset have significantly lower average weight.

In particular, Figure 7.10 shows the normalized average weight for nodes with same or different label with respect to the target node on the real graphs. It can be noticed that nodes with the same label receive, on average, higher weights than nodes with a different label, consistently across all graphs. This behavior is coherent with the expectation that nodes with the same label carry information that is relevant for the correct classification of the target node, and therefore should play an important role during message aggregation.

Furthermore, we can observe the normalized average weight with respect to the distance (number of hops) of a node from the target. Figure 7.11 depicts

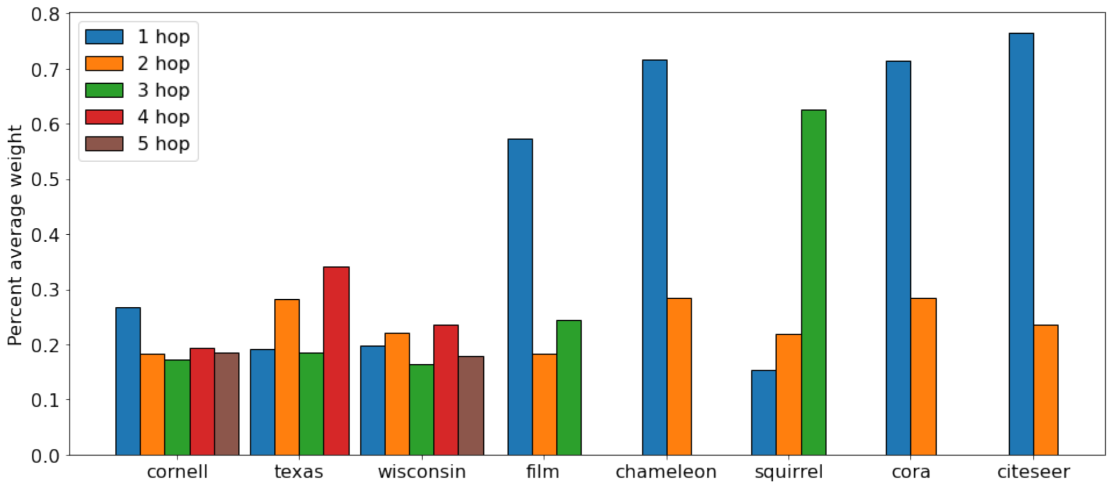


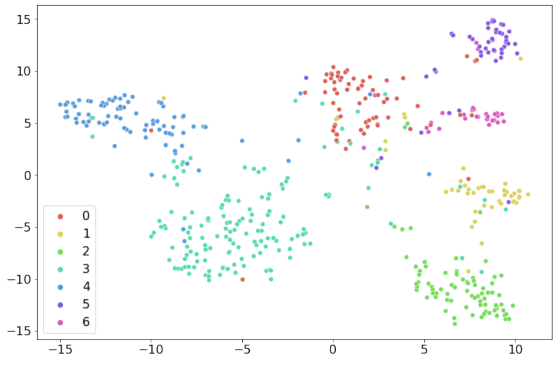
Figure 7.11: Average weight for nodes at different hop-neighborhoods with respect to the target node. We compute the average weight for nodes at each hop and we normalize columns such that, for each dataset, they sum to one. This is done for visualization purposes, since denser dataset have significantly lower average weight.

how the weights are distributed, on average, across different hops. Interestingly, in the homophilous graphs `Cora` and `Citeseer`, the weight distribution is largely dominated by the 1-hop neighbors, which is consistent with the fact that in homophilous graphs 1-hop neighbors tend to be similar to the target nodes and, therefore, it is useful to aggregate information from them. On the other graphs, the behavior is more diverse. In `Cornell`, `Texas` and `Wisconsin`, the distribution is almost uniform across different hops, meaning that relevant nodes are located in various positions in the graph independently of the distance from the target node. In `Film` and `Chameleon`, weights are mostly located in the first hop, which is unexpected given the low homophily ratio of these graphs. On `Squirrel`, in conclusion, the weight distribution is dominated by the third hop.

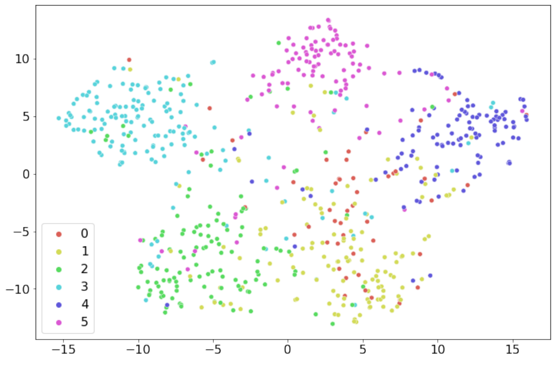
Node embeddings visualization

Another useful analysis to characterize the model behavior is the observation of the distribution of the node embeddings in their latent space. Indeed, this distribution is informative about how well the backbones of GATH and GCNH map the nodes to embeddings located in different portions of the latent space based on the observed

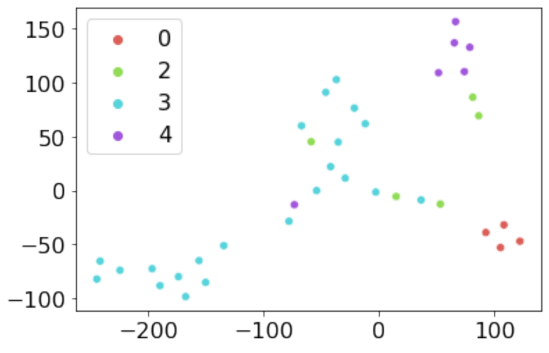
node labels.
Figure 7.12 shows a 2D representation of the node embeddings generated by GATH before the final linear layer used for classification on the test set. Dimensionality reduction is performed using TSNE. It can be noticed that the classes for `Cora`, `Citeseer`, `Cornell`, `Texas` and `Wisconsin` appear to be well separated. Indeed, it is possible to identify different clusters corresponding to different labels, and this is in line with the high classification accuracy achieved on these graphs. On `Chameleon`, it can be observed that classes 0 and 1 are separated from classes 2, 3 and 4, but the separation within these two subgroups is less evident. This is consistent with the findings derived from the CCNS matrixes in Figure 3.4, where we observed that the neighborhood patterns of classes in `Chameleon` allow for an effective separation of classes 0 and 1 from classes 2, 3 and 4. In conclusion, on `Squirrel` and `Film`, the separation among embeddings in different classes is not clear, which is consistent with the low classification accuracies of GATH on these two datasets.
Figure 7.13 shows the same representations for GCNH. Similar considerations can be made for `Cora`, `Citeseer`, `Cornell`, `Texas` and `Wisconsin`, for which the embeddings are quite well separated into clusters for the different classes. On `Chameleon`, the separation among classes appears to be better than the one achieved by GATH, which is consistent with the significant performance improvement of GCNH on this graph. However, GCNH identifies several smaller clusters of nodes belonging to the same class, instead of mapping all nodes belonging to the same class to the same region in the latent space. Embeddings for `Film` and `Squirrel` still appear not well separated.



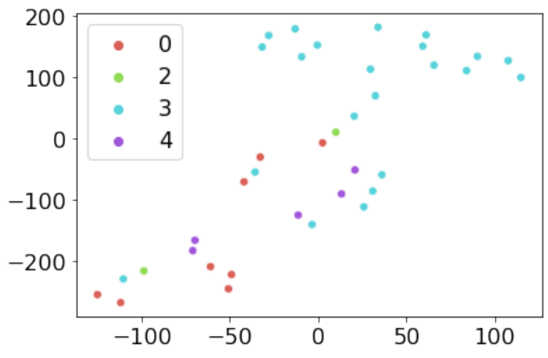
(a) Cora



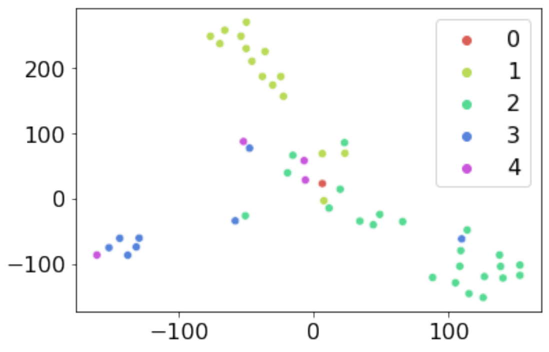
(b) Citeseer



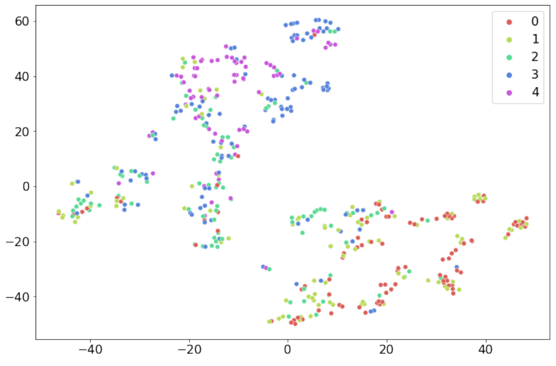
(c) Cornell



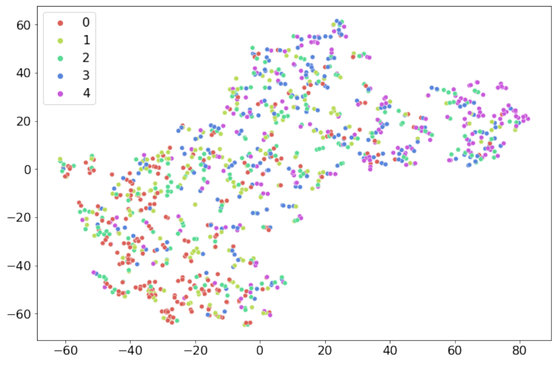
(d) Texas



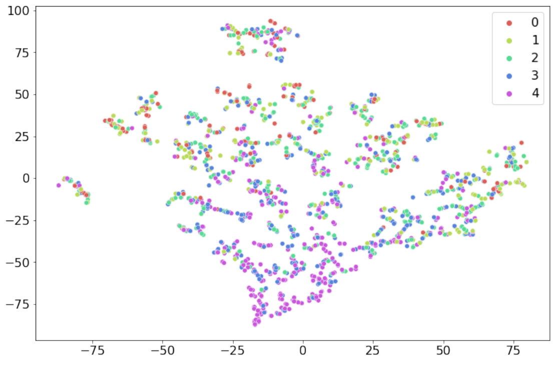
(e) Wisconsin



(f) Chameleon

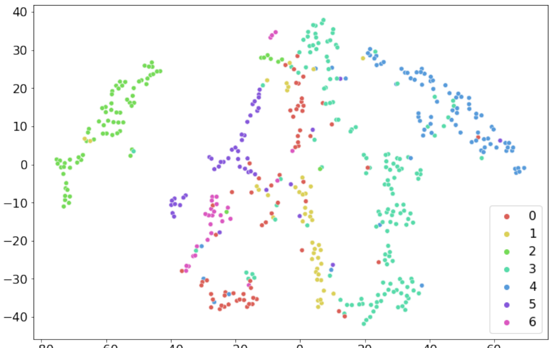


(g) Squirrel

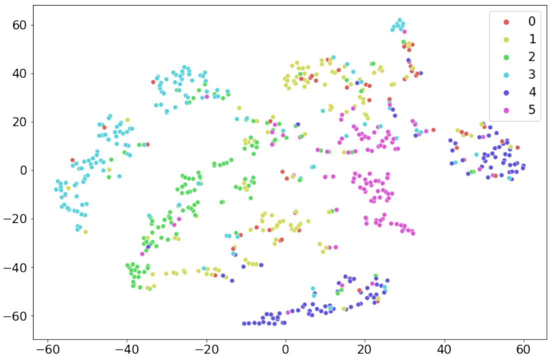


(h) Film

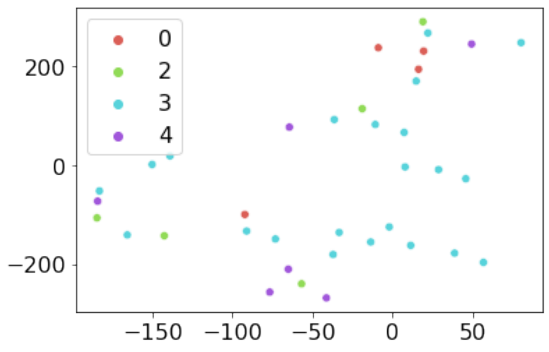
Figure 7.12: 2D visualization of node embeddings generated by GATH. Dimensionality reduction is performed by means of TSNE.



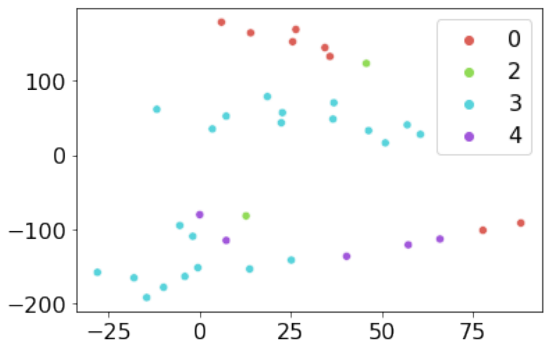
(a) Cora



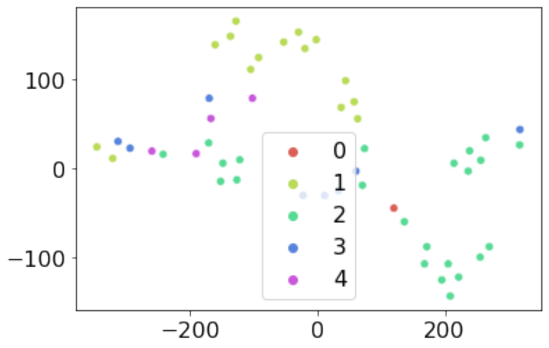
(b) Citeseer



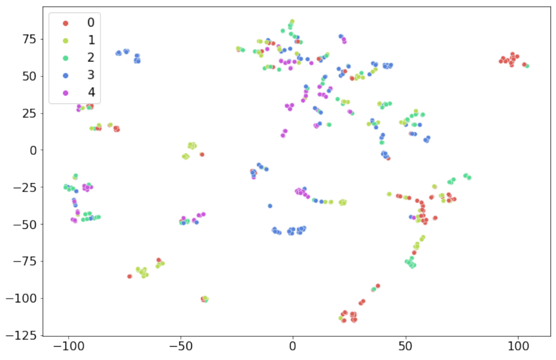
(c) Cornell



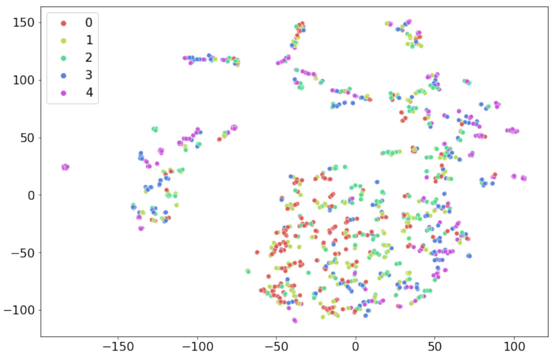
(d) Texas



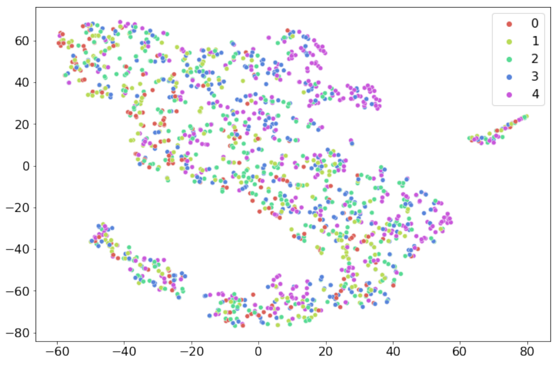
(e) Wisconsin



(f) Chameleon



(g) Squirrel



(h) Film

Figure 7.13: 2D visualization of node embeddings generated by GCNH. Dimensionality reduction is performed by means of TSNE.

Chapter 8

Conclusions

This chapter presents the conclusions of the thesis. First of all, we report the main findings in relation to the research questions introduced at the beginning. Then, we elaborate on the limitations and possible future developments for the work performed.

8.1 Main findings

We present the main findings of this thesis with reference to the research questions presented in Section 1.2.

RQ1. Do GNNs perform badly on all heterophilous graphs? Are there other metrics beyond heterophily that can be useful to characterize GNN performance on graphs?

Previous works [20] show that GNNs can perform well on some heterophilous graphs. The work carried out in this thesis confirms this by providing examples of graphs with low heterophily values where a GCN still achieves good classification results (e.g. Figure 5.3). However, as pointed out by [41], heterophilous graphs are in general harder to classify than homophilous graphs, and this motivates a deeper analysis of GNN performance on them.
Moreover, this thesis introduces 2-hop Neighborhood Class Similarity (2NCS), a new metric that measures the average over neighbors of a target node of the percentage of their neighbors with the same label as the target node. 2NCS measures a property that is relevant for GCN representation capabilities: if a node has a high 2NCS value, then it is easier for a GCN to classify it correctly. 2NCS is a relevant metric also because it provides additional information with respect to homophily. Indeed, high values of homophily correspond to high

values of 2NCS, but graphs with low homophily ratios can still have high 2NCS values, and these witness good GCN performance, as in the example reported in Figure 5.3.

RQ2. Is it possible to extend the design of GNNs in order to improve performance on heterophilous graphs?

GNNs can be extended to improve their performance on heterophilous graphs. Indeed, this thesis introduces two new GNN-based models that achieve competitive performance with respect to state-of-the-art approaches on real and synthetic graphs whose homophily ratios cover a large range of values. The first model, GATH, leverages the attention mechanism to learn a weight between two nodes, which is then used during message aggregation. Moreover, GATH aggregates information from non-connected nodes, and this turns out to be particularly effective on some heterophilous benchmarks where the useful information is located in different portions of the graph. In conclusion, GATH implements skip-connections to balance the contribution of the self-node and neighborhood embeddings, which proves to be fundamental to increase performance on some heterophilous graphs.
The second model, GCNH, is a simpler variation of GCN where skip-connections are implemented following the same formulation of GATH. Despite its simplicity, GCNH achieves very good results on all graphs, outperforming GATH and several state-of-the-art models on some benchmarks.

8.2 Limitations

One limitation of this thesis are the datasets used for evaluation. The choice of the datasets is motivated by the fact that they are the same graphs used in most of the works dealing with GNNs for heterophily. However, the 8 real-world graphs presented in Section 4.1 show some weaknesses. First of all, their size is generally small. `Cornell`, `Texas` and `Wisconsin` only contain about 200 nodes, which results in very large differences in accuracy among different runs, as witnessed by the large standard deviations reported for results on these graphs (e.g. Table 7.1). Furthermore, the biggest graph is `Film` with 7600 nodes, which is still not representative of the large-scale graphs containing million of nodes that are currently used for other graph machine learning tasks. Moreover, the graphs are either very homophilous ($h \approx 0.8$) or very heterophilous ($h \approx 0.2$), so they are not representative of the complete range of homophily values. The synthetic graphs used try to solve this problem, but synthetic graphs have limitations on their own, as they might not be representative of the different properties of real-world graphs.

We already present some limitations of 2NCS in Section 5.4. An additional weakness of this metric is that it is derived through intuitions, but it lacks a theoretical proof of its relation with GCN performance.
In conclusion, evaluations for GATH and GCNH are limited by the computational time and resources available for the experiments.

8.3 Future work

[15] have pointed out the limitations related to the datasets commonly used for heterophily and they have proposed 6 larger graphs with low homophily ratio. The evaluation of GATH and GCNH on these graphs is not performed within the context of this thesis because of the large computational time and resources necessary for the experiments. However, future developments of this work will include the evaluation of the proposed models on these larger benchmarks to assess their scalability.
To better characterize 2NCS, future works will focus on a stronger theoretical motivation of the metric and a deeper analysis of the mathematical relationship between 2NCS and GCN learning capabilities. Another interesting development consists in extending the definition of 2NCS to incorporate node features and model their impact on GCN performance. This, however, introduces a significant increase in the complexity of the scenario used for the analysis, which might lead to the impossibility to effectively incorporate node features in the metric. Indeed, the goal of 2NCS is to describe one specific graph property that is related to GCN performance, not to characterize GCN performance under all possible scenarios, given the variability of the information GCN can learn.
Another future expansion starts from the observation that GATH performance is dependent on the density of the graph. Indeed, the number of neighbors to perform message aggregation might explode when k is large and the graph is dense. To tackle this problem, one possible solution could be to sample the neighborhood of a node instead of considering it in its entirety. Several neighbor sampling strategies exist in the literature [50] and they allow for an increased scalability since the neighborhood size can be fixed as hyperparameter. However, neighborhood sampling has two main drawbacks. First of all, it slows down the pre-processing of the graph, resulting in very long execution times on large graphs, especially for sophisticated sampling strategies. Secondly, it does not allow to obtain a complete view of the neighborhood at once. To face this problem, it is common to perform several rounds during inference such that a different neighborhood is sampled at each round [51]. By averaging the results of different rounds, it is possible to obtain a complete view of the neighborhood. This process, however, causes a significant overhead, and therefore we do not implement it for the models presented in this

thesis, since the size of the graphs does not require it.

Appendix A

Experimental details

This Appendix reports additional details for the various experiments presented in the work.

A.1 Baselines on synthetic datasets

Figures 3.1, 7.1, 7.2a and 7.2b report results for three simple baselines (MLP, GCN and GAT) on the synthetic graphs. We have optimized the hyperparameter configurations for these models among the values reported in Table A.1. For all experiments on the synthetic datasets, we randomly generate train/evaluation/test splits with sizes 50%/20%/30%. We compute results on a single split for each dataset and we average the results of three different graphs for each level of homophily ratio reported in the figures.

Model	**#Epochs**	**#Layers**	**Hidden size**
GCN	{100}	{1,2,3}	{16}
GAT	{100}	{1,2,3}	{16}
MLP	{100}	{2}	{16,32}

Table A.1: Hyperparameters for GCN, GAT and MLP on synthetic graphs.

A.2 GCN on real-world graphs

In Sections 3.2 and 5, we report results for different variations of a GCN on the real-world graphs. In particular, Tables 3.1 and 5.2 show the performance of a standard GCN with 1 layer. The adjacency matrix is not normalized and we optimize the number of training epochs within the ranges reported in Table A.2.

Although the search space for hyperparameters is smaller than the one used by other works [10], it is sufficient to observe relevant differences in performance among datasets with similar homophily ratio, thus justifying the need for a new metric to be introduced.

Dataset	**#Epochs**
`Cornell`, `Texas`, `Wisconsin`	{100,200,300}
`Film`	{50,100}
`Chameleon`, `Squirrel`	{500,1000,1500,2000}
`Cora`, `Citeseer`	{100,200}

Table A.2: Number of epochs for GCN on real graphs.

A.3 2NCS evaluation

Section 5.3 reports several experiments showing the usefulness of 2NCS as a metric to understand GCN performance on graphs. In Figure 5.4, we report the classification accuracies of two variations of GCN. The simplified GCN is described in Section 5.2.1, whereas the standard GCN is the same presented in Appendix A.2. For both models, we perform a hyperparameter search using the values reported in Table A.4.
Then, Figures 5.5 and 5.6 report GCN and MLP performance with respect to node-level or class-level 2NCS values. Also in this case, the GCN is the same described in Appendix A.2 and the hyperparameters used for both models are the best ones from previous experiments. The plots show results for nodes in the test set for one of the splits taken from [18]. 2NCS is computed only on the nodes of the training set, since labels of nodes in the evaluation and test set are supposed to be unknown during training.

A.4 GATH and GCNH results

We optimize the hyperparameters for GATH and GCNH on real-world graphs through a grid search. In particular, we test different values for all of them and we select the best combination according to the best performance on the validation set. The main hyperparameters to be optimized for GATH are listed in the following.

- **k**: neighborhood size, i.e. message aggregation is performed on nodes up to the k-hop neighborhood from the target node

- **Batch size**: how many samples are contained in each batch. If batch size = $|V|$, then the complete training set is forwarded through the model at once at each epoch
- **#Epochs**: how many epochs are used for training
- **Hidden size**: the dimensionality of the generated node embeddings which will then be inputed to the final linear layer for classification. Note that there is also another hidden dimension in the attention computation of GATH. This dimension is set to 16 and it is not optimized for reasons of time
- **Dropout rate**: the percentage of attention weights that are randomly set to zero at each epoch. Setting dropout rate = 0 is equivalent to not using dropout at all

The same hyperparamters are present also in GCNH, with the exception of k, since GCNH does not aggregate from a larger neighborhood than the 1-hop. The tested values of hyperparameters for GATH are reported in Table A.3 and those for GCNH in Table A.4. We chose the numbers of epochs based on observations related to the evolution of training and evaluation losses on different datasets.

Dataset	**k**	**Batch size**	**#Epochs**	**Hidden size**	**Dropout rate**
`Cornell`, `Texas`, `Wisconsin`	{2,3,4,5}	{50,$\|V\|$}	{100,200,300}	{16,32}	{0.0,0.25,0.5}
`Film`	{2,3,4}	{300}	{50}	{16,32}	{0.0,0.25,0.5}
`Chameleon`, `Squirrel`	{2,3,4}	{300,$\|V\|$}	{50,100}	{16,32}	{0.0,0.25,0.5}
`Cora`, `Citeseer`	{2,3,4}	{300,$\|V\|$}	{100}	{16,32}	{0.0,0.25,0.5}

Table A.3: Hyperparameters for GATH on real graphs.

Dataset	**Batch size**	**#Epochs**	**Hidden size**	**Dropout rate**
`Cornell`, `Texas`, `Wisconsin`	{50,$\|V\|$}	{100,200,300}	{16,32}	{0.0,0.25,0.5}
`Film`	{300}	{50}	{16,32}	{0.0,0.25,0.5}
`Chameleon`, `Squirrel`	{300,$\|V\|$}	{500,1000}	{16,32}	{0.0,0.25,0.5}
`Cora`, `Citeseer`	{300,$\|V\|$}	{100}	{16,32}	{0.0,0.25,0.5}

Table A.4: Hyperparameters for GCNH on real graphs.

For the experiments on synthetic graphs, we apply the same grid search used for `Cora` and the best performing hyperparameter configuration is selected for each single graph.

A.5 Ablation studies

We obtain the results for the different variations of GATH on the synthetic datasets (Figures 7.2a, 7.3 and 7.4) and the values of the learned parameters on real-world graphs (Figures 7.9a, 7.10 and 7.11) using the best hyperparameter configuration obtained from the grid search described in Appendix A.4.

Bibliography

[1] M. Gori, G. Monfardini, and F. Scarselli. «A new model for learning in graph domains». In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 2. 2005, 729–734 vol. 2. DOI: 10.1109/IJCNN.2005.1555942 (cit. on p. 6).

[2] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. «The Graph Neural Network Model». In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80. DOI: 10.1109/TNN.2008.2005605 (cit. on p. 6).

[3] Jure Leskovec. *Graph Neural Networks 1: GNN Model.* University Lecture. 2020. URL: http://snap.stanford.edu/class/cs224w-2020/slides/06-GNN1.pdf (cit. on p. 6).

[4] Thomas N. Kipf and Max Welling. «Semi-Supervised Classification with Graph Convolutional Networks». In: *International Conference on Learning Representations (ICLR).* 2017 (cit. on pp. 7, 53).

[5] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. «Graph Attention Networks». In: *International Conference on Learning Representations* (2018). accepted as poster. URL: https://openreview.net/forum?id=rJXMpikCZ (cit. on pp. 8, 46–48, 53, 58).

[6] Shaked Brody, Uri Alon, and Eran Yahav. «How Attentive are Graph Attention Networks?» In: *International Conference on Learning Representations.* 2022. URL: https://openreview.net/forum?id=F72ximsx7C1 (cit. on pp. 9, 46, 48, 58).

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate.* 2014. DOI: 10.48550/ARXIV.1409.0473. URL: https://arxiv.org/abs/1409.0473 (cit. on p. 9).

[8] Miller Mcpherson, Lynn Smith-Lovin, and James Cook. «Birds of a Feather: Homophily in Social Networks». In: *Annual Review of Sociology* 27 (Jan. 2001), pp. 415–. DOI: 10.3410/f.725356294.793504070 (cit. on p. 10).

[9] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. «Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs». In: *Advances in Neural Information Processing Systems.* Vol. 33. Curran Associates, Inc., 2020, pp. 7793–7804. URL: `https://proceedings.neurips.cc/paper/2020/hash/58ae23d878a47004366189884c2f8440-Abstract.html` (visited on 04/07/2022) (cit. on pp. 10, 17, 18, 21, 30, 53).

[10] Yujun Yan, Milad Hashemi, Kevin Swersky, Yaoqing Yang, and Danai Koutra. «Two Sides of the Same Coin: Heterophily and Oversmoothing in Graph Convolutional Neural Networks». In: *arXiv:2102.06462 [cs]* (Nov. 2021). arXiv: 2102.06462. URL: `http://arxiv.org/abs/2102.06462` (visited on 02/09/2022) (cit. on pp. 10, 21, 53, 76).

[11] Di Jin, Zhizhi Yu, Cuiying Huo, Rui Wang, Xiao Wang, Dongxiao He, and Jiawei Han. «Universal Graph Convolutional Networks». In: *NeurIPS.* 2021 (cit. on pp. 10, 17–19).

[12] Lun Du, Xiaozhou Shi, Qiang Fu, Xiaojun Ma, Hengyu Liu, Shi Han, and Dongmei Zhang. «GBK-GNN: Gated Bi-Kernel Graph Neural Networks for Modeling Both Homophily and Heterophily». In: *arXiv:2110.15777 [cs]* (Apr. 2022). arXiv: 2110.15777. URL: `http://arxiv.org/abs/2110.15777` (visited on 05/18/2022) (cit. on pp. 10, 20).

[13] Xin Zheng, Yixin Liu, Shirui Pan, Miao Zhang, Di Jin, and Philip S. Yu. «Graph Neural Networks for Graphs with Heterophily: A Survey». In: *arXiv:2202.07082 [cs]* (Feb. 2022). arXiv: 2202.07082. URL: `http://arxiv.org/abs/2202.07082` (visited on 04/12/2022) (cit. on pp. 10, 17, 18).

[14] Kenta Oono and Taiji Suzuki. «Graph neural networks exponentially lose expressive power for node classification». In: *ICLR* (2020) (cit. on p. 10).

[15] Derek Lim, Felix Matthew Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Prasad Bhalerao, and Ser-Nam Lim. «Large Scale Learning on Non-Homophilous Graphs: New Benchmarks and Strong Simple Methods». In: *Advances in Neural Information Processing Systems.* Ed. by A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan. 2021. URL: `https://openreview.net/forum?id=DfGu8WwT0d` (cit. on pp. 11–13, 73).

[16] M.E.J. Newman. «Mixing patterns in networks». In: *Physical review. E, Statistical, nonlinear, and soft matter physics* 67 (Mar. 2003), p. 026126. DOI: `10.1103/PhysRevE.67.026126` (cit. on p. 12).

[17] Susheel Suresh, Vinith Budde, Jennifer Neville, Pan Li, and Jianzhu Ma. «Breaking the Limit of Graph Neural Networks by Improving the Assortativity of Graphs with Local Mixing Patterns». In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining.* New York, NY, USA: Association for Computing Machinery, Aug. 2021, pp. 1541–1551. ISBN: 978-1-4503-8332-5. URL: `https://doi.org/10.1145/3447548.3467373` (visited on 02/08/2022) (cit. on pp. 13, 19, 21, 22, 47).

[18] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. «Geom-GCN: Geometric Graph Convolutional Networks». In: *International Conference on Learning Representations.* 2020. URL: `https://openreview.net/forum?id=S1e2agrFvS` (cit. on pp. 13, 19, 30, 53, 54, 76).

[19] Leto Peel, Jean-Charles Delvenne, and Renaud Lambiotte. «Multiscale mixing patterns in networks». In: *Proceedings of the National Academy of Sciences* 115.16 (2018), pp. 4057–4062. DOI: `10.1073/pnas.1713019115`. eprint: `https://www.pnas.org/doi/pdf/10.1073/pnas.1713019115`. URL: `https://www.pnas.org/doi/abs/10.1073/pnas.1713019115` (cit. on p. 13).

[20] Yao Ma, Xiaorui Liu, Neil Shah, and Jiliang Tang. «Is Homophily a Necessity for Graph Neural Networks?» In: *International Conference on Learning Representations.* 2022. URL: `https://openreview.net/forum?id=ucASPPD9GKN` (cit. on pp. 17, 24–27, 32, 54, 71).

[21] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. «MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing». In: *Proceedings of the 36th International Conference on Machine Learning.* Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 21–29. URL: `https://proceedings.mlr.press/v97/abu-el-haija19a.html` (cit. on pp. 18, 30).

[22] Yu Wang and Tyler Derr. «Tree Decomposed Graph Neural Network». In: *Proceedings of the 30th ACM International Conference on Information and; Knowledge Management.* New York, NY, USA: Association for Computing Machinery, 2021, pp. 2040–2049. ISBN: 9781450384469. URL: `https://doi.org/10.1145/3459637.3482487` (cit. on p. 18).

[23] Sunil Kumar Maurya, Xin Liu, and Tsuyoshi Murata. «Simplifying approach to node classification in Graph Neural Networks». In: *Journal of Computational Science* 62 (2022), p. 101695. ISSN: 1877-7503. DOI: `https://doi.org/10.1016/j.jocs.2022.101695`. URL: `https://www.sciencedirect.com/science/article/pii/S1877750322000990` (cit. on pp. 19, 21, 30).

[24] Meng Liu, Zhengyang Wang, and Shuiwang Ji. *Non-Local Graph Neural Networks.* 2021. URL: `https://openreview.net/forum?id=heqv8eIweMY` (cit. on p. 19).

[25] Tianmeng Yang, Yujing Wang, Zhihan Yue, Yaming Yang, Yunhai Tong, and Jing Bai. *Graph Pointer Neural Networks.* 2021. DOI: `10.48550/ARXIV.2110.00973`. URL: `https://arxiv.org/abs/2110.00973` (cit. on p. 19).

[26] Tao Wang, Rui Wang, Di Jin, Dongxiao He, and Yuxiao Huang. *Powerful Graph Convolutioal Networks with Adaptive Propagation Mechanism for Homophily and Heterophily.* 2021. DOI: `10.48550/ARXIV.2112.13562`. URL: `https://arxiv.org/abs/2112.13562` (cit. on p. 19).

[27] Wei Jin, Tyler Derr, Yiqi Wang, Yao Ma, Zitao Liu, and Jiliang Tang. «Node Similarity Preserving Graph Convolutional Networks». In: *Proceedings of the 14th ACM International Conference on Web Search and Data Mining.* ACM. 2021 (cit. on p. 19).

[28] Dongxiao He, Chundong Liang, Huixin Liu, Mingxiang Wen, Pengfei Jiao, and Zhiyong Feng. «Block modeling-guided graph convolutional neural networks». In: *Proceedings of the AAAI Conference on Artificial Intelligence.* Vol. 36. 4. 2022, pp. 4022–4029 (cit. on p. 19).

[29] Xiang Li, Renyu Zhu, Yao Cheng, Caihua Shan, Siqiang Luo, Dongsheng Li, and Weining Qian. «Finding Global Homophily in Graph Neural Networks When Meeting Heterophily». In: *arXiv preprint arXiv:2205.07308* (2022) (cit. on p. 20).

[30] Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. «Beyond Low-frequency Information in Graph Convolutional Networks». In: *AAAI.* AAAI Press, 2021 (cit. on p. 20).

[31] Sitao Luan, Chenqing Hua, Qincheng Lu, Jiaqi Zhu, Mingde Zhao, Shuyuan Zhang, Xiao-Wen Chang, and Doina Precup. «Is Heterophily A Real Nightmare For Graph Neural Networks on Performing Node Classification?» en. In: (Sept. 2021). URL: `https://openreview.net/forum?id=LBv-JtAmm4P` (visited on 02/09/2022) (cit. on pp. 20, 21).

[32] Yang Ye and Shihao Ji. «Sparse Graph Attention Networks». In: *arXiv:1912.00552 [cs, stat]* (Apr. 2021). arXiv: 1912.00552. URL: `http://arxiv.org/abs/1912.00552` (visited on 05/31/2022) (cit. on p. 20).

[33] Liang Yang, Mengzhe Li, Liyang Liu, bingxin niu, Chuan Wang, Xiaochun Cao, and Yuanfang Guo. «Diverse Message Passing for Attribute with Heterophily». In: *Advances in Neural Information Processing Systems.* Ed. by A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan. 2021. URL: `https://openreview.net/forum?id=4jPVcKEYpSZ` (cit. on p. 20).

[34] Jiong Zhu, Ryan A. Rossi, Anup Rao, Tung Mai, Nedim Lipka, Nesreen K. Ahmed, and Danai Koutra. «Graph Neural Networks with Heterophily». In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.12 (May 2021), pp. 11168–11176. URL: `https://ojs.aaai.org/index.php/AAAI/article/view/17332` (cit. on pp. 21, 33).

[35] Cristian Bodnar, Francesco Di Giovanni, Benjamin Paul Chamberlain, Pietro Lio, and Michael M. Bronstein. «Neural Sheaf Diffusion: A Topological Perspective on Heterophily and Oversmoothing in GNNs». In: *ICLR 2022 Workshop on Geometrical and Topological Representation Learning*. 2022. URL: `https://openreview.net/forum?id=HtLzqEb1aec` (cit. on pp. 21, 24, 53, 54).

[36] Enyan Dai, Shijie Zhou, Zhimeng Guo, and Suhang Wang. *Label-Wise Message Passing Graph Neural Network on Heterophilic Graphs*. 2021. DOI: `10.48550/ARXIV.2110.08128`. URL: `https://arxiv.org/abs/2110.08128` (cit. on p. 21).

[37] Xiaojun Ma, Qin Chen, Yuanyi Ren, Guojie Song, and Liang Wang. «Meta-Weight Graph Neural Network: Push the Limits Beyond Global Homophily». In: *Proceedings of the ACM Web Conference 2022*. WWW '22. Virtual Event, Lyon, France: Association for Computing Machinery, 2022, pp. 1270–1280. ISBN: 9781450390965. DOI: `10.1145/3485447.3512100`. URL: `https://doi.org/10.1145/3485447.3512100` (cit. on p. 21).

[38] Zhewei Wei Ming Chen, Bolin Ding Zengfeng Huang, and Yaliang Li. «Simple and Deep Graph Convolutional Networks». In: (2020) (cit. on p. 22).

[39] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. «Adaptive Universal Generalized PageRank Graph Neural Network». In: *International Conference on Learning Representations*. 2021. URL: `https://openreview.net/forum?id=n6jl7fLxrP` (cit. on pp. 22, 53).

[40] Vijay Lingam, Rahul Ragesh, Arun Iyer, and Sundararajan Sellamanickam. *Simple Truncated SVD based Model for Node Classification on Heterophilic Graphs*. 2021. DOI: `10.48550/ARXIV.2106.12807`. URL: `https://arxiv.org/abs/2106.12807` (cit. on p. 22).

[41] Jiong Zhu and Danai Koutra. *Revisiting the problem of heterophily for GNNs*. Blog post. 2021. URL: `https://www.jiongzhu.net/revisiting-heterophily-GNNs/#the-effect-of-compatibility-matrices` (cit. on pp. 26, 71).

[42] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. «Social Influence Analysis in Large-Scale Networks». In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '09. Paris, France: Association for Computing Machinery, 2009, pp. 807–

816. ISBN: 9781605584959. DOI: 10.1145/1557019.1557108. URL: https://doi.org/10.1145/1557019.1557108 (cit. on p. 30).

[43] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. «Multi-Scale attributed node embedding». In: *Journal of Complex Networks* 9.2 (May 2021). cnab014. ISSN: 2051-1329. DOI: 10.1093/comnet/cnab014. eprint: https://academic.oup.com/comnet/article-pdf/9/2/cnab014/40435146/cnab014.pdf. URL: https://doi.org/10.1093/comnet/cnab014 (cit. on p. 30).

[44] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. «Collective Classification in Network Data». In: *AI Magazine* 29.3 (Sept. 2008), p. 93. DOI: 10.1609/aimag.v29i3.2157. URL: https://ojs.aaai.org/index.php/aimagazine/article/view/2157 (cit. on p. 30).

[45] Galileo Mark Namata, Ben London, Lise Getoor, and Bert Huang. «Query-driven Active Surveying for Collective Classification». In: *Workshop on Mining and Learning with Graphs*. 2012 (cit. on p. 30).

[46] Ryan A. Rossi and Nesreen K. Ahmed. «The Network Data Repository with Interactive Graph Analytics and Visualization». In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI'15. Austin, Texas: AAAI Press, 2015, pp. 4292–4293. ISBN: 0262511290 (cit. on p. 33).

[47] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. «Do Transformers Really Perform Badly for Graph Representation?» In: *Advances in Neural Information Processing Systems*. Ed. by A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan. 2021. URL: https://openreview.net/forum?id=OeWooOxFwDa (cit. on p. 47).

[48] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Deep Residual Learning for Image Recognition». In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90 (cit. on p. 47).

[49] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. «Densely Connected Convolutional Networks». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017 (cit. on p. 47).

[50] Xin Liu, Mingyu Yan, Lei Deng, Guoqi Li, Xiaochun Ye, and Dongrui Fan. «Sampling methods for efficient training of graph convolutional networks: A survey». In: *CoRR* abs/2103.05872 (2021). arXiv: 2103.05872. URL: https://arxiv.org/abs/2103.05872 (cit. on p. 73).

[51] Yixin Liu, Zhao Li, Shirui Pan, Chen Gong, Chuan Zhou, and George Karypis. «Anomaly Detection on Attributed Networks via Contrastive Self-Supervised Learning». In: *IEEE Transactions on Neural Networks and Learning Systems* (2021) (cit. on p. 73).