ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Sybil Detection using Graph Neural Networks

Master's Thesis

Stuart Heeb
stuart.heeb@inf.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Andreas Plesner
Prof. Dr. Roger Wattenhofer

September 10, 2024

Acknowledgements

First and foremost I would like to thank my supervisors, Prof. Dr. Roger Wattenhofer and Andreas Plesner for the opportunity to complete this thesis in the Distributed Computing group. In particular Andreas helped guide the process of writing this master thesis by providing valuable feedback and suggestions in our regular meetings. His insights into the intricate details of the topics were invaluable. He created a supportive and open environment that allowed me to grow and learn. Finally, I would like to thank my family and friends for their sustained support and encouragement during the time of writing this thesis.

Abstract

This thesis explores the application of Graph Neural Networks (GNNs) for Sybil detection in social networks and presents three novel approaches based on GNN architectures that leverage only the structural properties of the network: SybilGCN, SybilRGCN and SybilGAT.

In recent years, online social networks platforms have experienced tremendous growth. Sybil entities—fake or malicious accounts—threaten the integrity of such networks by spreading misinformation, perpetrating scams or spamming. Existing algorithms for structure-based Sybil detection tend to struggle in the presence of many attack edges. Our proposed methods address these limitations. The evaluation compares these algorithms against baselines SybilRank, SybilBelief and SybilSCAR on both synthetic and real-world social network graphs. Experiments include pre-training on sampled subgraphs, robustness evaluations across different network models and sizes and adversarial attack scenarios, and performance assessment on a real-world labeled Twitter network graph.

Our results demonstrate that the proposed GNN-based approaches consistently outperform existing algorithms, particularly in scenarios with high attack complexity and numerous compromised edges, without compromising evaluation runtime. The evaluation on the Twitter dataset consisting of more than 269k nodes and 6.8M edges confirms these findings.

This research contributes to the field of social network security by presenting effective, structure-based Sybil detection methods that are adaptable to different network structures and resilient against sophisticated attack strategies. The findings pave the way for future work in preserving social network integrity and user protection.

Contents

Chapter 1
Introduction

In the course of the last decade, online social networks have evolved from simple platforms for communication into ecosystems that influence almost every aspect of life. From personal interactions to political content and marketing, platforms like Instagram, Facebook[1] and Twitter[2] have become hubs for information exchange, servicing billions of users every day. This rapid growth has been accompanied by a vast set of problems, particularly Sybil entities, which are fake or malicious accounts aiming to deceive, manipulate or exploit the network and its (real) users.

Sybil attacks are a persistent threat to the integrity and security of social networks. Sybil accounts are used to spread spam or misinformation, perpetrate scams, commit fraud or negatively influence the system in other ways. The effects of these actions can be detrimental and must be addressed. Therefore, Sybil detection has become a critical area of research, aiming to preserve the authenticity of user interactions and to protect users. Traditionally, the countermeasures have included both structure-based and content-based approaches or combinations thereof, often relying heavily on the latter. However, with the rise of modern generative artificial intelligence (GenAI) technologies, automated accounts can now produce highly sophisticated text or images and even interact with other content in the network. As a result, content-based detection methods are facing diminishing returns, as the line between human-created and artificially-generated content becomes harder to discern—even by humans.

In light of these developments, some previous research [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12] has resorted to developing algorithms that rely solely on the structural properties of the network (and a small set of known nodes). Sybil entities often exhibit distinctive patterns of behavior, which include the way in which they connect to other entities in the network. By examining these structural characteristics of the network, it is possible to detect Sybil accounts.

[1] http://www.facebook.com.

[2] http://www.twitter.com. Twitter is now named X, http://www.x.com. Since all the relevant research and data considered in this work is from before this name change, we will refer to the platform as Twitter.

This work aims to do this by utilizing Graph Neural Networks (GNNs).

GNNs are a powerful tool for analyzing graph-structure data, making them a promising tool for performing this task on a social network where the nodes represent users (some of which are Sybils) and the edges represent trust relationships (that can be compromised by Sybils) or interactions between users. GNNs have gained a lot of traction in recent years, but, to our knowledge, have yet to be applied to perform structure-based Sybil detection without considering any other features. By aggregating information along the edges of a network, GNNs offer the same basis of propagation that previous work, including the state-of-the-art algorithms, use to detect Sybils [4, 5, 11, 12]. On top of this, GNNs are capable of learning weights, potentially being able to adapt better to different network graphs, offering more flexibility.

We show that, utilizing Graph Convolutional Networks (GCNs), Relational Graph Convolutional Networks (RGCNs) and Graph Attention Networks (GATs), different architectures of GNN models are capable of not only matching algorithms [4, 5, 11] presented in recent years but outperforming them in the task of structure-based Sybil detection social network graphs, particularly in scenarios with high attack complexity and many compromised edges. The experiments, utilizing both real and synthetic networks, include an evaluation after pre-training on sampled subgraphs, a robustness evaluation in terms of network model and size, an adversarial attack study and an evaluation on a Twitter network graph. Our approach shows robust performance across these experiments, even as the detection task becomes more challenging.

Chapter 2

Related Work

In this section we will lay out the most relevant past research on Sybil detection. The focus will be on structure-based Sybil detection, the distinction of which compared to other approaches will be discussed in Section 3.2, along with a formal definition of the problem itself. Additionally, we will bring up some important points about previous work in the area of social networks, which are the basis for many of these algorithms.

The algorithms that are chosen as baselines (bold below) for evaluation in this thesis are among the listed algorithms below and will be more thoroughly described in Section 3.3 and evaluated in Chapter 6.

Structure-based methods The majority of previous work considered for this research were structure-based methods, meaning that the only features available for Sybil detection are the graph structure and a set of known (honest and/or Sybil) nodes [1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12]. Most methods make use of the *homophily assumption*, which states that nodes connected by an edge tend to share the same label. Based on this assumption, the labels of a few known nodes are propagated through the social network. This is generally done using random walks (RW) [1, 2, 3, 4, 6, 7, 11] or loopy belief propagation (LBP) [5, 10, 8, 11, 12].

SybilGuard [1] and SybilLimit [2] assume that it is more likely for a random walk starting from a known honest node to reach other honest nodes than Sybil nodes, and vice versa. They use the same length of random walks for all nodes. While SybilGuard accepts $\mathcal{O}(\sqrt{n}\log n)$ Sybils per attack edge [1], it suffers from a high false-negative rate (FNR) [13]. The improved algorithm SybilLimit reduces the number of accepted Sybils per attack edge to $\mathcal{O}(\log n)$ while allowing more attack edges [2].

SybilInfer [3] is a centralized random walk procedure that uses a probabilistic model via Bayesian inference. This allows the algorithm to assign a degree of certainty besides just classifying the nodes.

SybilRank [4] uses early-stopping random walks to propagate trust scores rooting from a set of known honest nodes based on the assumption that the honest region of the network is fast-mixing. The trust scores are degree-normalized and ranked, allowing classification with a threshold value. The derived security guarantees are based on the assumption that the attack edges are randomly established between honest and Sybil nodes.

Existing random walk methods have the disadvantage that they can only leverage known honest nodes or known Sybil nodes, but not both simultaneously [14]. They also tend to lack robustness to label noise in the set of known nodes [11].

SybilBelief [5] utilizes a set of known honest nodes and, optionally, a set of known Sybil nodes to perform classification. Accounting for prior probabilities, the algorithm uses LBP to infer the posterior probabilities of nodes being Sybil.

SybilSCAR [11] aims to combine the approaches of random walks and LBP by introducing a novel local update rule which is applied iteratively for a given number of iterations or until convergence. The algorithm has two variants: SybilSCAR-C assumes **c**onstant homophily between any two nodes, and takes a parameter specifying this. Setting this parameter, however, usually requires some analysis of the full graph. The other variant is SybilSCAR-D which **d**ynamically computes homophily for each edge individually.

GANG [8] performs Sybil detection on directed social network graphs by incorporating different types of relationships (bidirectional, unidirectional) between users. If two users are linked bidirectionally and have the same label, the probability of their joint state is increased. The authors also derive an optimized version of GANG that removes the need for the actual message passing of LBP by reformulating the problem using matrices.

SybilHP [12] was developed for directed social networks and uses LBP combined with a homophily estimator to classify the nodes. It is designed to overcome the limitation many algorithms have of implicitly assuming constant homophily, which doesn't account for the directional nature of some social trust relationships.

Approaches based on LBP can incorporate knowledge about both known honest and known Sybil nodes simultaneously and tend to be more robust to label noise compared to RW-based methods [14].

Feature-augmented methods Increasingly, there has been research that utilizes additional features of nodes or other information to increase their detection accuracy [14, 15, 16, 17, 18]. Such features could be information about a node itself (e.g., number of friends of a user in a social network or their profile description), about edges or data about activity in the network (e.g., likes or content of posts or comments).

TrustGCN [15] leverages graph convolutional networks (GCNs) to classify nodes using a two-stage process: trust propagation (via random walks) and trust-guided graph convolution [19].

BotRGCN [16] is a Twitter bot detection algorithm that leverages different possible kinds of edges in a social network by applying a relational graph convolutional network (RGCN) [20].

SATAR [17] is a self-supervised representation learning framework for Twitter bot detection. It aims to adapt better to different types of social media bots and is proven to generalize in real-world scenarios.

Early Sybil detection Some methods specialize in early Sybil detection, using additional information about friend request targets and responses, along with the network topology. The motivation for these methods is the prevention of Sybils in the network, not just their detection after they have established themselves, aiming to avoid the negative effects they have on the network. SybilEdge [14] aggregates over these friend request features, giving more weight to the request targets that are preferred by other Sybil nodes (in contrast to honest nodes) while considering how these friend request targets respond. PreAttacK [18] uses initial friend request behaviors to perform classification by approximating the posterior probability that a new node is Sybil or not under their proposed multi-class Preferential Attachment model for unanswered friend requests. PreAttacK can successfully (AUC ≈ 0.9) detect Sybil nodes before any edges have been actualized (that is, friend requests have been accepted). These methods are, in a sense, a type of feature-augmented method, as they use additional information about the network and its nodes to perform classification, where these features are temporally relevant.

Social network research Research on social networks has played a crucial part in the development of Sybil detection algorithms, as it allows making important assumptions on which these algorithms are based.

These findings include that the node degree distribution of social network graphs is similar to a power law distribution [21, 22]. This means that very few nodes have a very high degree (a lot of edges), and most nodes have a low degree. Social networks have also been shown to have high clustering coefficients and short average path lengths [21, 22].

There are bodies of research inspecting the mixing time of social network, a crucial part and often a strong assumption for Sybil detection [23, 24]. While research shows that mixing time is slower than anticipated [23], it has still been a common assumption that the regions of a social network are fast-mixing.

Chapter 3

Background

All notations can be found in Table 3.1 at the end of this chapter.

3.1 Social Networks

3.1.1 Definition

In this section, we will formally introduce social networks that will be used to perform Sybil detection. The problem of Sybil detection is formally defined in Section 3.2.1.

Definition 3.1 (Social network)**.** A social network is a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. The nodes represent entities in the social network (e.g., users on a social media platform) and the edges represent some kind of trust relationship (e.g., a friendship connection on a social media platform).

The edges of a social network can be directed or undirected. Since, due to the nature of the available underlying data or for other reasons, much of the previous work focuses on the evaluation of undirected graphs [4, 5, 11].[1] We will follow suit in this. A directed graph can be transformed into an undirected graph by either transforming all edges into undirected ones and thereby keeping all edges, or only transforming bidirectional edges. Any such transformation will of course lose information that the directionality might have implied.

Definition 3.2 (Sybil and honest nodes)**.** A Sybil node is a node which is not in the intended set of nodes for a particular network and has some kind of adverse effect on the network (e.g., a fake user on a social media platform). A node that is not Sybil is honest.

For convenience, we will denote the set of Sybil nodes as S and the set of honest nodes as H. Of course, by construction, is it holds that $S \cap H = \emptyset$ and

[1] Many online trust relationships are naturally undirected.

$S \cup H = V$. In the sense of a binary classification task, we consider a Sybil node to be *positive* and an honest node to be *negative*.

Definition 3.3 (Sybil and honest regions)**.** The set of Sybil nodes in a social network form the Sybil region $G_S = (S, E_S)$ (where $S \subseteq V$, $E_S \subseteq E$), and the set of honest nodes form the honest region $G_H = (H, E_H)$ (where $H \subseteq V$, $E_H \subseteq E$).

In this thesis, we will assume that the social network is divided into these two regions, and that the Sybil region is controlled by an adversary entity. The regions themselves could be themselves divided into multiple structurally differentiable subregions (e.g., each representing a community), but we will consider them a "unit" in any case. Unless otherwise mentioned, we will assume that the two regions are of equal size.[2] These regions (whether they are real graphs or randomly generated ones) can be fused together with a synthetization scheme, which will be explained in Section 3.1.4. We will explore the effects of different region sizes in Section 6.5.2.

While edges represent trust relationships, edges between Sybil nodes and honest nodes are of course compromised by definition. Such edges are called *attack edges*.

Definition 3.4 (Attack edges)**.** An edge between a Sybil node and an honest node is called an *attack edge*. Formally, the set of attack edges is

$$E_T = E \setminus (E_H \cup E_S). \tag{3.1}$$

3.1.2 Assumptions

Assumptions about the underlying social network are the basis of Sybil detection. We present the most common assumptions or categories thereof. These assumptions are implicitly part of our process, as will become clear in Section 3.1.4, where we explain how social networks are commonly synthesized in literature.

Homophily The tendency of two connected nodes to have the same label is called *homophily*. This is a crucial assumption and is the basis of structure-based Sybil detection. Without it, the problem would be close to impossible to solve, since the trust relationships between nodes would lose all meaning and the problem would therefore be ill-defined in itself.

Mixing Time A common assumption is that the honest and Sybil regions of the social network are *fast-mixing* within themselves, but that the regions are *not* fast-mixing with each other. This is based on the assumption that nodes within

[2] This is commonly done in literature and allows using a real social network graph as the regions.

a "community" are well-connected, but nodes across communities are not. This assumption is rooted in the implicit assumption that Sybils have a more difficult time establishing trust relationships (edges) with honest nodes than honest nodes have with each other.

It is clear that the transformation from a directed to an undirected graph, or the other way around, can (and most likely will) change the mixing time of the involved social network. Generally, undirected graphs are faster mixing than directed graphs, but in some cases, directed graphs can be faster mixing than their undirected counterpart [23, 24].

Attack Edges Attack edges are commonly assumed to be limited, and at least somewhat randomly located. Particularly the latter part of this assumption is often violated in the presence of adversarial behavior, as will be explained in Section 3.5. We take the opportunity to challenge this assumption by evaluating social networks under adversarial attacks. Regarding the former part, of course an increasing number of attack edges will make the problem of Sybil detection harder to solve, since the honest and Sybil regions will become less distinguishable. We will revisit this paradigm multiple times throughout this thesis, and evaluate with increasing number of attack edges to see how the algorithms perform. There is a duality between this assumption and the last one, as the mixing-time assumption will generally hold when attack edges are sparse.

3.1.3 Power Law Distribution of Node Degrees

Analyzing real-world social network graphs, a trend appears: The node degrees tend to be distributed close to a power law distribution [22], shown in Figure 3.1. This means that very few nodes have a very high degree (a lot of connections), and most nodes have a low degree (*heavy-tailed distribution*). This is commonly called a *scale-free network* [22]. In the sense of social networks, Figure 3.1 is to be interpreted in the following way: The x-axis represents the node degree, and the y-axis represents the number of nodes with that degree.

Figures 4.1 to 4.3 in Chapter 4 show the node degree distribution for a real-world Facebook social network graph and for synthetically generated graphs which aim to mimic this property and are used for social network synthetization.

3.1.4 Social Network Synthetization

Due to the scarceness of real-world (labeled) social network graphs, previous research has resorted to using methods in order to synthesize social networks [1, 3, 2, 4, 25, 26, 7, 27, 11]. Given the assumption made in the beginning of this section that there is an honest region $G_H = (H, E_H)$ and Sybil region $G_S = (S, E_S)$, the

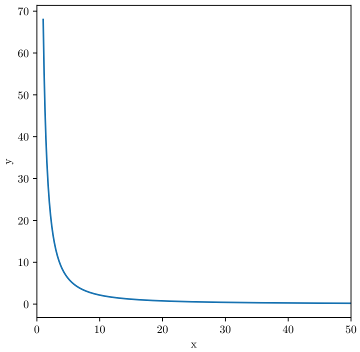

Figure 3.1: Power Law distribution.

goal is to synthesize these regions into a social network by adding attack edges. These region graphs can be real (Section 4.1) or synthetic (Section 4.2).

Most of the aforementioned research that employs this approach assumes these attack edges to be *fully random*, and limited (to adhere to the assumptions presented above). To that end, the honest region H and Sybil region S are randomly connected by a set of attack edges E_T, the size of which can be specified and, in this work, will be set as a number of attack edges *per Sybil*. Our procedure for adding random attack edges is as follows: Nodes $u \in H$ and $v \in S$ are picked uniformly at random, and the edge (u, v) is added to E_T (initially, $E_T = \{\}$). This is repeated until the desired number of attack edges is reached, where no duplicate edges are allowed. The final social network is then $G = (H \cup S, E_H \cup E_S \cup E_T)$. With this setup, there can be at most $|H| \cdot |S|$ attack edges. Reaching this maximum number would make the honest and Sybil region fully interconnected to each other (which is a very unrealistic scenario).

Adding fully random attack edge is the simplest approach, and the easiest in terms of solving the problem of Sybil detection. Section 3.5 explains how an adversarial attack can be imitated by placing attack edges in an adversarially more intelligent way, making the problem harder to solve.

Social networks are synthesized in this way with the desired *scale-free* property in mind, that is, that the node degrees still follow a power law distribution. This is important, and will be revisited in Section 4.2.

3.2 Sybil Detection

Two major approaches for countermeasures against Sybils in social networks are *Sybil detection* and *Sybil prevention*. Sybil detection aims to identify Sybil nodes in the network, while Sybil prevention aims to prevent Sybil nodes from being created in the first place, or before they can cause meaningful damage.

Roughly speaking, any such approach can be classified into two categories, or a combination thereof: *structure-based* or *content-based.* Structure-based approaches rely on the structure of the social network, while content-based approaches depend on content from the social network. Such content could be anything from posts made on a social media platform to additional features about specific nodes or edges.

This thesis focuses on the problem of Sybil detection when the only information available is the structure of the social network and a small set of known nodes.

Why no additional features? Several recent studies have studied Sybil detection using additional features [14, 15, 16, 17, 18]. This could include analyzing a user's posts or other activity on an online social network platform. We purposefully do not include such features for numerous reasons:

- With the rise of generative artificial intelligence (GenAI), humans can create human-like entities (e.g., bots, something that would be classified as Sybil) in social networks that become increasingly harder to distinguish from humans–even by humans. This makes it increasingly difficult to even manually label content-based features as originating from a Sybil or an honest node, let alone predicting them using automated classification methods.
- Including additional features always raises concerns about data privacy. By including fewer features (only the structure and some labels), we can circumvent most problems of this kind.[3]
- A structure-based approach offers enhanced generalizability across diverse social platforms and cultural contexts, as social network properties often remain consistent even when user behavior and content vary significantly.

The following section will formally introduce the problem setting of structure-based Sybil detection used in this thesis.

3.2.1 Problem Definition

Following the definitions in Section 3.1, we assume that the sets of Sybil nodes S and honest nodes H are divided into two disjoint sets, the training and the test set. We will denote the honest train and test set as H_{train} and H_{test} respectively, and the Sybil train and test set as S_{train} and S_{test} respectively. If not otherwise

[3] Of course in some cases, especially when data is sparse, the structure alone can allow someone to deduce the identity of some nodes. In such cases, this must still be addressed.

mentioned, we will assume that the training sets H_{train} and S_{train} are made up of 5% of their respective total sets, chosen uniformly at random.

Problem 3.5 (**Structure-based Sybil detection**)**.** Given a social network, a set of known honest nodes H_{train} and a set of known Sybil nodes S_{train}, the goal is to predict the set of remaining Sybil nodes S_{test}.

Implicitly, predicting S_{test} will also predict $H_{\text{test}} = V \setminus \{H_{\text{train}} \cup S_{\text{train}} \cup S_{\text{test}}\}$. Often times, as is the case with the novel algorithms presented in this thesis, an algorithm will solve this problem by estimating a probability for each node to be Sybil. This probability can then be used to predict the set of Sybil nodes by applying a threshold. The performance evaluation will be done on the test nodes. The metrics used for the evaluation of such a structure-based Sybil detection algorithm are presented in Section 4.3.2.

3.3 Baseline Algorithms

From the past research presented in Chapter 2, we have identified three algorithms that are commonly used as baselines for the Sybil detection problem, while fulfilling the requirements detailed in the problem definition in Section 3.2.1, namely performing Sybil detection on an undirected social network while only requiring the graph structure and a set of known nodes. These algorithms are SybilRank [4], SybilBelief [5], and SybilSCAR [11]. These will be our baselines. In this section, we will provide a more detailed overview of each of these algorithms.

3.3.1 SybilRank

SybilRank [4] is a ranking-based algorithm for Sybil detection that works by propagating trust starting from a small number of trust seeds through the network. It is designed for undirected social network graphs. Following our definitions in Sections 3.1 and 3.2, these trust seeds form the set H_{train}, where the algorithm takes no known Sybil nodes as input ($S_{\text{train}} = \varnothing$). These trust scores are propagated through the network using early-stopping random walks, based on the assumption that the honest region of the network is fast-mixing, but that it is not well-connected to the Sybil region. The algorithm performs $\mathcal{O}(\log n)$ iterations, where at each step the current trust of a node is computed by summing up the degree-normalized trust from the previous iteration of all neighbors. The final trust values are additionally degree-normalized, and are then used to rank the nodes in the network, and a threshold value or pivot index is used to determine the cut-off point between honest and Sybil nodes in this ranking list (the paper acknowledges that this step will most likely include some manual inspection).

Given the number of iterations the total asymptotic runtime of SybilRank is $\mathcal{O}(n \log n)$ for any number of trust seeds [4].

3.3.2 SybilBelief

The SybilBelief [5] algorithm takes as input a small set of known honest nodes, and optionally a small set of known Sybil nodes. Using these nodes it performs loopy belief propagation on an undirected social network graph to approximate the posterior probabilities of each node being Sybil. The propagation terminates after convergence (with $\epsilon = 10^{-3}$) or a maximum number of iterations d. This is accomplished by defining node potentials to incorporate prior knowledge and edge potentials to represent correlations between nodes. This implicitly allows a node ranking as with SybilRank. It has some advantages over SybilRank and other previous approaches, such as being able to incorporate both known honest and Sybil nodes and being more robust to label noise [5].

As in [5], we used $\{\theta^+, \theta^-, \theta\} = \{0.9, 0.1, 0.5\}$ as initial belief values. In order to facilitate a fair comparison, we provide SybilBelief with the optional set of known Sybil nodes.

With the complexity of one iteration being $\mathcal{O}(m)$, the total complexity of SybilBelief is $\mathcal{O}(m \cdot d)$, where m is the number of edges in the network. Since social networks are often sparse graphs, this makes the final asymptotic runtime $\mathcal{O}(m \cdot d) = \mathcal{O}(n \cdot d)$, which is similar to the runtime $\mathcal{O}(n \log n)$ of SybilRank by setting d accordingly, which typically results in good performance [5].

3.3.3 SybilSCAR

SybilSCAR [11] is a Sybil detection algorithm that aims to combine the advantages of random walk-based and LBP-based approaches by introducing a novel local update rule. This local rule is applied iteratively until convergence (with $\epsilon = 10^{-3}$) or until a maximum number of iterations d is reached. There are two variants of the algorithm: SybilSCAR-C and SybilSCAR-D. The former assumes constant homophily for all nodes, while the latter allows for different homophily values for each pair of connected nodes to be used (which are not continuously adjusted). SybilSCAR-C requires some knowledge about the full social network, which is why for our evaluation we will use the graph-adaptive SybilSCAR-D algorithm. SybilSCAR is shown to be more accurate than both SybilRank and SybilBelief, and more robust to label noise than the former. It also offers more advantages in terms of time and space complexity and convergence [11].

As in [11], we used $\{\theta^+, \theta^-, \theta\} = \{0.6, 0.4, 0.5\}$ as priors. We used the matrix form algorithm described in the paper [11] and optimized its runtime by using sparse matrix operations (allowing it to run in matrix form when evaluating large graphs such as the Twitter network). We tested against the public C++ code by the authors, and our implementation performed equally (up to numerical differences, and sometimes better) to the comparison. Due to this, we used our implementation.

The runtime of SybilSCAR is $\mathcal{O}(m \cdot d)$, where m is the number of edges in the network. It can be implemented in a parallel fashion by dividing nodes into groups [11].

3.4 Graph Neural Networks

This section provides a concise background of Graph Neural Networks (GNNs). Chapter 5 gives a detailed description of how these networks can be used for structure-based Sybil detection.

3.4.1 What are GNNs?

Graph Neural Networks (GNNs) are a class of neural networks that operate on graph-structured data. They are an extension of traditional neural networks that are designed to aggregate data along the edges of the graph. This allows GNNs to learn from the structure of the graph, as well as the features of the nodes and edges. GNNs can be used for a variety of tasks, such as node classification, graph classification or link prediction.

We will look at three major types of GNNs: Graph Convolutional Networks (GCNs) [19], Relational Graph Convolutional Networks (RGCNs) [20], and Graph Attention Networks (GATs) [28]. Each of these types of GNNs has its own set of properties and uses, which result in different strengths and weaknesses in graph-based Sybil detection and different settings thereof.

Graph Convolutional Networks (GCNs)

Similar to how Convolutional Neural Networks (CNNs) work on grid-structured data, Graph Convolutional Networks (GCNs) [19] are a type of neural network that operate on graph-structured data. After an aggregation step (akin to the convolution step), a GCN layer takes this representation and feeds it to a traditional neural network, that is determined by the number of input and output channels along with other hyperparameters. The aggregation step is often the mean, but can be some other kind of aggregation. Typically, activation functions are applied in between layers. The number of layers a network has determines the propagation “reach”, i.e., how many hops information can travel.

Relational Graph Convolutional Networks (RGCNs)

Relational Graph Convolutional Networks (RGCNs) [20] work much like GCNs, but they are capable to work on multi-relational graphs. Whereas in GCNs all edges are considered to be equal, RGCNs can take advantage of different edge types. These edge types are often rooted in the actual relationship they represent. One could, for example, learn different weights for different edges, where the edges could be “following”, “like”, or “repost” interactions between users on an online social network platform.

Graph Attention Networks (GATs)

Graph Attention Networks (GATs) [28] introduce an attention mechanism into the process. Such attention mechanisms have offered great success in other machine learning areas. The attention mechanism allows the model to focus on the most relevant neighboring nodes during aggregation by assigning different attention weights instead of treating all nodes equally. For this, one can specify a number of attention heads, which are then applied multiple times in parallel before combining the results.

3.4.2 Node Classification

One of the most common tasks for which GNNs are employed is node classification. This is also the most relevant task when considering Sybil classification as introduced in Section 3.2. The goal is to classify nodes into two classes based on the final output of the GNN.

Transductive vs. Inductive Learning

In general, there are two modes in which one can conduct node classification with GNNs, namely *transductive* and *inductive* learning.

Transductive learning involves training a GNN model on a specific dataset (graph), and predicting on that same graph. This could be in the setting of semi-supervised learning where a subset of labels is known, and the model is tasked with predicting all the node labels. The main disadvantage is that the model is usually not able to generalize to unseen graphs or datasets, but the model might be able to produce more accurate results on the graph it was trained on.

Inductive learning, on the other hand, involves training a GNN model on a specific dataset, and then predicting on a different, unseen graph. The graph the model is trained on can be fully or partially labeled. It is also possible to train on multiple (e.g., small) graphs before performing inference on an unseen graph. This approach could also be applied to graph classification, for example when trying to predict properties of molecules. Inductive learning is better suited when the graph structure is dynamic, or when classification needs to be performed without being able to re-train (e.g., time sensitive applications).

3.4.3 Other Methods with GNNs

GNNs are capable of performing other tasks, such as link prediction or graph classification. Such tasks could potentially be useful for Sybil detection, as one could use link prediction to predict attack edges instead of directly predicting Sybil nodes, or graph classification to classify an entire graph region as being Sybil. This is not the focus of this work, but could be an interesting direction for future research, see Section 7.2.

3.5 Graph Adversarial Attacks

We will give a short introduction to graph adversarial attacks adapted from [29], and then describe the targeted attack we employed in the evaluation of this thesis in detail.

3.5.1 Taxonomy on Graph Adversarial Attacks

Following the taxonomy from [29], we introduce the components of graph adversarial attacks. We adapt this taxonomy slightly in a way to make it applicable to the way one would attack structure-based Sybil detection algorithms in our problem setting. This adaptation removes the attack on the actual neural network but only considers attacking the problem parameters from Section 3.2. The targeted attack we employ and use as a stress test is described in Section 3.5.2.

Attacker's Capacity The capacity of an attacker is measured by the point in time when the attacker can perform the attack. The attacker can either perform the attack at training or inference. The former is called *poisoning attack*, as the attacker "poisons" the training process. The latter case is referred to as an *evasion attack*, where the attack happens after training.

Perturbation Type According to [29], there are three possible perturbation types: modifying features (in our case, the only features are the labels), adding or deleting edges and injecting nodes.

Attacker's Goal There are in principle two types of attacks, that are classified by the goal of the attacker: targeted attacks and untargeted attacks. A targeted attack aims to make the algorithm misclassify certain nodes of the test set, while an untargeted attack aims to make the algorithm misclassify any node and cause overall bad performance. Note that does distinction between targeted and untargeted attacks is not congruent with our use of the terms targeted and untargeted attack edges, but rather describes the intended effect of the attack.

Attacker's Knowledge The attacker's knowledge of the labels can be divided into three categories: white-box, gray-box and black-box knowledge. White-box knowledge means that the attackers knows the full training set, black-box knowledge means that the attacker knows nothing about the training set, and gray-box knowledge is when the attacker knows some of the labels.

3.5.2 Targeted Attack Edges

In this section, we describe a targeted attack on a social network that can be described as a poisoning (or in some cases, evasion) attack, performing an "untargeted attack" (with the goal of causing general loss in performance) via means of adding or removing edges of the social network. The attacker has a varying degree of knowledge of the training labels (gray or white-box knowledge). The act of modifying features (in our case, node labels), will be revisited in Section 6.2 in the robustness evaluation of label noise.

The setting for this attack is within the process of social network synthetization as described in Section 3.1.4. With a varying degree, a number of random and targeted attack edges are added to the network within this procedure, causing the graph to become poisoned in the sense described above. The number of attack edges m_T to be added in the network synthetization process is determined beforehand, and the specific attack determines how these edges are added. We will now describe the targeted attack used in this thesis in detail.

An attack edge is placed uniformly at random between target sets $T_H \subseteq H$ and $T_S \subseteq S$. Generally, it is assumed that a targeted attack edge "originates" from a Sybil node and "hits" an honest node. We will denote a general undirected attack edge as $e = (u, v)$, where $u \in T_H$ and $v \in T_S$. We will assume that $T_S = S$, since we consider the Sybil region to be an entity that tries to disturb the system as a whole. When a fully random attack edge is placed, it holds that $T_H = H$. The essence of the targeted attack is now to determine when a random attack edges is placed and when a targeted attack edge is placed (that is, how T_H is defined), and, in the presence of a targeted attack edge, how close to a target honest node that it "hits".

To this end, there are two parameters: the *targeted attack edge probability* $p_T \in [0, 1]$ and the *discrete hit distance probability distribution function (pdf)*

$\boldsymbol{p} \in [0,1]^{K+1}$, where K is the maximal hit distance.[4] We will define X_T as a random indicator variable denoting, for each edge placement, that a targeted attack edge is being placed. The opposite occurrence is denoted as $\overline{X}_T$ (a fully random attack edge being placed). Of course, attack edges placed randomly ($T_H = H$) can (and in some cases will) coincide with attack edges placed in a targeted fashion ($T_H \subsetneq H$).

Targeted edge probability p_T This describes the probability for any attack edge to be a targeted attack edge, rather than a random attack edge. That is,

$$\mathbb{P}\left[X_T = 1\right] = p_T \quad \text{and} \quad \mathbb{P}\left[\overline{X}_T = 1\right] = 1 - p_T. \tag{3.2}$$

When a random attack occurs, $T_H = H$ (as mentioned above, it is assumed $T_S = S$). Formally, the attack edge $e = (u, v)$ is generated with $u \in \text{Uniform}(H)$ and $v \in \text{Uniform}(S)$, as described in Section 3.1.4. Otherwise, $T_H = H_{\text{train}}$. In that case, the parameter $\boldsymbol{p}$ determines how close to a target honest node (which is an honest training node) the attack edge is placed.[5]

Discrete hit distance pdf $\boldsymbol{p}$ Given $\boldsymbol{p} = [p_0, p_1, \ldots, p_K]$ is a discrete pdf, it must sum to unity:

$$\sum_{i=0}^{K} p_i = 1 \tag{3.3}$$

The parameter $\boldsymbol{p}$ determines, given a targeted attack edge (u, v) is being placed (which happens with probability p_T), how close to a node $u' \in \text{Uniform}(H_{\text{train}})$ the honest node u is chosen. Formally, the pdf $\boldsymbol{p}$ describes the following probabilities:

$$p_k = \mathbb{P}\left[u \in N_k(u') \mid X_T = 1\right] \tag{3.4}$$

where $N_k(\cdot)$ denotes the k-hop neighborhood. The choice of $u \in N_k(u')$ is made uniformly at random. Note that this choice of neighborhood could flow into the other region, essentially placing an attack edge connecting two Sybils, which does not constitute an attack edge by the definition in Section 3.1. We will overlook this, as the attacks are designed in a way that this happens very rarely and does not affect the results significantly even when it does.

Figure 3.2 shows the targeted attack algorithm as a probability tree, in the way it is implemented.

[4] More formally, $\boldsymbol{p}$ is a function $\{0, \ldots K\} \to [0,1]$, but we will view it as a vector.

[5] Note that the attacking entity does not necessarily need white-box knowledge of the labels (training set), indeed a partial knowledge (gray-box) can be sufficient. In that case, the attacking entity uses the information available to them as a proxy for H_{train}.

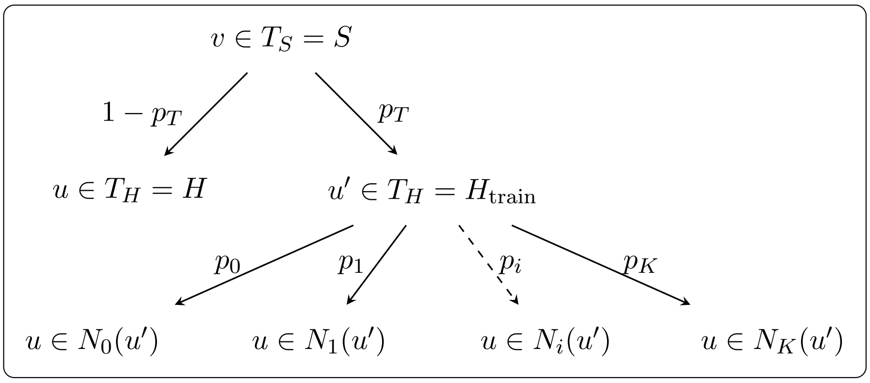

Figure 3.2: Probability tree visualization for the targeted attack.

After a targeted attack with parameters p_T and $\boldsymbol{p}$, the expected number of random attack edges is

$$\mathbb{E}\left[|E_{T_{\text{rand}}}|\right] = (1 - p_T) \cdot m_T. \tag{3.5}$$

The expected number of targeted attack edges hitting a node with distance k from $T_H = H_{\text{train}}$ is

$$\mathbb{E}\left[|E^{(k)}_{T_{\text{targ}}}|\right] = p_T \cdot p_k \cdot m_T. \tag{3.6}$$

Of course, the expected total number of attack edges is m_T:

$$\begin{aligned}
\mathbb{E}\left[|E_T|\right] &= \mathbb{E}\left[|E_{T_{\text{rand}}}| + \sum_{k=0}^{K} |E^{(k)}_{T_{\text{targ}}}|\right] && (3.7)\\
&= \mathbb{E}\left[|E_{T_{\text{rand}}}|\right] + \sum_{k=0}^{K} \mathbb{E}\left[|E^{(k)}_{T_{\text{targ}}}|\right] && (3.8)\\
&= (1 - p_T) \cdot m_T + \sum_{k=0}^{K} p_T \cdot p_k \cdot m_T && (3.9)\\
&= (1 - p_T) \cdot m_T + p_T \cdot m_T \underbrace{\sum_{k=0}^{K} p_k}_{=1} && (3.10)\\
&= m_T && (3.11)
\end{aligned}$$

Example 3.6 shows an instantiation of this attack which aims to explain the attack in a more practical way. A comprehensive evaluation of different instantiations of this attack and the effect on the analyzed algorithms can be found in Section 6.4.

Example 3.6 (Example of a targeted attack)**.** Assume that we are adding 1000 attack edges in the synthetization scheme from Section 3.1.4. Given a targeted attack with parameters $p_T = 0.1$ and $\boldsymbol{p} = (0.25, 0.25, 0.5)$, the expected number of random attack edges is $0.9 \cdot m_T$, i.e. 90% of all attack edges (900 attack edges). This means that, in expectation, 100 attack edges (10%) will be targeted. Out of these 100 attack edges, 25% will hit a known honest (train) node directly, 25% will hit a neighbor (which could be a train node itself, or in some cases, as mentioned above, even a Sybil node) and the remaining 50% will hit a node with distance 2 from a known honest node.

Description	**Notation**
Social network graph	$G = (V, E)$
Nodes	V, $\lvert V\rvert = n$
Edges	E, $\lvert E\rvert = m$
Honest region	$G_H = (H, E_H)$, $\lvert H\rvert = n_H$
Sybil region	$G_S = (S, E_S)$, $\lvert S\rvert = n_S$
Training (known) nodes	$(H_{\text{train}}, S_{\text{train}})$
Test nodes	$(H_{\text{test}}, S_{\text{test}})$
Attack edges	E_T, $\lvert E_T\rvert = m_T$
Targeted attack probability	p_T
Target hit distance pdf	$\boldsymbol{p}$
Random attack edges	$E_{T_{\text{rand}}}$
Targeted attack edges	$E_{T_{\text{targ}}}$
$E_{T_{\text{targ}}}$ with hit distance k	$E_{T^{(k)}_{\text{targ}}}$

Table 3.1: Notations.

Chapter 4

Data & Experiments

4.1 Real Datasets and Data Sources

Finding real social network data proves to be very difficult due to such as privacy concerns and—presumably—competitiveness of the corporations that own the data. However, there are some datasets that are publicly available and can be used for research purposes. In this section, we will present two such datasets which are used extensively in this work.

In the following we will make the distinction of *labeled* and *unlabeled* datasets. By this we mean that the labeled datasets contain information about the ground truth of the data, i.e., which nodes are honest and which are Sybil. The unlabeled datasets do not contain this information and are can therefore not be used directly for Sybil classification.

In Section 4.2 we will show how synthetic graphs can be generated which can in turn be used alongside real unlabeled graphs to synthesize labeled social networks as shown in Section 3.1.4. These synthesized networks can be used in the same way a real labeled social network would be used.

4.1.1 Social Media Platforms

While there are many other social media platforms, Facebook and Twitter are among the more popular choices not only for users but also for research in the area of Sybil detection. This is in part due to the availability of data,[1] which is why we will focus on these two platforms.

[1] The Twitter API has seen some changes in recent years which makes using it less feasible (more expensive) for dataset creation and curation.

Facebook

As briefly described in Chapter 2, there are some bodies of research that originated from Meta[2] [14, 18].[3] In addition to this, there is an *unlabeled* Facebook dataset on the SNAP[4] platform consisting of 4'039 nodes and 88'234 undirected edges [30, 31]. The dataset is a friendship network from Facebook, where the nodes are users and the edges are friendships between the users. Figure 4.1 shows the node degree distribution of the Facebook social network graph.

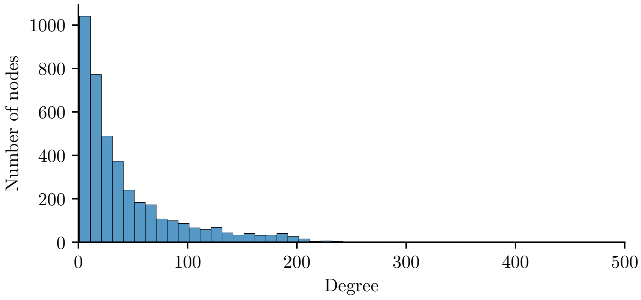

Figure 4.1: Node degree distribution of the Facebook graph.

Following previous research, we will be using this graph as regions of a synthesized social network [25, 26, 7, 11].

Twitter

As for the platform Twitter, there is a greater amount of available public datasets, for example the Twitter dataset, a real-world fully labeled social network graph consisting of 269'640 nodes and 6'818'501 edges [12]. The nodes represent users, and the directed edges represent the "following" relationship. This dataset was sampled and processed [12] from a previously much larger crawled graph [32]. Initially, the Twitter API was used to crawl the graph and then, retroactively and repeatedly over the past few years, used to determine which accounts were honest or Sybil accounts by querying their account status, effectively labelling the nodes.

The dataset consists of 178'377 honest nodes and 91'263 Sybil nodes. Of the honest nodes, $|H_{\text{train}}| = 20'000$ nodes are known ($\approx 11.2\%$), and of the Sybil nodes, $|S_{\text{train}}| = 10'000$ nodes are known ($\approx 10.9\%$). These training sets were provided with the dataset [12]. The remaining nodes $H_{\text{test}} \cup S_{\text{test}}$ are used for

[2] http://www.meta.com. Meta is the corporation behind Facebook.

[3] A majority of the authors were Meta/Facebook researchers.

[4] https://snap.stanford.edu/index.html.

evaluation, unlike the source paper of the dataset, where all nodes were used for evaluation [12].

4.1.2 Other Use Cases

As part of this thesis, we reached out to Swisscom[5] in hopes of a collaboration. With the rise of spam calls, that for some people can occur up to multiple times per week, we had the idea of using the implied directed graph, where nodes are people (or phone numbers) and the edges are attempted (answered and unanswered) calls, to identify which nodes are likely to be malicious (Sybil). With the help of the kind of data Swisscom could provide, this is an avenue that could have been further explored, allowing us to directly apply and test our research. Unfortunately, our contact attempt was left unanswered.

4.2 Synthetic Data

As shown in Section 3.1.4, real or synthetic regions can be used to create synthesized social networks by adding attack edges. In this section, we will present the synthetic regions used in this thesis.

4.2.1 Synthetic Regions

Previous work has often resorted to using synthetic graphs for the honest and Sybil regions [25, 26, 7, 11]. Of course, such graphs will be unlabeled. Among the numerous ways to generate these regions synthetically, we will present two of the most common methods found in literature. The parameters used are fine-tuned according to past research or adjusted based on the analysis of real-world social network characteristics [3, 4, 25, 33, 26, 34, 27].

Barabási-Albert Model

Barabási-Albert (BA) model generator creates a graph of n nodes by attaching new nodes (with $m \in \{1, \ldots, n\}$ edges each) that are preferentially attached to existing nodes with high degree [35]. We will evaluate different values for the parameter m, and will refer to this in the specific experiments.

Figure 4.2 shows the node degree distribution of a graph generated with the Barabási-Albert model with 4039 nodes and $m = 10$. The number of nodes was chosen to match the Facebook graph from Section 4.1.1.

[5] https://www.swisscom.ch/.

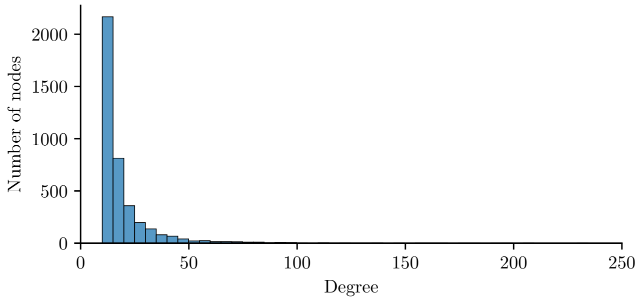

Figure 4.2: Node degree distribution of a Barabási-Albert graph with 4039 nodes and $m = 10$.

Power Law Model

The Power Law (PL) graph generator is an algorithm for generating graphs with power law distributed degrees, and approximate average clustering [36]. The parameters of this random graph generator are n (the number of nodes), $m \in \{1, \ldots, n\}$ (the number of random edges to add for each node), and $p \in [0, 1]$ (probability of adding a triangle after adding a random edge). This is equivalent to the BA model, but with the added chance (controlled by p) that a newly added random edge is closed to form a triangle [35]. Different values for m and p will be considered, and this will be clearly stated in the experiment setups. A graph generated with the PL model may be disconnected (but this did not happen in our experiments).

Figure 4.3 shows the node degree distribution of a graph generated with the Power Law model with 4039 nodes, $m = 10$, and $p = 0.8$.

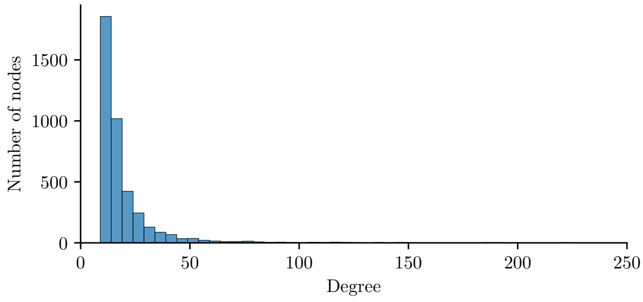

Figure 4.3: Node degree distribution of a Power Law graph with 4039 nodes, $m = 10$ and $p = 0.8$.

4.2.2 Synthesized Social Networks

As mentioned in Section 3.1.4, synthetic social networks can be created by fusing the honest and Sybil regions together. The honest and Sybil regions can be real graph, such as the Facebook graph from Section 4.1.1, or synthetic graphs such as the ones mentioned in Section 4.2.1.

Figure 4.4 shows a social network that has been synthesized with the Facebook graph from Section 4.1.1 by adding 4 random attack edges per Sybil, totaling 16'156 attack edges.

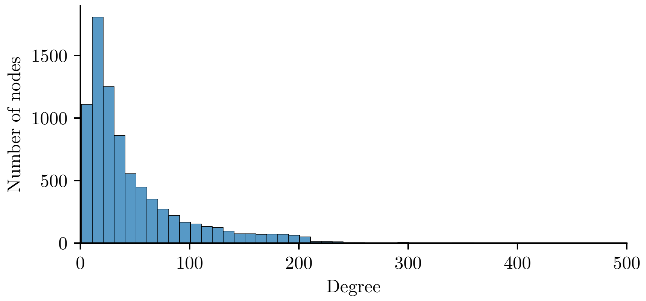

Figure 4.4: Node degree distribution of a social network synthesized with the Facebook graph (Section 4.1.1) by adding 4 random attack edges per Sybil.

4.3 Experiments & Evaluation

4.3.1 Experiments

The following experiments include the evaluation of GNN-based algorithms and can be found in Chapter 6, where the performance of these algorithms is compared to the baseline algorithms.

When the experiment involves a pre-training step, the GNN algorithms are deployed in the *inductive* approach. In some experiments, due to their nature and setup, there is no graph to pre-train on. In these cases, the GNN algorithms are deployed in the *transductive* approach. The approach used in each experiment is clear from context. The distinction between these two approaches was explained in Section 3.4.2 in general and will be explained in Section 5.2.1 specifically for the presented GNN-based Sybil detection.

- **Experiment 1: Evaluation of different Pre-training Strategies** in Section 6.1 analyzes how different pre-training approaches applied to the GNN algorithms influence their performance compared to the baseline algorithms.

- **Experiment 2: Robustness of GNN Algorithms** in Section 6.2 evaluates the robustness of both the GNN and baseline algorithms in different settings, such as different data models, varying graph size, training set size and label noise.
- **Experiment 3 Evaluation on Real-World Twitter Network** in Section 6.3 evaluates the algorithms on the real-world Twitter dataset.
- **Experiment 4: Adversarial Attack Study** in Section 6.4 applies adversarial attacks to the social networks and evaluates the performance of the algorithms under these attacks.
- **Experiment 5: Miscellaneous** in Section 6.5 explores some interesting questions such as what happens when the graph size scales exponentially, or when the honest and Sybil regions are no longer the same size. Additionally, some metrics other than the AUC score are explored and the runtime of the algorithms is analyzed.

In the experiments where social networks are synthesized, the synthetization process is rerun from scratch for each run of the experiment, including the network graph itself as well as the attack edges.

Attack Edges Except for the robustness evaluations in Sections 6.2 and 6.4.1, where the number of attack edges varies within the evaluation, we will set different number of attack edges for the various experiments. The reason for this is, firstly to evaluate various difficulties for the algorithms (to make sure that any observed trend is consistent), and secondly to adjust the "difficulty" of the problem where it is needed. For example, in the adversarial attack evaluation in Section 6.4, the number of attack edges (which are targeted) is reduced compared to the previous experiments to make the problem more tractable, creating a likely more realistic scenario in the presence of adversarial attacks. In general, increasing the connectivity (average degree of the graph, with node degree distribution close to the power law) hand in hand with the number of attack edges makes the problem stagnate in difficulty. This is taken into consideration in the different experiment setups.

For each experiment every scenario was run five times, settings the seeds to $[42, 43, 44, 45, 46]$. The mean and standard deviation values are taken over the five experiment runs. All experiments were performed on an Apple M1 Pro with 16 GB RAM.

4.3.2 Evaluation Metrics

Following the definitions from our modeling of social networks in Section 3.1, we consider Sybil nodes to be *positive* and honest nodes to be *negative* in the sense of our binary classification task. We evaluate exclusively on the test set.[6]

The main evaluation metric used is the *AUC score* (Area Under ROC curve). The reason for this is that the AUC score is a good metric for binary classification tasks, as it is invariant to class imbalance and threshold selection. An AUC score of 0.5 represents a random classifier and 1.0 indicates a perfect classifier.

In addition to the AUC score the metrics *accuracy* (proportion of correctly classified nodes), *precision* (proportion of correctly classified Sybil nodes of all predicted Sybils) and *recall* (proportion of correctly classified Sybil nodes of all actual Sybils) are also computed. Some results of these metrics will be presented and analyzed in Section 6.5.3.

4.3.3 Implementation Details

The code[7] for this thesis was implemented in `Python`[8] [37]. Graph-related implementations were done with `networkx`[9] [38] and `torch-geometric`[10] [39] was used for graph neural networks. The analysis of generated data and scores was performed utilizing `seaborn`[11] [40], which was also used to generate the plots. The implementation for sampling methods, the use of which will be explained in Section 5.2.1, were taken from the `little-ball-of-fur`[12] [41] library.

[6] Some past research [12] has evaluated on the entire set of nodes, likely yielding overoptimistic results. Any comparisons with other algorithms are (re-)evaluated on the test set.

[7] https://github.com/stuartheeb/GNN-sybil-detection.

[8] https://python.org/downloads/release/python-3100/.

[9] https://networkx.org/documentation/stable/release/release_3.1.html.

[10] https://pypi.org/project/torch-geometric/2.5.3/.

[11] https://seaborn.pydata.org/whatsnew/v0.13.2.html.

[12] https://little-ball-of-fur.readthedocs.io/en/latest/index.html, version 2.3.1.

Chapter 5

Training GNNs for Sybil Detection

5.1 GNNs vs. Traditional Graph Algorithms

The main bulk of previous work is made up of traditional[1] graph algorithms [1, 2, 3, 4, 5, 6, 7, 8, 10, 11]. While these algorithms operate directly on the graph by propagating values through the system in some manually defined way, GNNs have the capability to learn the weights of the network.

Why GNNs for Sybil detection? The current algorithms for structure-based Sybil detection on undirected graphs almost exclusively employ some sort of propagation procedure, such as random walks (RW) [1, 2, 3, 4, 6, 7, 11] or loopy belief propagation (LBP) [5, 10, 8, 11]. All these approaches take, in essence, some values and propagate them through the system to arrive at a value for each node, and use this value to determine whether a node is a Sybil or not. This raises the question: why not use a GNN to learn the propagation procedure along with its intricate details? Intuitively, instead of manually designing and optimizing an algorithm that may or may not work well on graphs that it has not been tested on, we could use GNNs in either a transductive or inductive setting to learn from the training set and try to generalize to the test set.

5.1.1 SybilRank as a GNN: A Case Study

Traditional graph algorithms for Sybil detection can be directly "translated" into a GNN setting, by designing the layers of the GNN to propagate (without learning) in the exact same way as the underlying algorithm. We will present this translation for the SybilRank algorithm.

[1] By "traditional" we mean algorithms that do not utilize more sophisticated methods such as machine learning, but operate directly on the graph.

The SybilRank [4] algorithm is an iterative algorithm that propagates initial trust values from a set of known honest nodes through the network, arriving at final trust values that can be used for ranking. The number of iterations is set to $\mathcal{O}(\log n)$. Following [4], let T_G denote the total trust.

The GNN adaptation of the SybilRank algorithm, which we will name SybilRankGNN, is derived directly from [4] and works as follows: The feature (in this case, a scalar) of each node is simply the initial trust value, which is set to $t_u = \frac{T_G}{|H_{\text{train}}|}$ for each node u. The features of all other nodes are set to 0. These node features are then propagated through the GNN, which is defined as follows: the GNN consists of $\lceil \log_2 n \rceil$ `GCNConv` [19] layers, where each layer has a single input channel and output channel. Additionally, the weight matrix is initialized and fixed to **1**, the bias is set to `False` and the aggregation is set to ``sum''. The forward function of the `GCNConv` layer is designed to perform degree-normalization before propagation in each step, as described in [4]. In fact, there is no relevant backward function, as there are no weights to learn (just like SybilRank only propagates values). The final trust values are handled as in [4] to produce a ranked list of nodes which can then be used to predict the Sybils.

SybilRankGNN was tested and unsurprisingly produces equal results to the original SybilRank algorithm.

5.2 Sybil Detection using GNNs

5.2.1 Algorithmic Procedure

Given the social networks graph and a set of training nodes $H_{\text{train}} \cup S_{\text{train}}$, we want to perform Sybil detection using a GNN model. In the following, the procedure for training a model and performing Sybil detection is described. The specific model architectures are introduced in Section 5.3, and losses and optimizers are discussed in Section 5.2.2.

First, the training nodes are split into an actual training set and a validation set, the former of which is used to specify the input features. In each epoch of training, a loss and an optimizer is used to perform backward propagation, where the ground truth is determined with the help of the known nodes. The model is trained for a specified number of epochs, where the validation set is used for early stopping following a patience integer parameter. Early stopping is used to prevent overfitting. If there has been no improvement in terms of validation loss for the last epochs (patience, default is 5), the training process is stopped and the best model (according to validation loss) is retrieved for prediction. The validation set is also used to estimate an optimal threshold (in terms of AUC score), which is then used to predict the Sybils depending on the output of the network. The predictions are then evaluated on the test set (the remaining nodes not used for training), or on a separate graph.

Validation Split As mentioned above, the known nodes are split into a training set and a validation set. For the training phase, the split is set to 80% training and 20% validation. In this case, the validation set is used for early stopping. When the inference phase is performed on a new graph as a separate phase (i.e., inductive rather than transductive learning, as described in the following section), the split it set to 90% training and 10% validation.

Transductive vs. Inductive Learning

In the context of Sybil detection using GNNs, the transductive learning approach describes the process of learning on a graph with a small set of known nodes, and then predicting the labels of the remaining nodes of the same graph. This is done as described in Section 5.2.1.

The concept of inductive learning for Sybil detection using GNNs consists of first training a model on one or multiple graphs and then applying the learned model to one or multiple *unseen* graphs. The former will be called the *training step*, the latter will be called the *inference step*. We assume that the graphs that are learned on are well-labeled or, particularly when they are small, even fully known. For our experiments, we will assume all nodes are labeled in the small training graph. The evaluation graph is a graph that has not been seen during training, and has a small set of known nodes, which are used as input features for the inference step.

Note that the inference step after the process of pre-training can be initiated by a small fine-tuning training step, which allows the model to fine-tune itself to the new graph. In evaluations, we do *not* perform fine-tuning, but directly apply the pre-trained model to the new graph, in order to evaluate the true generalization capabilities of the model without being too optimistic.

The pre-training concept of inductive learning is a tool that can be applied to many scenarios, and we will now go into more details on how exactly it can be used in a realistic setting.

Pre-training

As mentioned above, pre-training a model on one or multiple graphs and then applying this model to unseen graphs is an instance of inductive learning. There are many ways one can use this paradigm in order to perform Sybil detection that are close to real-world scenarios. In the following, we will present some of these ideas.

To predict the node labels of a graph, one could sample a small subgraph, and then train a model on this subgraph. Ideally, this small subgraph would have mostly (or even all) known labels which could be achieved by manual inspection. After pre-training on this graph, the model would be applied to the remaining graph, and the labels of the remaining nodes would be predicted. Typically, the sampled subgraph could be around $5 - 10\%$ of the original graph in terms of size. Note that in our implementation of this methodology, the training nodes are kept as they were before subsampling, and not resampled after the graph sampling process (meaning that the subgraph and the remaining graph will likely not have the same proportion of known honest and Sybil nodes—enforcing this would likely be too optimistic).

Another, more theoretical approach, would be to train a model on a smaller graph and evaluate on a larger graph, where both graphs are generated from the same process, e.g., with the same networks synthetization scheme (see Section 3.1.4). Typically, the smaller graph could be around 10% of the size of the larger graph.

The evaluation of different pre-training strategies and scenarios are presented in Sections 6.1.1, 6.1.2 and 6.4.2.

Sampling Methods

There are different choices of sampling methods that can be used to sample social network graphs. Two prominent ones are the Forest Fire (FF) sampling [42, 43] and the Metropolis-Hastings Random Walks (MHRW) sampling [44]. For this work, we chose the former method, Forest Fire (FF) sampling.

5.2.2 Optimizers and Losses

The choice of optimizer and loss function is in some cases crucial for the performance of the model. In the following, we will present the optimizers and losses that were used in this work.

The optimizer used for our evaluation is the Adam optimizer [45] with learning rate 0.005. While a weight decay parameter was experimented with, we did not achieve superior results.

We use two different loss functions depending on the output width of the GNN model. For output with 1, we use the binary cross-entropy loss[2], and for output with 2, we use the cross-entropy loss[3]. In the first case the output is aimed to represent the probability of the node being a Sybil, and in the second case the two entries in the output vector are supposed to represent the probability of the

[2] https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html.

[3] https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html.

node being Sybil or honest respectively. By default, we use output width 2 for all of our evaluations, since we did not recognize a notable difference in performance between the two options.

Choosing good optimizers and loss functions in a somewhat unexplored area of this work and could be explored as a venue leading to improved results as will be mentioned in Section 7.2.

5.3 GNN Algorithms

In this section three GNN architectures are presented. They make up the different GNN algorithms that are evaluated in this work: SybilGCN, SybilRGCN and SybilGAT.

5.3.1 SybilGCN

The SybilGCN takes the following model parameters: Input width I, hidden width H, output width O, and number of layers L. The model consists of L `GCNConv` [19] layers. The first layer has `in_channels` $= I$, `out_channels` $= H$, the last layer has `in_channels` $= H$, `out_channels` $= O$, and all layers in between have `in_channels` $= H$, `out_channels` $= H$. All layers have `bias` $=$ `False`.

In the forward definition of the model, a tanh[4] activation function is applied after each layer but the last one. After the last layer, a sigmoid[5] activation function is applied if $O = 1$, or a softmax[6] activation function if $O = 2$.

Our default model parameters for the SybilGCN model is $I = 1$, $H = 1$, $O = 2$. We refer to this standard configuration with x layers as SybilGCN-Lx, and we evaluate $L \in \{2, 4, 8\}$.

5.3.2 SybilRGCN

The SybilRGCN model consists of the same model parameters I, H, O, L as the SybilGCN model, but additionally takes an edge type mask, which will be explained below. The model is built as a concatenation of L `RGCNConv` [20] layers. The layer parameters `in_channels`, `out_channels` and `bias` are set in exactly the same way as for the SybilGCN model. The structure of the layers including activation functions is also the same.

[4] https://pytorch.org/docs/stable/generated/torch.nn.functional.tanh.html.

[5] https://pytorch.org/docs/stable/generated/torch.nn.functional.sigmoid.html.

[6] https://pytorch.org/docs/stable/generated/torch.nn.functional.softmax.html.

When considering the different type of nodes in the network, there are nine different edge types for an edge:

$$(u, v) \in \{\underbrace{\text{known honest}}_{H_{\text{train}}}, \underbrace{\text{known Sybil}}_{S_{\text{train}}}, \underbrace{\text{unknown}}_{H_{\text{test}} \cup S_{\text{test}}}\}^2 \tag{5.1}$$

Up to symmetry, this results in six different edge types, where U denotes unknown, namely

$$\{\texttt{(H-H)}, \texttt{(H-S)}, \texttt{(H-U)}, \texttt{(S-S)}, \texttt{(S-U)}, \texttt{(U-U)}\}. \tag{5.2}$$

The `edge_type_mask` parameter determines which edge types are used for the forward propagation of the `RGCNConv` layer. An example of a mask is

$$\{\texttt{(H-H)} : 1, \texttt{(H-S)} : 0, \texttt{(H-U)} : 0, \texttt{(S-S)} : 1, \texttt{(S-U)} : 0, \texttt{(U-U)} : 0\}, \tag{5.3}$$

which is also the default mask that we used. This means that edge types `(H-H)` and `(S-S)` *each* receive separate weights in the learning process, and another set of weights is used for all remaining edge types.

The default parameters for this model are $I = 1$, $H = 2$, $O = 2$, and the `edge_type_mask` shown above. SybilRGCN-Lx refers to such a SybilRGCN model with x layers, where the only variant we evaluate is SybilRGCN-L2, since deeper models were not consistently competitive while having a much higher computational cost.

Note that the edge type mask gives each type a separate set of weights (and gives the remaining edge types another set of weights). It would be possible to group these edge types together in a more sophisticated way and potentially improve the performance. This is a possible area for future work, see Section 7.2.

5.3.3 SybilGAT

SybilGAT consists of the same parameters introduced above: I, H, O and L. Additionally, N determines the number of heads for the attention mechanism. There are L `GATConv` [28] layers. The first layer has `in_channels` $= I$, `out_channels` $= H$ and `heads` $= N$, the last layer has `in_channels` $= H \cdot N$, `out_channels` $= O$ and `heads` $= 1$. The layers in between have `in_channels` $= H \cdot N$, `out_channels` $= H$ and `heads` $= N$. Our experiments showed that adding a dropout before the application with probability `p` $= 0.5$ improved the performance, probably reducing overfitting effects. Therefore, this is the default behavior of the model. We performed evaluations with both the tanh and exponential linear unit (ELU) activation functions, and found tanh to be more effective.

The default parameters for this model are $I = 1$, $H = 4$, $O = 2$ and $N = 4$. An instantiation of the SybilGAT model with x layers will be referred to as SybilGAT-Lx. We evaluate $L \in \{2, 4, 8\}$.

Figure 5.1 shows the loss curves of a SybilGAT model with 4 layers. The model is being pre-trained on a sampled subgraph of size 10% original graph. The original network graph was generated following the network synthetization scheme from Section 3.1.4 using the Barabási-Albert model.

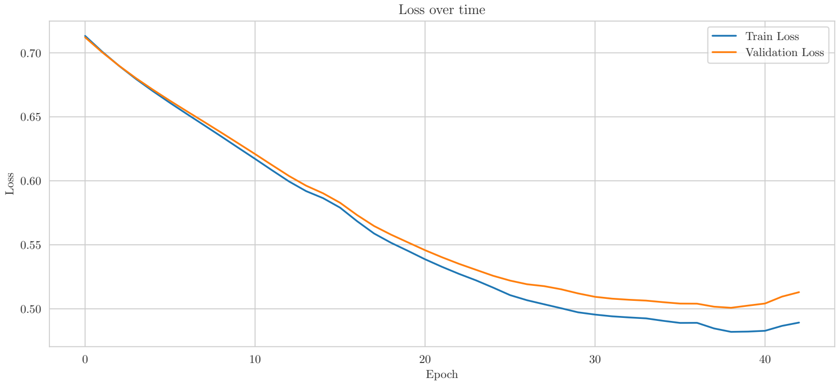

Figure 5.1: Training and validation losses during training process of the SybilGAT-L4 algorithm. The model is being pre-trained on a sampled subgraph of a network generated with the Barabási-Albert model. This is part of the experiment in Section 6.1.1.

In general, the setup of the models including choices such as activation functions and other parameters has a lot of potential for further exploration. This is briefly discussed in Section 7.2.

Chapter 6

Evaluation

6.1 Experiment 1: Pre-training Strategies

As the first experiment, different pre-training strategies are evaluated. In Section 6.1.1, the GNN algorithms are pre-trained on a sampled subgraph, and evaluated on the remaining graph, while being compared to the baseline algorithms. In Section 6.1.2, the pre-training is performed on a smaller social network generated with the same methodology, or pre-trained on a fully synthesized network and applied to a synthetic network with real regions.

An additional pre-training strategy involving training on a with random attack edges and then evaluating on a network with attack edges will be presented in Section 6.4.

6.1.1 Pre-training on Sampled Subgraph

In this part of the experiment, we pre-train on a sampled subgraph and evaluate on the remaining network. The sampling method used is the Forest fire sampling technique mentioned in Section 5.2.1. The sampled subgraph consists of 10% of the nodes of the full social network.

Network with Real Facebook Graph (FB)

We use the synthetization methodology from Section 3.1.4 with the real Facebook graph from Section 4.1.1 to create a synthesized network. To that end, 20 random attack edges per Sybil are added between the regions. The GNN algorithms are pre-trained on a sampled subgraph, and the evaluations and comparisons are performed on the remaining network.

As shown in the left subplot of Figure 6.1 (FB), SybilGCN-L2 clearly outperforms all other algorithms in this experiment, including the baselines. The only one of the SybilGAT algorithms that can get close to this performance is the one with 2 layers, which hints towards a trend that we will continue seeing with the Facebook graph. This occurrence is most likely due to the fact that the Facebook graph is highly connected, having a high average node degree, besides having a node degree distribution similar to the power law distribution. The SybilRGCN algorithm performs rather poorly in this experiment, barely keeping up with the performance of the baselines.

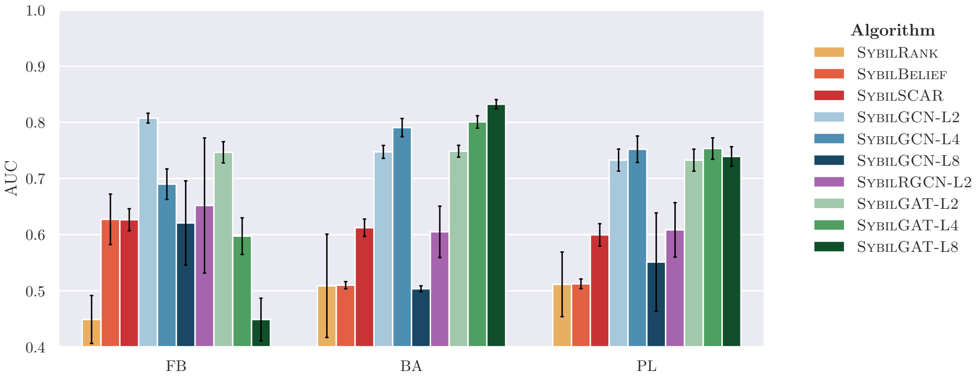

Figure 6.1: Evaluation of AUC score on the remaining network after pre-training on a sampled subgraph, consisting of 10% of the nodes. Networks are constructed using the Facebook (FB) graph, Barabási-Albert (BA) model and Power Law (PL) model as regions, adding random attack edges.

Synthesized Network with Barabási-Albert Model (BA)

Using the same approach mentioned above but now with the Barabási-Albert model from Section 4.2.1 with parameter $m = 10$. The network consists of 5000 nodes, where 15 random attack edges were added per Sybil.

As for the Barabási-Albert (BA) model in Figure 6.1 (middle), the SybilGAT algorithms generally outperform all baselines as well as the other GNN-based algorithms (comparing the respective algorithms with the same number of layers). Once again SybilRGCN performs poorly compared to the other GNN algorithms.

Synthesized Network with Power Law Model (PL)

Like with the Barabási-Albert model, but now using the Power Law model from Section 4.2.1 with parameter $m = 10$ and $p = 0.8$, we evaluate the same network size as above with the same number of random attack edges per Sybil added.

The left side of Figure 6.1 shows the results of the Power Law (PL) model, generally showing the same trends as in the previous experiment with the Barabási-Albert model. However, now the performance of the SybilGCN and Sybil-

GAT algorithms is closely matched (except for the deep layer version). The performance of SybilRGCN remains poor compared to the other GNN-based algorithms.

6.1.2 Pre-training on Smaller Network

In this part of the experiment, we pre-train on a small synthesized social network, and then evaluate on a larger social network, which can be either fully synthesized (following the same data model or a different one) or synthesized with the real Facebook graph.

Pre-training on Barabási-Albert Model Network

After pre-training on a synthesized social network, consisting of the Barabási-Albert model with parameter $m = 6$ with 2'000 nodes and 8 random attack edges per Sybil, we evaluate on three different networks and end up with three variants of this experiment, namely:

- `BA-BA`: A synthesized social network using the Barabási-Albert model with the same parameters and attack edges per Sybil as the pre-train network, but now with 20'000 nodes.
- `BA-PL`: A synthesized social network using the Power Law model with parameters $m = 6$, $p = 0.8$ consisting of 20'000 nodes and 8 random attack edges per Sybil.
- `BA-FB`: A synthesized social network using the Facebook graph, with 8 added random attack edges per Sybil.

Experiment `BA-BA` As can be seen in Figure 6.2, the baseline algorithm SybilRank performs rather well in this experiment, achieving an AUC score of above 0.8, only to be outperformed by SybilGAT-L4 and SybilGAT-L8. The SybilGCN algorithms perform well but cannot keep up with the aforementioned algorithms, but do outperform the other two baseline algorithms.

Experiment `BA-PL` In this experiment the performance of all algorithms is generally lower than in the previous experiment, and discrepancies between algorithms are also smaller, as shown in Figure 6.2. The SybilGAT algorithms still outperform all other algorithms, with the SybilGCN algorithms following closely behind. The SybilRGCN algorithm performs similarly to the baselines algorithms.

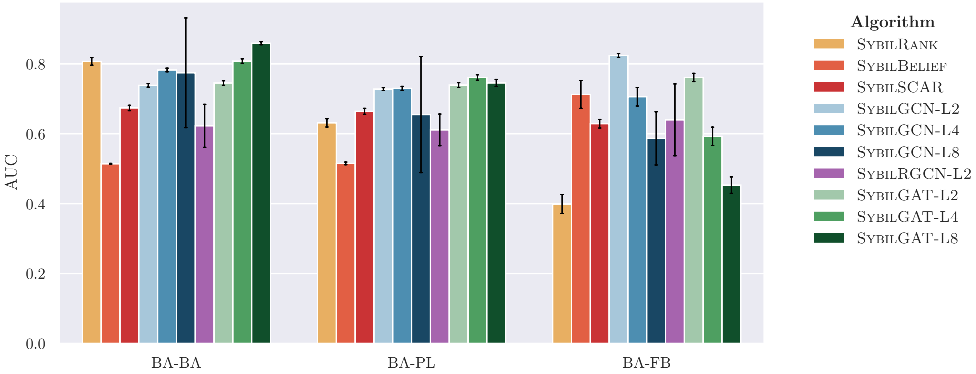

Figure 6.2: Evaluation of AUC score on a large network (FB, BA, PL) after pre-training on a small network constructed using the Barabási-Albert (BA) model, adding 8 random attack edges per Sybil in each case.

Experiment `BA-FB` Continuing the trend of the Facebook graph mentioned in Section 6.1.1, now visible in Figure 6.2, the shallow model SybilGCN-L2 performs the best in this experiment, followed by SybilGAT-L2, and then by the baseline SybilBelief.

Pre-training on Power Law Model Network

We perform the same set of three experiments as above, but now pre-training on a synthesized network using the Power Law graph with parameters $m = 6$, $p = 0.8$ and 2'000 nodes and 8 added random attack edges per Sybil. The evaluation networks are defined exactly the same as above, and the experiments are likewise named `PL-BA`, `PL-PL` and `PL-FB` respectively.

Note that the baseline algorithms for the following three parts of this algorithm will perform the same as in the three parts of the previous experiment, since they are evaluated on the same network setups (they do not perform any pre-training).

Experiment `PL-BA` Figure 6.3 shows a similar picture for this experiment as for the corresponding part of the one above: SybilRank performs well, only to be outranked by SybilGAT-L4. Now, however, SybilGCN-L8 and SybilGAT-L8 have a very high standard deviation, which previously only happened to the SybilGCN-L8 algorithm. This might be due to the fact that their architecture is too complex for the underlying graph, or some other case of overfitting.

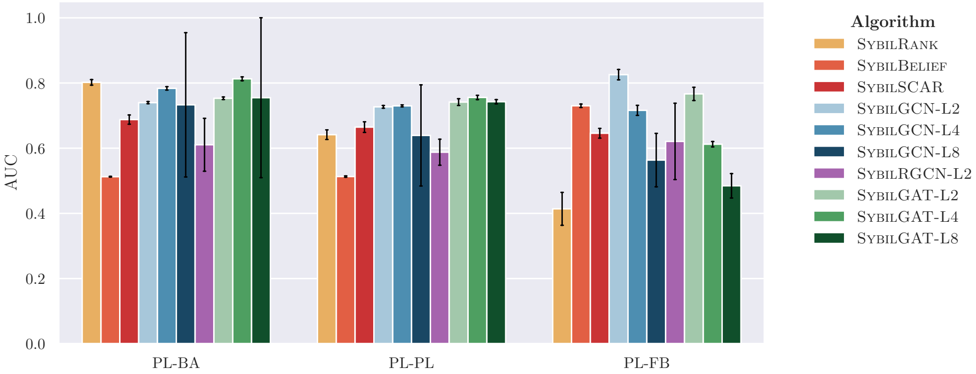

Figure 6.3: Evaluation of AUC score on a large network (FB, BA, PL) after pre-training on a small network constructed using the Power Law (PL) model, adding 8 random attack edges per Sybil in each case.

Experiment `PL-PL` Figure 6.3 presents an interesting discovery for the GNN-based algorithms: pre-training on a Power Law graph and then evaluating on a larger Power Law graph does not perform better than pre-training on the same graph model and then evaluating on a different graph model, namely the Barabási-Albert model. In fact, the scores of the middle subplot are not better than the ones on the left. The trend, however, still exists: the SybilGAT algorithms outperform all other algorithms.

Experiment `PL-FB` As can be seen in Figure 6.3, this experiment shows the same trends as the experiment `BA-FB` in the previous section: The SybilGCN algorithms (especially the shallow version) perform the best, followed by the SybilGAT algorithms, and then the baseline SybilBelief algorithm.

6.2 Experiment 2: Robustness Analysis

In this experiment, we test the robustness of each algorithm, including the baselines, individually. Additionally, we compare the robustness of the algorithms compared to each other. This evaluation is done for four categories of robustness: data models, graph size, training set sizes and training label noise.

In each of the subexperiments, we examine the performance with varying number of (random) attack edges, signifying an increase in how hard the problem is. The number of attack edges per Sybil evaluated range from 2 to 12. Therefore, the robustness in number of attack edges is always implicitly evaluated.

The values for each category mentioned above, including their default value, are as follows:

- Data model: Facebook graph, Barabási-Albert model ($m = 6$) and Power Law model ($m = 6$, $p = 0.8$). Default: Power Law model ($m = 6$, $p = 0.8$).
- Graph size: 1000, 2000, 4000, 8000, 16'000 nodes. Default: 2000 nodes.
- Train set sizes: 0.5%, 1%, 5%, 10%, 20%. Default: 5%.
- Label noise: 0%, 10%, 20%, 30%. Default: 0%.

For the evaluation of each category (treated as variable), all the other values are kept at the default value.

The results for each experiment will be presented by first presenting the effect of varying the parameter in question has on each algorithm individually, and then compare the algorithms to each other in each setting of the parameter. This will give useful insight in to how the parameter influences the problem complexity and then how well the algorithms handle this change in difficulty.

Note that we have removed the baselines SybilRank and SybilBelief as well as the SybilGCN-L8 and SybilGAT-L8 algorithms from this experiment to make the plots more readable. The reason for this is that SybilRank and SybilBelief perform very poorly with increasing number of attack edges and are no longer competitive, and that the deep layer versions of GCN and GAT algorithms do not perform very reliably in these experiments. The complete experiment data can be found in Appendix A.1.

6.2.1 Data Models

We compare the two different synthetic data models from Section 4.2.1 which are used in the network synthetization scheme (Section 3.1.4): The Barabási-Albert (BA) model (Section 4.2.1) and the Power Law (PL) model (Section 4.2.1). The total network size is kept at 2000 nodes, the training set size at 5% of all nodes, and the label noise at 0%.

Figure 6.4 shows no major difference between the Barabási-Albert (BA) model and the Power Law (PL) model. It indicates that there *might* be a slight advantage in the sense of better mean AUC score for the Barabási-Albert model. It seems that this potential difference is only clearly visible when the GNN model has 4 layers, which indicates that it might have something to do with the connectivity within the regions.

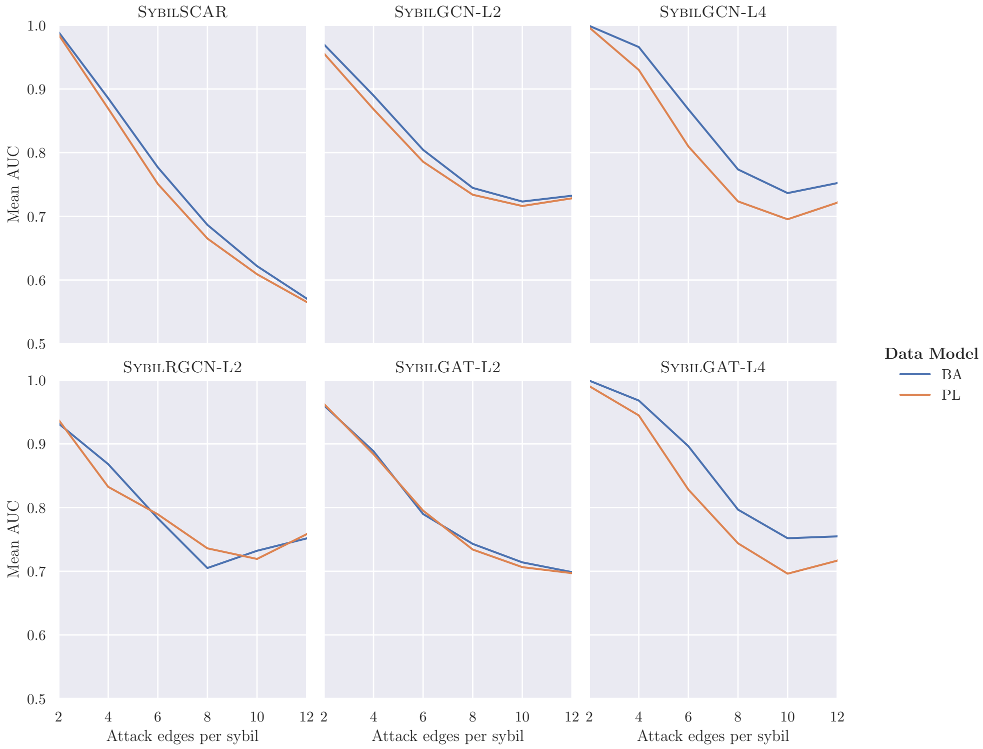

Figure 6.4: Evaluation of AUC score on a network constructed with either the Barabási-Albert (BA) or Power Law (PL) model with 2000 nodes, with varying number of random attack edges per Sybil. Comparison of different data models for each algorithm.

Due to the Barabási-Albert model potentially being easier to solve within our problem framework, we chose the Power Law model when we fix the model while varying other parameters, in order to be conservative in our evaluation and conclusions.

As can be seen in Figure 6.5, the GNN-based algorithms outperform the SybilSCAR algorithm significantly in both data models, even more so when the attack edges per Sybil increase.

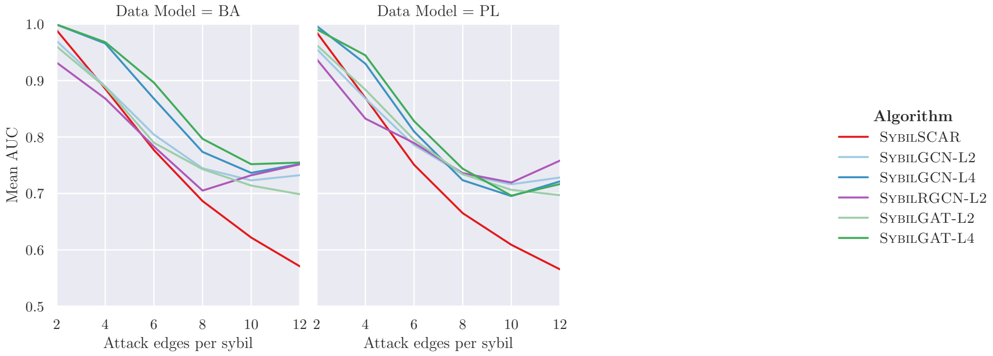

Figure 6.5: Evaluation of AUC score on a network constructed with either the Barabási-Albert (BA) or Power Law (PL) model with 2000 nodes, with varying number of random attack edges per Sybil. Comparison of different algorithms for each data model.

6.2.2 Graph Size

In this experiment the performance of the algorithms is tested with varying total graph size. The evaluated sizes are 1000, 2000, 4000, 8000 and 16’000 nodes. The Power Law model is used while the training set size is kept at 5% of all nodes and the label noise at 0%.

According to Figure 6.6 there seems to be no difference in performance in any of the shown algorithm when varying the graph size exponentially from 1000 to 16’000. This is most likely due to the fact that the number of attack edges is set as number of attack edges *per Sybil*, meaning that it scales with the graph size. However, the larger the regions get, the more edges are added within the regions too, although this does not seem to influence the algorithms’ performance.

Figure 6.7 paints a similar picture as Figure 6.5: SybilSCAR is outperformed by all other examined algorithms across all graph sizes. This is no surprise as we just established that the graph size does not seem to influence the performance.

6.2.3 Training Set Sizes

We compare different sizes of training sets to see how many nodes (in % of all nodes) the algorithms need to achieve good performance. The evaluated percentages are 0.5%, 1%, 5%, 10% and 20% of all nodes. The Power Law model is used while the total graph size is kept at 2000 nodes and the label noise at 0%.

As one would expect the problem gets easier to solve when more training data is available, which is confirmed by Figure 6.8. It also shows another interesting occurrence: the SybilSCAR seems to suffer less from lack of training data *in a*

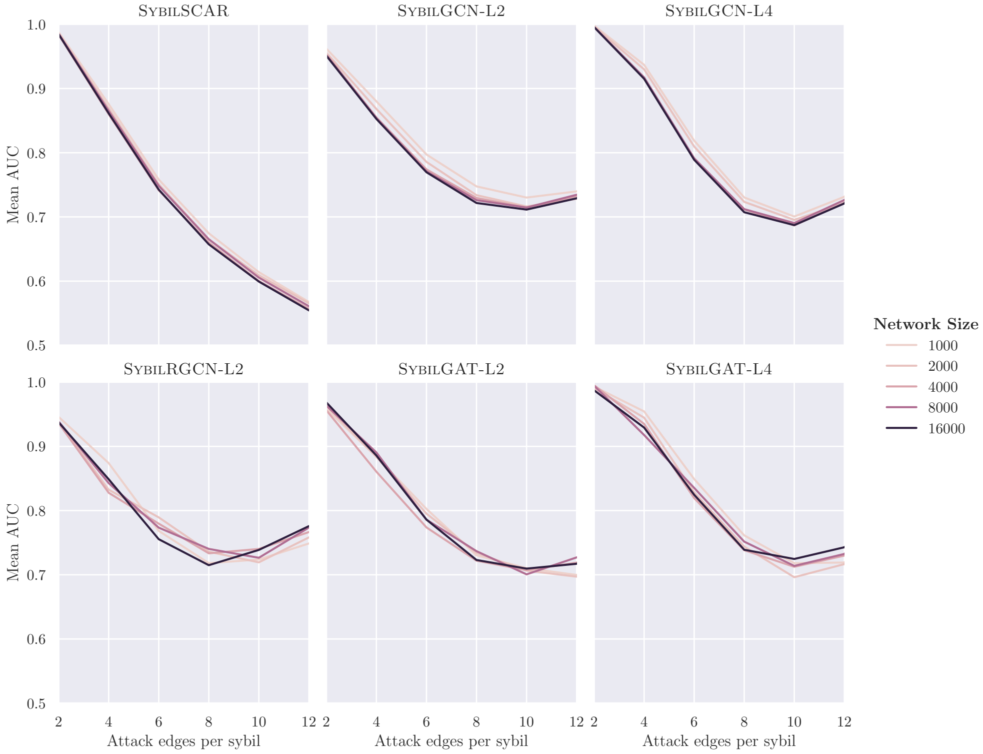

Figure 6.6: Evaluation of AUC score on a network constructed with the Power Law (PL) model, with varying number of random attack edges per Sybil. Comparison of different network sizes (1000, 2000, 4000, 8000, 16'000) for each algorithm.

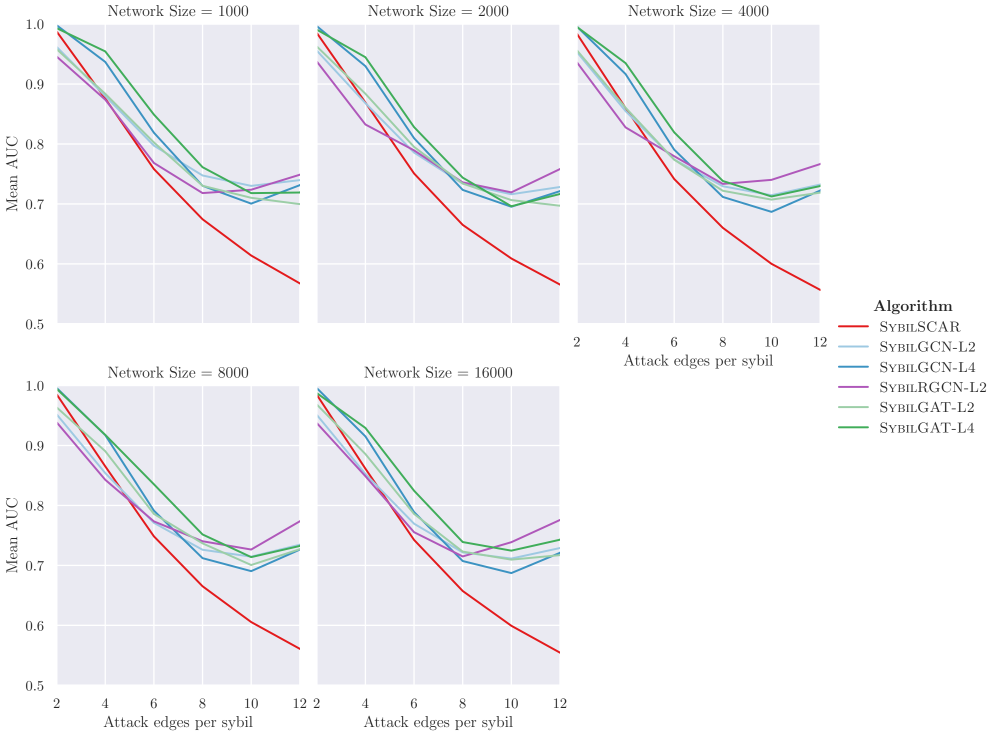

Figure 6.7: Evaluation of AUC score on a network constructed with the Power Law (PL) model, with varying number of random attack edges per Sybil. Comparison of different algorithms for each network size (1000, 2000, 4000, 8000, 16'000).

relative sense. It manages to almost close the gap in performance to the GNN algorithm when training data is sparsely available, such as the cases where the training set is set to 0.5% and 1% of the total set of nodes.

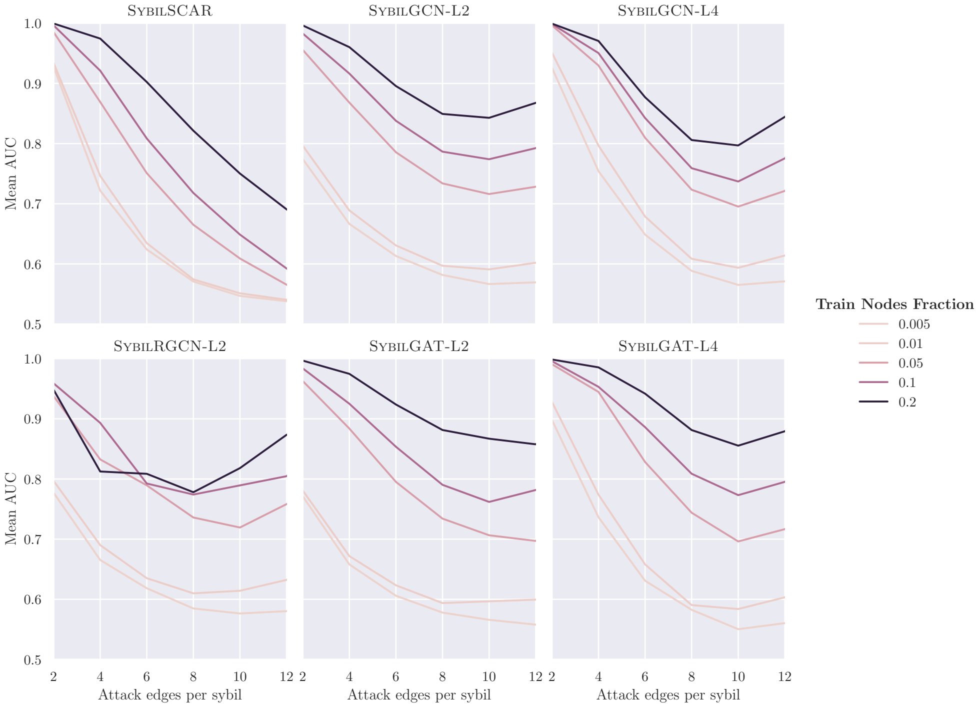

Figure 6.8: Evaluation of AUC score on a network constructed with the Power Law (PL) model with 2000 nodes, with varying number of random attack edges per Sybil. Comparison of different training nodes set fractions (0.5%, 1%, 5%, 10%, 20%) for each algorithm.

As hinted above and shown in Figure 6.9, when the amount of known nodes is very small, SybilSCAR manages to perform similarly well to the GNN-based algorithms. But as soon as there is 1% or more known nodes, the GNN-based algorithms outperform the SybilSCAR algorithm clearly with increasing attack edges per Sybil, just as in the previous experiments.

6.2.4 Training Label Noise

In this experiment, we perturb the training labels, via adding noise. Given a percent of label noise, the labels that proportion of nodes (for both honest and Sybil nodes) is flipped (from Sybil to honest and vice versa). The evaluated label noise percentages are 0%, 10%, 20% and 30%. Throughout this experiment, the Power Law model is used, the total graph size is 2000 nodes and the training set

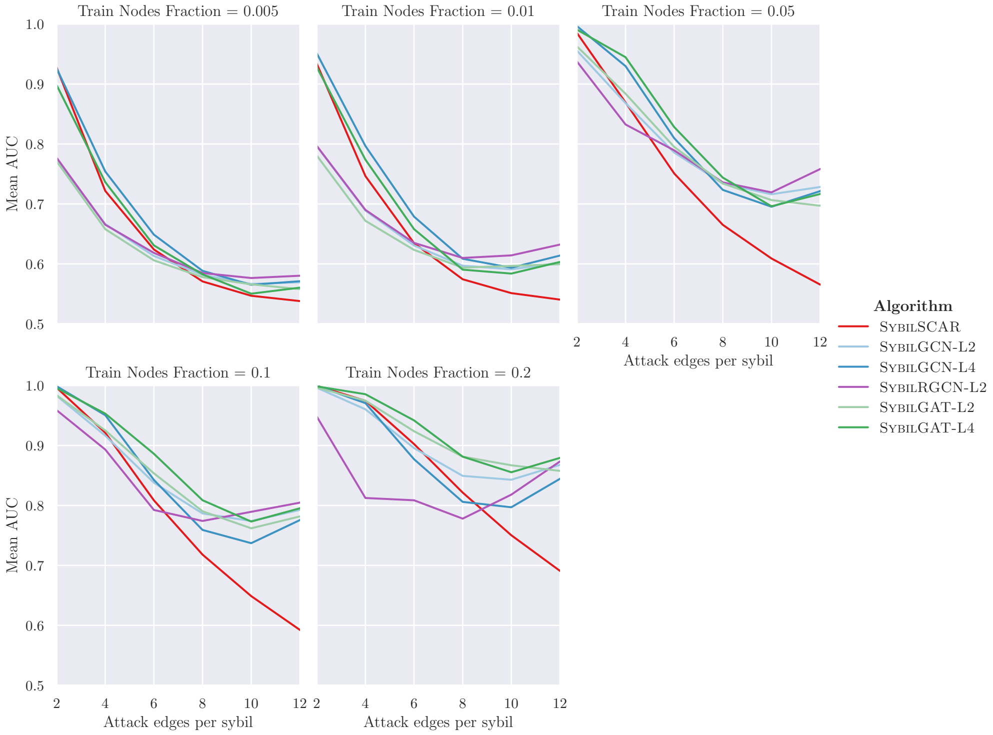

Figure 6.9: Evaluation of AUC score on a network constructed with the Power Law (PL) model with 2000 nodes, with varying number of random attack edges per Sybil. Comparison of different algorithms for each training nodes set fraction (0.5%, 1%, 5%, 10%, 20%).

size is 5% of all nodes. Evaluation is still done on the test nodes only, with the real, unperturbed labels.

Figure 6.10 shows that, as one might expect, label noise in the training set has a negative effect on the performance of the algorithms. This happens across the board to all algorithms, however it seems that SybilSCAR is *relatively* less affected by the label noise (meaning the gaps between the lines of Figure 6.10 are smaller for SybilSCAR than for the other algorithms).

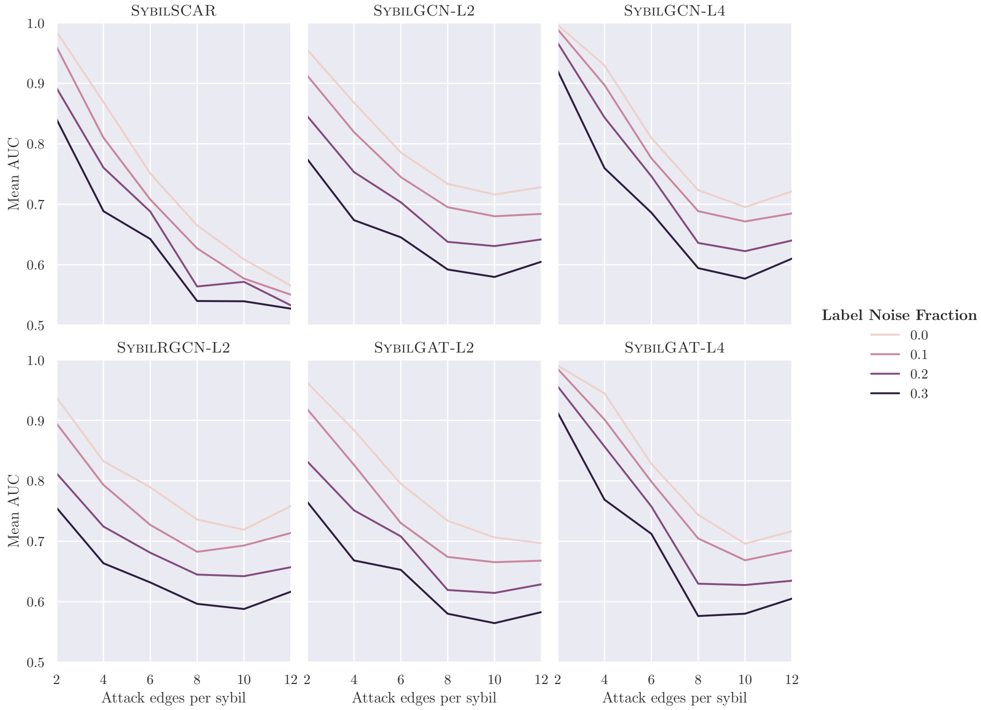

Figure 6.10: Evaluation of AUC score on a network constructed with the Power Law (PL) model with 2000 nodes, with varying number of random attack edges per Sybil. Comparison of different label noise levels (0%, 10%, 20%, 30%) for each algorithm.

Comparing the algorithms with each other, as done in Figure 6.11, however, shows a similar scenario as in the previous experiments: the GNN-based algorithms outperform the baseline in all cases, increasingly so with more attack edges per Sybil. This is less visible with a high label noise of 30%, but the trend remains.

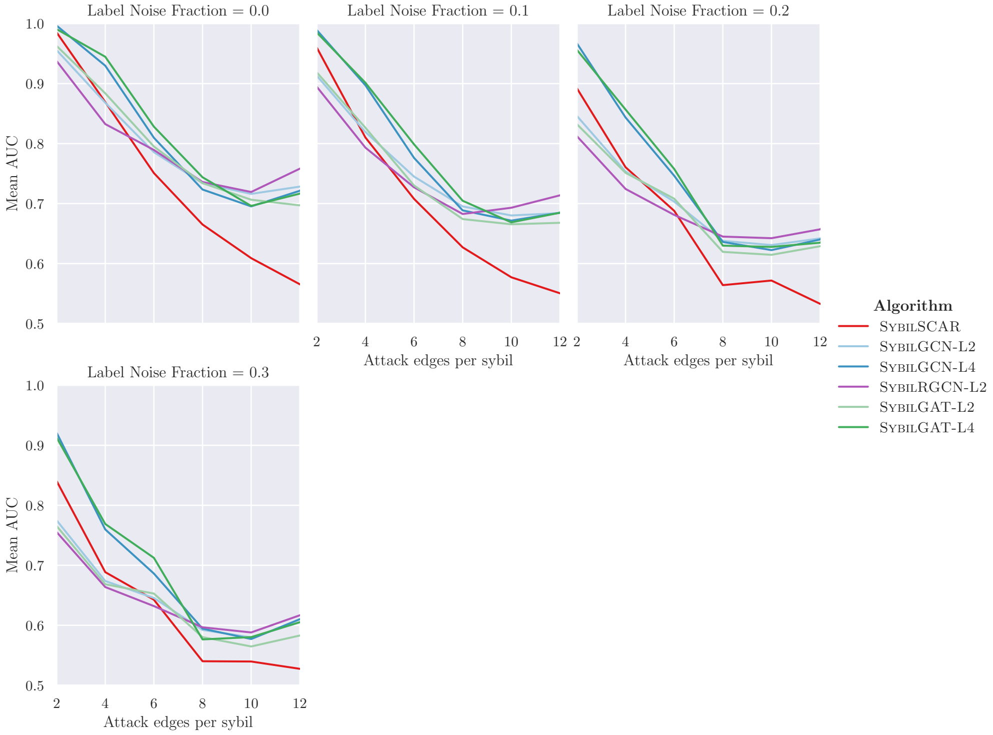

Figure 6.11: Evaluation of AUC score on a network constructed with the Power Law (PL) model with 2000 nodes, with varying number of random attack edges per Sybil. Comparison of different algorithms for each label noise level (0%, 10%, 20%, 30%).

6.3 Experiment 3: Real-World Twitter Dataset

In this experiment we evaluate the algorithm on the real-world labeled Twitter graph from Section 4.1.1, which was transformed to an undirected graph by keeping all edges that exist in either direction. We want to establish if it is feasible for the algorithms to pre-train on a small subgraph, and then reliably evaluate on the remaining subgraph, as one might do in a real-world application where a small part of the graph has already been inspected. To that end, we pre-train on a sampled subgraph of the network which is 5% of the original size. The evaluation is then performed on the remaining network. The sampling method is the Forest fire sampling technique from Section 5.2.1. As mentioned in Section 4.1.1, the training set of this dataset consists of ≈ 10% of all nodes [12].

As shown in Figure 6.12, SybilGAT-L4 outperforms all other algorithms, including the baselines. The SybilGAT-L2 algorithm performs slightly worse, followed by the baseline SybilSCAR and the corresponding two SybilGCN algorithms. The deep GCN and GAT models do not perform as well as their shallow counterparts, which is most likely due to structural properties of the underlying graph. The SybilRGCN algorithm performs poorly compared to the other GNN-based algorithms.

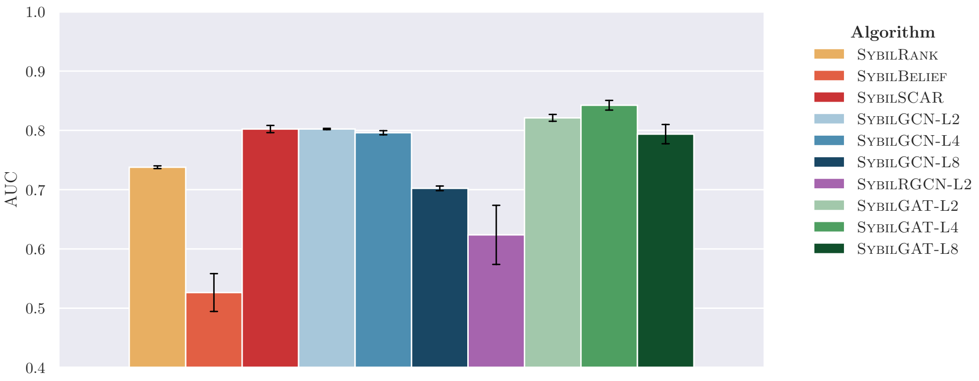

Figure 6.12: Evaluation of AUC score on Twitter dataset, after pre-training on a sampled subgraph 5% of the size of the original network. Evaluation performed on the remaining network.

6.4 Experiment 4: Adversarial Attack Study

6.4.1 Robustness: Targeted Attack Edges

In this experiment, similarly to Section 6.2, we evaluate[1] over a varying number of attack edges per Sybil, but now with targeted attack edges. Once again, the number of attack edges per Sybil evaluated vary from 2 to 12. We use the Power Law model with parameters $m = 10$ and $p = 0.8$ with total graph size 2000 nodes. We evaluate four instantiations of our model for a targeted attack presented in Section 3.5.2:

- $p_T = 0.05$ and $\boldsymbol{p} = [0.25, 0.25, 0.5]$,
- $p_T = 0.10$ and $\boldsymbol{p} = [0.25, 0.25, 0.5]$,
- $p_T = 0.15$ and $\boldsymbol{p} = [0.25, 0.25, 0.5]$ and
- $p_T = 0.20$ and $\boldsymbol{p} = [0.25, 0.25, 0.5]$.

Note that the p_T parameter varies while the target hit distance probability density function $\boldsymbol{p}$ is kept constant at $[0.25, 0.25, 0.5]$. As mentioned in Section 3.5.2, this instantiation of p means that for a targeted attack edge, the probability of it being placed at a *known* (train) honest node with distance 0, 1, and 2 is 0.25, 0.25, and 0.5, respectively.

Figure 6.13 shows that, as expected, for each attack in the order presented above, the AUC scores steadily decrease, as this represents an increase in difficulty. The reason for this is that the p_T value increases, effectively increasing the probability for any attack edge to be *targeted* rather than *random*, which is synonymous with an increase in expected number of targeted attack edges.

Figure 6.14 shows a comparison of the different algorithms for each targeted attack tested. The main observed trends are that the SybilGCN, SybilRGCN and SybilGAT algorithms outperform the baselines clearly in all attacks, especially when the number of attack edges per Sybil grows. The higher the p_T parameter of the tested attack, the more pronounced the strength of the SybilRGCN-L2 algorithm becomes. Due to the nature of this algorithm and how it uses specific edge types to its advantage, and with there being an increasing number of such specified "attack edge types" (connections between known nodes), it even starts to improve its AUC score when the attack becomes harder. From the plot it is also clear that the SybilGCN algorithms are not suitable for a scenario with strong adversarial influences. Additionally, the plots imply

[1] The results for the deep versions of the GCN and GAT algorithms (SybilGCN-L8 and SybilGAT-L8) were omitted from this experiment due to the fact that they performed very irregular and had an obfuscating effect on the presentation effort of the plots. The complete experiment data can be found in Appendix A.1.

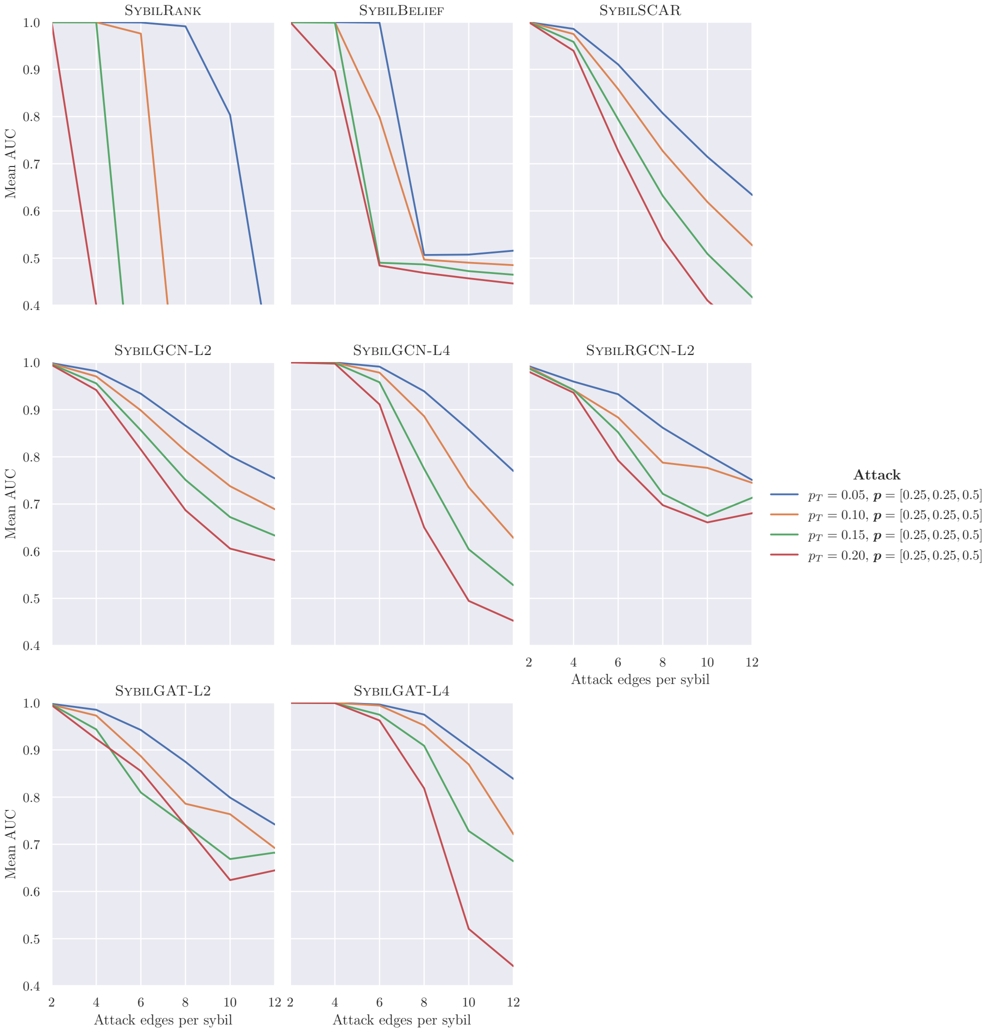

Figure 6.13: Evaluation of AUC score on a network constructed with the Power Law (PL) model with 2000 nodes, with varying number of *targeted* attack edges per Sybil. Comparison of different attacks for each algorithm.

that with increasing attack complexity, a smaller number of layers (effectively the propagation distance of the labels) in the GNN architecture is beneficial. This is an avenue to be further explored in future work.

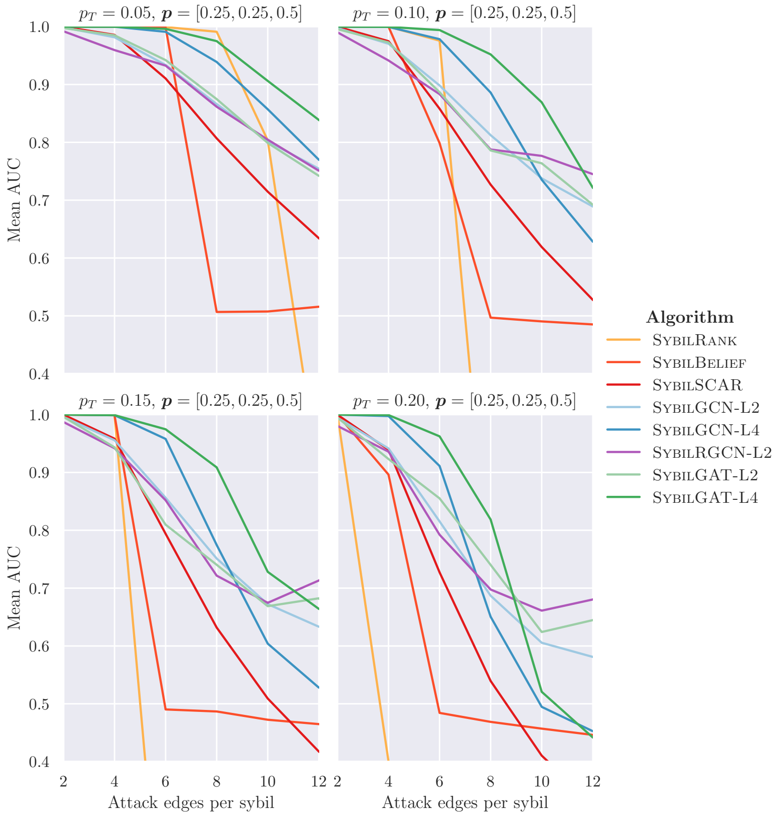

Figure 6.14: Evaluation of AUC score on a network constructed with the Power Law (PL) model with 2000 nodes, with varying number of *targeted* attack edges per Sybil. Comparison of different algorithms for each attack.

6.4.2 Pre-training Before Targeted Attack

We evaluate the capability of the GNN-based algorithms to perform well on a network with targeted attack edges, after being pre-trained on the same network which only contains random attack edges. We evaluate both the Barabási-Albert model ($m = 10$) and Power Law model ($m = 10$, $p = 0.8$), with total network size 2000 nodes.

The pre-training network is constructed with 10 random attack edges per Sybil, and the evaluation network consists of the same regions, but attacked following a targeted attack with attack parameters $p_T = 0.05$ and $\boldsymbol{p} = [0.25, 0.25, 0.5]$

placing 10 attack edges from scratch.

Figure 6.15 shows the result of this experiment for both the Barabási-Albert and Power Law models. The results of the evaluation on these two data models show the same trends and do not offer any meaningful insight into differences of the models. The plot shows that the SybilGCN (except the deep version) and the SybilGAT algorithms are superior to all other algorithms, including the baselines. The SybilRGCN algorithm performs poorly across the board, which could potentially be explained by the fact that being after trained on a randomly attacked network and then directly applied to one with targeted attack edges, it never had the chance to make use of its ability to differentiate and learn the edge types that predominantly appear in targeted attacks. Note that the SybilRank evaluation results in a very high standard deviation, indicating that it is not equipped to handle adversarial attacks in a robust manner. Each version of the SybilGAT algorithm outperform its counterpart of SybilGCN with the same number of layers. This concludes that the SybilGAT algorithms, when pre-trained on a randomly attacked network can adapt to a targeted attack scenario and still perform well, without the need to fine-tune on the network with targeted attack edges.

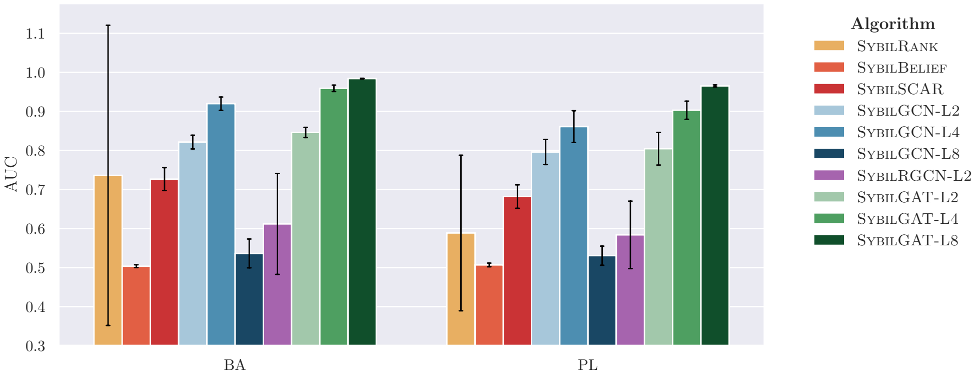

Figure 6.15: Evaluation of AUC score on a network constructed with the Power Law (PL) model with 2000 nodes, which was attacked with 10 targeted attack edges per Sybil, after being pre-trained on a the same network constructed with 10 *random* attack edges per Sybil.

6.5 Experiment 5: Miscellaneous

6.5.1 Scaling to Very Large Networks

Here we evaluate the capability of the GNN algorithms to scale to very large networks. To this end, we use the Power Law model ($m = 6$, $p = 0.8$) to construct a social network with 8 random attack edges per Sybil and 2000 nodes in total. This is the pre-training network. We then evaluate the algorithms on a network with the same parameters, but with a much larger amount of nodes, namely, 8000, 32'000 and 128'000 nodes, taking an evaluation on a network with 2000 nodes as a type of baseline.

The plots in Figure 6.16 indicate that even with drastically increasing network size (exponentially), the performance trends remain unchanged. This is the case for the traditional baseline algorithms as well as the GNN-based algorithm that were pre-trained on a small network of 2000 nodes. Additionally, the GNN-based algorithm seem to adapt better to very large network sizes, as their performance degrades less (in a relative sense) than the performance of the baseline algorithms, confirming the results from Section 6.2.2.

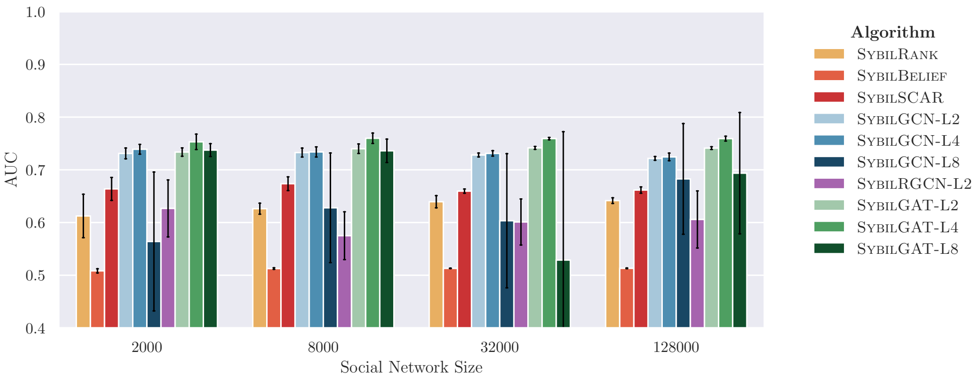

Figure 6.16: Evaluation of AUC score on networks of exponentially growing size, after pre-training on a small network of size 2000 nodes. The networks were constructed using the Power Law (PL) model.

6.5.2 Different Region Sizes

To evaluate what happens when the regions are not the same size, we construct asymmetric social networks using the Power Law model ($m = 6$, $p = 0.8$). We evaluate eight different scenarios, namely when either the honest or Sybil region has a base size of 1000 nodes, while the other region has a size that is 2×, 4×, 6× and 8× larger. The number of attack edges is

$$8 \times (\text{total network size}/2), \tag{6.1}$$

where the total network size is

$$(1 + \text{multiplier}) \times 1000, \tag{6.2}$$

depending on the multiplier used. The reason to not use a number of attack edges *per Sybil* is because that would be make the problem infeasible when the number of Sybils is much larger than the number of honest nodes.

Note that for these experiments (and all others), the training nodes are sampled proportionally to their respective region sizes. In this case, this will lead to the training set containing more honest nodes if the honest region is larger, and more Sybil nodes if the Sybil region is larger.

More Sybil Nodes

Constructing and evaluating a network consisting of 1000 honest nodes and a varying number of Sybil nodes, Figure 6.17 shows a trend emerging: the more imbalanced the network is, the worse the performance of the algorithms. This hold across the board for all algorithms. Comparing the performance of the algorithms shows that SybilRGCN excels, ranking above all other algorithms. While in the case where the network is slightly imbalanced the SybilGAT algorithms can keep up with this, when the imbalance is larger, that is no longer the case. It is clear from the plot that when this imbalance increases, all algorithms except SybilRGCN become useless as their AUC score drops to around 0.5.

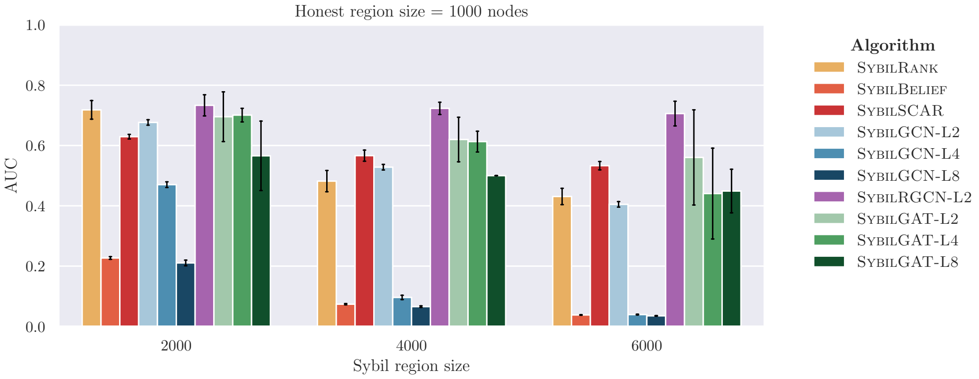

Figure 6.17: Evaluation of AUC score on a network that was constructed with the Power Law (PL) model with 1000 nodes in the honest region, and varying number of nodes in the Sybil region.

More Honest Nodes

Reversing the setup of the experiment by constructing a network with 1000 Sybil nodes and a varying number of honest nodes, the results, as Figure 6.18 shows, are surprisingly similar. The main trends are the same, with the SybilRGCN algorithm outperforming all other algorithms, but now with the SybilGAT being able to compete more closely as the imbalance increases. Once again, as the imbalance becomes more pronounced, most algorithms become useless, with their AUC score hovering around 0.5, which is what can be achieved by random guessing.

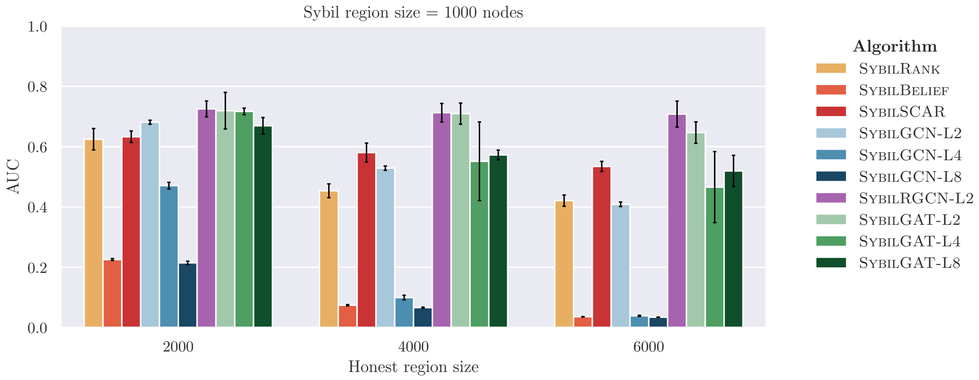

Figure 6.18: Evaluation of AUC score on a network that was constructed with the Power Law (PL) model with 1000 nodes in the Sybil region, and varying number of nodes in the honest region.

We see that most algorithms, both baseline and GNN-based, have great difficulty in dealing with networks that are imbalanced in terms of the number of honest and Sybil nodes. The SybilRGCN algorithm is the only one that can handle this imbalance to some extent. This is a clear indication that most of the algorithms are not robust to imbalanced networks, and that this is an area that needs further research.

6.5.3 Other Metrics

To evaluate the general performance of all algorithms with respect to other metrics than the AUC score, we present the results of the accuracy, precision and recall for the experiment in Section 6.1.1.

As a quick reminder, the experiment in Section 6.1.1 pre-trains the GNN algorithms on a subgraph of the network and then evaluates on the remaining network. The subgraph consists of 10% of the nodes of the full social network. The evaluation is performed on the test nodes only. The results of this experiment can be summarized as follows: the evaluation was dominated by the SybilGCN (except the L8 version) and the SybilGAT in terms of AUC score.

While we consider the AUC score to be the most significant metric, we include this short analysis for completeness and to give a more complete picture of the performance of the algorithms, as these metrics can be of interest in different scenarios, depending on the requirements and potential cost of making a wrong prediction in either direction.

Accuracy As shown in Figure 6.19, the trend observed in the experiment in terms of AUC score continues to hold when looking at accuracy. A slight exception to this is the first part of the experiment, the evaluation on the network synthesized with the Facebook graph.

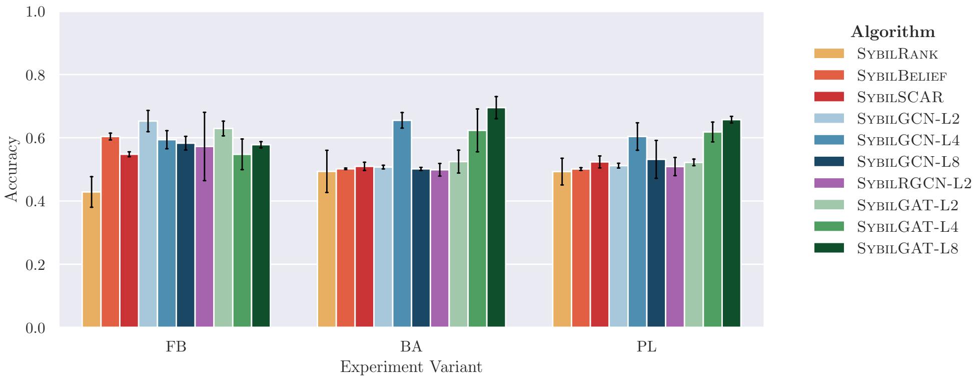

Figure 6.19: Accuracy scores of experiments from Section 6.1.1.

Generally, the accuracy scores are lower than one might except after seeing the AUC scores of this experiment. This is most likely due to the way thresholding is done by the GNN-based algorithms. The problem of setting an appropriate threshold for prediction after the last layer of the network is important and is an area that needs further inspection (see Section 7.2).

Precision The evaluation of the precision statistic of this experiment shows in essence the same trend as the original experiment, as can be seen in Figure 6.20. Precision is particularly important when the cost of false positives is high, which one might deem a sensible requirement for a Sybil detection system.

Recall Figure 6.21 shows that the recall scores of many of the algorithms are very low, especially in the two fully synthesized networks (BA and PL). This is the case even for the algorithms that clearly outperform in terms of AUC score. In a sense, this can be explained in the fact that in many applications a trade-off

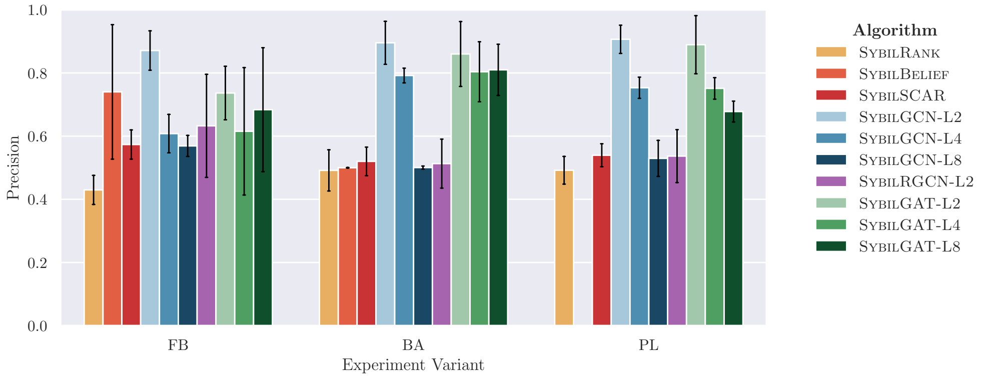

Figure 6.20: Precision scores of experiments from Section 6.1.1.

needs to be made between recall and precision, and in the case of the analyzed algorithms, an implicit (or explicit) priority is on precision. Recall is important when the cost of false negatives is high.

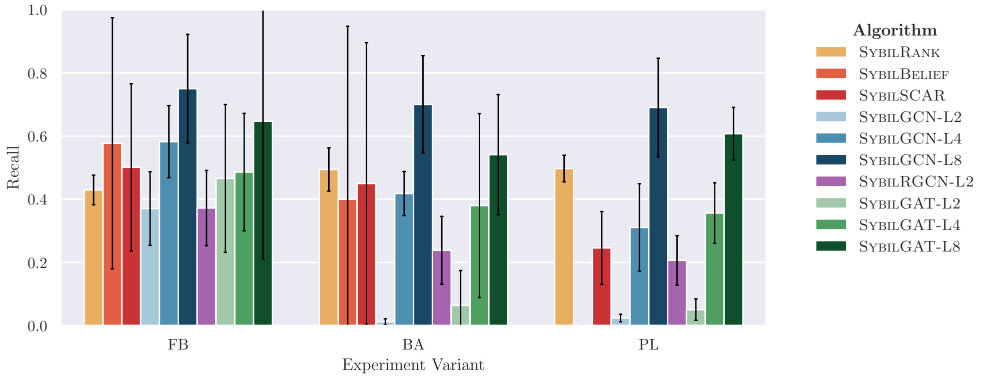

Figure 6.21: Recall scores of experiments from Section 6.1.1.

6.5.4 Runtime Analysis

To get an idea of how the runtimes of the GNN algorithms compare to the baseline algorithms, we will compare the evaluation runtime of the algorithms in several experiments where pre-training was done beforehand. The pre-training runtime is of course much higher than not only the evaluation runtime, but also the runtime of the baseline algorithms.

Considering the thought behind pre-training is to do it once (or when something about the network structure changes or new data becomes available) and then apply the pre-trained model to multiple graphs, we compare the evaluation runtime, taking the pre-train runtime as a one (or few) time overhead cost. This must be noted when interpreting the results, as when including pre-training time, the GNN-based algorithms will have a much higher runtime and make the comparison obsolete.

Runtime after Subgraph Pre-training

Evaluating the runtime of the experiment in Section 6.1.1, which pre-trains on a subsampled graph consisting of 10% of the original network, we see in Figure 6.22 that most GNN-based algorithms have a significantly lower runtime than the baseline algorithms. This, combined with the fact that the SybilGCN and SybilGAT algorithms perform better in terms of AUC score, shows clear superiority of these GNN-based algorithms. Exceptions to this are the SybilRGCN and SybilGAT-L8 algorithms, with the former having a significantly higher runtime than all the other algorithms and the latter having a similar runtime as SybilRank and SybilSCAR. The inferiority of SybilRGCN in terms of runtime can be explained by the fact that it intrinsically has a higher complexity due to having to handle different edge types (which not only must be detected, but also necessitate the need for separate weight matrices).

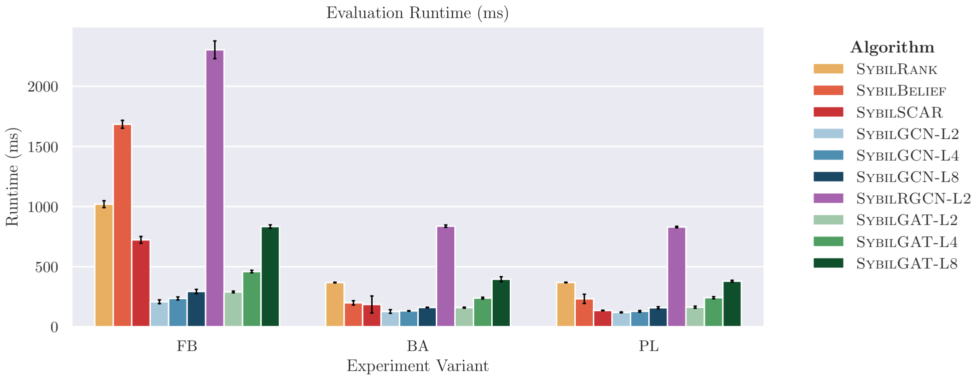

Figure 6.22: Runtimes of experiments from Section 6.1.1.

Runtime of Twitter Dataset

Looking at the runtimes of the Twitter dataset evaluation in Section 6.3 (which is also based on the pre-training on subgraph approach), shows a similar trend as mentioned previously, except that now SybilRGCN runs faster in relation to before. This can be seen in Figure 6.23. Again, the algorithms that outperform the baselines in terms of AUC score also have a lower runtime in the evaluation step.

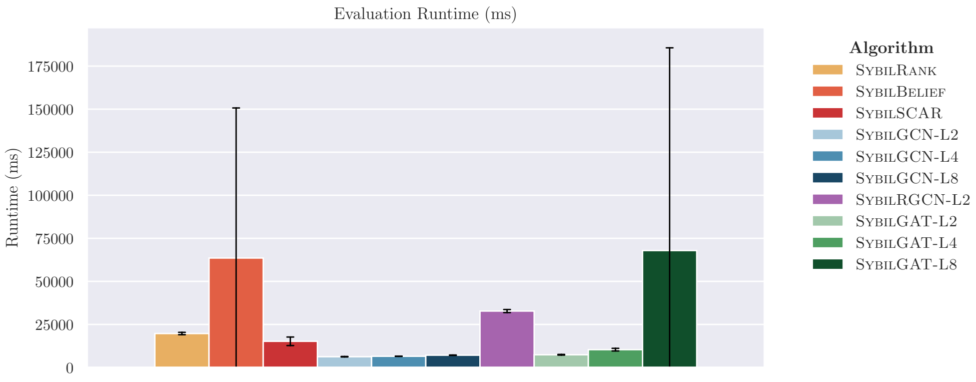

Figure 6.23: Runtimes of experiment from Section 6.3.

Runtime of Very Large Networks

We will now look at the runtime of the experiment in Section 6.5.1, in which the GNN algorithms are pre-trained on a very small graph of the same synthetization technique, and then evaluated on an exponentially larger network alongside the baseline algorithms. The results of this experiment can be seen in Figure 6.24. As expected the runtime scales with the size of the network (note the logarithmic scale on the y-axis). The SybilGCN and SybilGAT not only outperform the baseline algorithms in terms of AUC score as presented in Section 6.5.1, they also have a lower runtime. The runtime of the SybilRGCN is still poor. An interesting observation is that the runtime of SybilGAT-L4 suddenly increases much more than exponentially in the largest network evaluation.

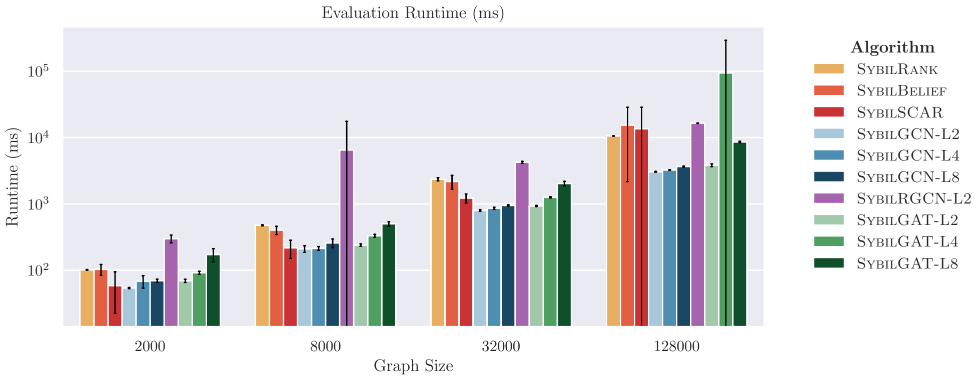

Figure 6.24: Runtimes of experiment from Section 6.5.1.

Chapter 7

Conclusion

7.1 Main Findings

This thesis has investigated the use of graph neural network (GNNs) for the task of structure-based Sybil detection on undirected social network graphs. To this end, it has evaluated the feasibility of using Graph Convolutional Networks (GCNs) and Relational Graph Convolutional Networks (RGCNs) and Graph Attention Networks (GATs). The resulting algorithms SybilGCN, SybilRGCN and SybilGAT were compared to the baselines SybilRank, SybilBelief, and SybilSCAR on both real-world and synthetic social network graphs.

Section 6.1 presented the evaluation in a setting where the GNN models were pre-trained on a small graph before being applied to a larger graph. The comparison with the baselines was done on the latter graph. In Section 6.1.1, the GNNs were pre-trained on a sampled subgraph that was 10% of the size of the full graph and evaluated on the remaining graph. The examined social networks were synthetically generated using a real-world Facebook social network graph, the Barabási-Albert (BA) model and the Power Law (PL) model by adding random attack edges. The results showed that the SybilGCN and SybilGAT algorithms clearly outperform the baselines. Section 6.1.2 examined how well the algorithms perform when pre-trained on a smaller network and then evaluated on a larger network that was generated with the same methodology. Pre-training was done on synthetic networks using the Barabási-Albert model and the Power Law model, while evaluation was conducted on larger versions of these graphs as well as a network constructed with a real-world Facebook network. This set of experiments show superiority of SybilGCN and SybilGAT over the baselines, albeit the difference to the baselines being less pronounced than in the previous experiment.

The evaluation in Section 6.2 conducted various ablation studies investigating the effects of different parameters in the construction of synthesized social networks. The studied effects were the underlying data model, size of the network, size of training (known) set and degree of label noise in the training set. The results of the inspection of these factors within the algorithms were somewhat unsurprising: the data model has only a minor influence on the performance, the size of the network does not affect performance, the algorithms perform better with more training data, and the algorithms are negatively affected by increasing degree of label noise. The degree to which the algorithms are affected by these factors offers a useful insight: SybilSCAR seems to be *relatively* more robust to lack of training data and increased label noise (but in all comparisons it is still inferior to the novel GNN-based algorithms, particularly with an increasing amount of attack edges). Comparing the algorithm to each other in each of the mentioned settings, the previously established superiority of the GNN-based algorithms was reiterated across the board.

The evaluation of the real-world labeled Twitter social network graph in Section 6.3 was conducted in the same way as the experiment in Section 6.1.1: The GNN algorithms were pre-trained on a sampled subgraph of the network (now only 5% of the full size) and evaluated on the remaining network. The results show superior performance of the SybilGAT algorithm over the baselines, while SybilGCN performed on par with the baselines.

Section 6.4 studied the effect of adversarial attacks on the network by adding targeted (rather than random) attacks. Section 6.4.1 conducted a robustness analysis with four different targeted attack strategies. The results showed that the GNN-based algorithms outperform the baselines, and SybilRGCN performs extraordinarily well. The experiment in Section 6.4.2 pre-trained on a network with random attack edges and evaluated on the same network that had been attacked with targeted attack edges. The results showed that the SybilGCN and SybilGAT algorithms can adapt to the targeted attacks and still perform well without the need to fine-tune on the network with targeted attack edges.

Various other effects and scenarios were examined in Section 6.5. Evaluating exponentially growing networks does not degrade the performance the studied algorithms. Evaluating networks where there is a heavy imbalance between the number of honest and Sybil nodes presents a challenge to some examined algorithms, but SybilRGCN can outperform the other algorithms in all scenarios, with the SybilGAT algorithms also performing well. All aforementioned trends of performance, which are in terms of AUC score, are mostly (but not all to the same extent) consistent with the examination of accuracy, precision and recall. Finally, the runtime analysis shows that after the pre-training process, the SybilGCN and SybilGAT algorithms are significantly faster than the baselines algorithms during inference.

In summary, the main findings of this thesis are as follows: The SybilGCN and SybilGAT algorithms have shown to be superior to the baselines in almost all examined experiment settings, including the evaluation on the real-world labeled Twitter network. SybilRGCN shows clears superiority when the attack edges increasingly target known nodes in the network and when there is a heavy imbalance between the number of honest and Sybil nodes.

7.2 Outlook

While the GNN-based algorithms SybilGCN, SybilRGCN and SybilGAT perform very well compared to the baselines, there are still some areas of limitations that can be explored.

As for the architecture of the GNN themselves, it seems that varying number of layers perform vastly differently in different scenarios. This is an avenue to be further explored, by potentially making the number of layers depend on the network size, network connectivity or some other characteristic. Additionally, the optimizer and loss function for these models is an unexplored area that could possibly further improve performance. Such experimentation could also be done with different activation functions and arrangement thereof within the network.

The way in which the GNN algorithms perform thresholding for the final classification could be improved, as the optimal threshold estimation done with the validation set seems to have potential for improvement, as the metrics other than AUC score indicate.

One could also consider using GNNs for different tasks than node classification but with the end goal of Sybil detection, for example predicting attack edges or classifying subgraphs (Sybil communities) directly.

The relational nature of the graph could be further improved, since the way edge types are grouped in SybilRGCN is a currently unexplored area.

The evaluations were conducted on undirected graphs, although the GNN algorithms could technically also run on directed graphs. Assessing the performance of these algorithms on directed graphs and evolving them is an important area of future work.

The robustness to label noise and lack of training data of the GNN algorithms is, in a relative sense, inferior to SybilSCAR. While it is not directly clear how this shortfall could be addressed, it is an area of further exploration.

The lack of available data was a major challenge. Investing more time and effort connecting to other institutions in order to collect data could potentially greatly improve the quality and quantity of future research of this nature.

On a conceptual level, there is still a lack of understanding as to why the proposed methods are superior. Learning the intricate details of the problem in conjunction with the GNN algorithms could yield insights that open avenues for further improvement.

Finally, on a more general scale, it seems that some of the assumptions that are made by previous research (and this work) should be challenged and either confirmed or dismissed [4, 5, 11, 12]. Some research regarding the question if some of these assumptions actually hold has already been conducted [23, 24], but more might be necessary.

Bibliography

[1] H. Yu, M. Kaminsky, P. B. Gibbons, and A. D. Flaxman, “SybilGuard: Defending against sybil attacks via social networks,” *IEEE/ACM Transactions on Networking*, vol. 16, no. 3, pp. 576–589, 2008.

[2] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao, “SybilLimit: A near-optimal social network defense against sybil attacks,” *IEEE/ACM Trans. Netw.*, vol. 18, no. 3, pp. 885–898, 2010.

[3] G. Danezis and P. Mittal, “SybilInfer: Detecting sybil nodes using social networks.” in *Network and Distributed System Security Symposium*, 01 2009.

[4] Q. Cao, M. Sirivianos, X. Yang, and T. Pregueiro, “Aiding the detection of fake accounts in large scale social online services,” in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, 2012.

[5] N. Z. Gong, M. Frank, and P. Mittal, “SybilBelief: A semi-supervised learning approach for structure-based sybil detection,” *IEEE Transactions on Information Forensics and Security*, 2014.

[6] Y. Boshmaf, D. Logothetis, G. Siganos, J. Lería, J. Lorenzo, M. Ripeanu, K. Beznosov, and H. Halawa, “Ìntegro: Leveraging victim prediction for robust fake account detection in large scale osns,” *Computers & Security*, 2016.

[7] J. Jia, B. Wang, and N. Z. Gong, “Random walk based fake account detection in online social networks,” in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2017, pp. 273–284.

[8] B. Wang, N. Z. Gong, and H. Fu, “GANG: Detecting fraudulent users in online social networks via guilt-by-association on directed graphs,” in *2017 IEEE International Conference on Data Mining (ICDM)*, 2017, pp. 465–474.

[9] B. Wang, L. Zhang, and N. Z. Gong, “SybilSCAR: Sybil detection in online social networks via local rule based propagation,” in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 2017, pp. 1–9.

[10] P. Gao, B. Wang, N. Z. Gong, S. R. Kulkarni, K. Thomas, and P. Mittal, “SybilFuse: Combining local attributes with global structure to perform robust sybil detection,” 2018. [Online]. Available: https://arxiv.org/abs/1803.06772

[11] B. Wang, J. Jia, L. Zhang, and N. Z. Gong, "Structure-based sybil detection in social networks via local rule-based propagation," 2020. [Online]. Available: https://arxiv.org/abs/1803.04321

[12] H. Lu, D. Gong, Z. Li, F. Liu, and F. Liu, "SybilHP: Sybil detection in directed social networks with adaptive homophily prediction," *Applied Sciences*, vol. 13, no. 9, 2023.

[13] M. Al-Qurishi, M. Alrakhami, A. Alamri, M. Alrubaian, S. M. M. Rahman, and M. S. Hossain, "Sybil defense techniques in online social networks: A survey," *IEEE Access*, vol. PP, pp. 1–1, 01 2017.

[14] A. Breuer, R. Eilat, and U. Weinsberg, "Friend or faux: Graph-based early detection of fake accounts on social networks," 2020. [Online]. Available: https://arxiv.org/abs/2004.04834

[15] Y. Sun, Z. Yang, and Y. Dai, "TrustGCN: Enabling graph convolutional network for robust sybil detection in osns," *2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 1–7, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:231645934

[16] S. Feng, H. Wan, N. Wang, and M. Luo, "BotRGCN: Twitter bot detection with relational graph convolutional networks," in *Proceedings of the 2021 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, ser. ASONAM '21. ACM, Nov. 2021. [Online]. Available: http://dx.doi.org/10.1145/3487351.3488336

[17] S. Feng, H. Wan, N. Wang, J. Li, and M. Luo, "SATAR: A self-supervised approach to twitter account representation learning and its application in bot detection," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, ser. CIKM '21. ACM, Oct. 2021. [Online]. Available: http://dx.doi.org/10.1145/3459637.3481949

[18] A. Breuer, N. Khosravani, M. Tingley, and B. Cottel, "Preemptive detection of fake accounts on social networks via multi-class preferential attachment classifiers," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 105–116. [Online]. Available: https://doi.org/10.1145/3580305.3599471

[19] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2016.

[20] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *Extended Semantic Web Conference*, 2017.

[21] R. Toivonen, J.-P. Onnela, J. Saramäki, J. Hyvönen, and K. Kaski, "A model for social networks," *Physica A: Statistical Mechanics and its Applications*, vol. 371, no. 2, pp. 851–860, 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378437106003931

[22] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 29–42. [Online]. Available: https://doi.org/10.1145/1298306.1298311

[23] A. Mohaisen, A. Yun, and Y. Kim, "Measuring the mixing time of social graphs," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: Association for Computing Machinery, 2010, pp. 383–389. [Online]. Available: https://doi.org/10.1145/1879141.1879191

[24] A. Mohaisen, H. Tran, N. Hopper, and Y. Kim, "On the mixing time of directed social graphs and security implications," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 36–37. [Online]. Available: https://doi.org/10.1145/2414456.2414476

[25] W. Wei, F. Xu, C. C. Tan, and Q. Li, "SybilDefender: A defense mechanism for sybil attacks in large social networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 12, pp. 2492–2502, 2013.

[26] P. Gao, N. Z. Gong, S. Kulkarni, K. Thomas, and P. Mittal, "SybilFrame: A defense-in-depth framework for structure-based sybil detection," 2018. [Online]. Available: https://arxiv.org/abs/1503.02985

[27] H. Asadian and H. H. S. Javadi, "Identification of sybil attacks on social networks using a framework based on user interactions," *SECURITY AND PRIVACY*, vol. 1, no. 2, p. e19, 2018. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spy2.19

[28] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018. [Online]. Available: https://arxiv.org/abs/1710.10903

[29] W. Jin, Y. Li, H. Xu, Y. Wang, S. Ji, C. Aggarwal, and J. Tang, "Adversarial attacks and defenses on graphs: A review and empirical study," *SIGKDD Explorations Newsletter*, vol. 22, no. 2, 2021.

[30] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[31] J. Leskovec and J. Mcauley, "Learning to discover social circles in ego networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/7a614fd06c325499f1680b9896beedeb-Paper.pdf

[32] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a social network or a news media?" in *WWW '10: Proceedings of the 19th international conference on World wide web*. New York, NY, USA: ACM, 2010, pp. 591–600.

[33] Q. Cao and X. Yang, "SybilFence: Improving social-graph-based sybil defenses with user negative feedback," 2013. [Online]. Available: https://arxiv.org/abs/1304.3819

[34] S. Misra, A. S. Md Tayeen, and W. Xu, "SybilExposer: An effective scheme to detect sybil communities in online social networks," in *2016 IEEE International Conference on Communications (ICC)*, 2016, pp. 1–6.

[35] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, p. 509–512, Oct. 1999. [Online]. Available: http://dx.doi.org/10.1126/science.286.5439.509

[36] P. Holme and B. J. Kim, "Growing scale-free networks with tunable clustering," *Physical Review E*, vol. 65, no. 2, Jan. 2002. [Online]. Available: http://dx.doi.org/10.1103/PhysRevE.65.026107

[37] G. Van Rossum and F. L. Drake Jr, *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[38] A. Hagberg, P. Swart, and D. Chult, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, 06 2008.

[39] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[40] M. L. Waskom, "seaborn: statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021. [Online]. Available: https://doi.org/10.21105/joss.03021

[41] B. Rozemberczki, O. Kiss, and R. Sarkar, "Little Ball of Fur: A python library for graph sampling," in *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*. ACM, 2020, p. 3133–3140.

[42] J. Leskovec and C. Faloutsos, "Sampling from large graphs," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 631–636. [Online]. Available: https://doi.org/10.1145/1150402.1150479

[43] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: Densification laws, shrinking diameters and possible explanations," in *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, ser. KDD '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 177–187. [Online]. Available: https://doi.org/10.1145/1081870.1081893

[44] C. Hübler, H.-P. Kriegel, K. Borgwardt, and Z. Ghahramani, "Metropolis algorithms for representative subgraph sampling," in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 283–292.

[45] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017. [Online]. Available: https://arxiv.org/abs/1412.6980

Appendix A

Appendix

A.1 Complete Experiment Data

A.1.1 Experiment 1: Pre-training Strategies

The complete data for the experiments from Section 6.1.1 and Section 6.1.2 can be found in Table A.1 and Table A.2 respectively.

A.1.2 Experiment 2: Robustness Analysis

The complete data for the experiments from Section 6.2 is too large to print, but can be found in the repository.[1]

A.1.3 Experiment 3: Real-World Twitter Dataset

The complete data for the experiments from Section 6.3 can be found in Table A.3.

A.1.4 Experiment 4: Adversarial Attack Study

The complete data for the experiments from Section 6.4.1 is too large to print, but can be found in the repository.[1]

The complete data for the experiments from Section 6.4.2 can be found in Table A.4.

A.1.5 Experiment 5: Miscellaneous

The complete data for the experiments from Section 6.5.1 and Section 6.5.2 can be found in Table A.5 and Table A.6 respectively.

[1] https://github.com/stuartheeb/GNN-sybil-detection.

Table A.1: Complete data for the experiments from Section 6.1.1. Runtimes are in seconds. Five runs for each experiment.

Variant	Algorithm	AUC	Accuracy	Precision	Recall	Pretrain Runtime	Runtime
facebook	\textsc{SybilRank}	0.449 ± 0.038	0.429 ± 0.043	0.430 ± 0.041	0.429 ± 0.042		1.020 ± 0.026
facebook	\textsc{SybilBelief}	0.627 ± 0.040	0.604 ± 0.010	0.740 ± 0.190	0.577 ± 0.356		1.684 ± 0.029
facebook	\textsc{SybilSCAR}	0.626 ± 0.017	0.548 ± 0.007	0.573 ± 0.041	0.501 ± 0.237		0.723 ± 0.025
facebook	\textsc{SybilGCN-L2}	0.808 ± 0.008	0.653 ± 0.030	0.871 ± 0.056	0.371 ± 0.104	5.658 ± 0.675	0.208 ± 0.014
facebook	\textsc{SybilGCN-L4}	0.690 ± 0.024	0.594 ± 0.026	0.608 ± 0.054	0.582 ± 0.102	9.320 ± 1.241	0.236 ± 0.011
facebook	\textsc{SybilGCN-L8}	0.621 ± 0.067	0.583 ± 0.019	0.569 ± 0.030	0.750 ± 0.154	14.297 ± 5.837	0.294 ± 0.015
facebook	\textsc{SybilRGCN-L2}	0.652 ± 0.108	0.572 ± 0.097	0.633 ± 0.146	0.372 ± 0.106	6.425 ± 0.614	2.304 ± 0.066
facebook	\textsc{SybilGAT-L2}	0.747 ± 0.017	0.629 ± 0.021	0.736 ± 0.076	0.466 ± 0.209	1.484 ± 0.463	0.290 ± 0.006
facebook	\textsc{SybilGAT-L4}	0.597 ± 0.029	0.548 ± 0.043	0.615 ± 0.180	0.486 ± 0.166	2.173 ± 1.295	0.460 ± 0.009
facebook	\textsc{SybilGAT-L8}	0.449 ± 0.034	0.578 ± 0.009	0.684 ± 0.176	0.647 ± 0.390	1.739 ± 0.432	0.834 ± 0.012
barabasi-albert	\textsc{SybilRank}	0.509 ± 0.082	0.494 ± 0.060	0.492 ± 0.058	0.494 ± 0.061		0.368 ± 0.002
barabasi-albert	\textsc{SybilBelief}	0.510 ± 0.006	0.502 ± 0.002	inf ± nan	0.400 ± 0.490		0.199 ± 0.016
barabasi-albert	\textsc{SybilSCAR}	0.612 ± 0.014	0.510 ± 0.012	0.520 ± 0.040	0.450 ± 0.399		0.186 ± 0.063
barabasi-albert	\textsc{SybilGCN-L2}	0.747 ± 0.010	0.507 ± 0.005	inf ± nan	0.012 ± 0.008	3.603 ± 0.037	0.128 ± 0.014
barabasi-albert	\textsc{SybilGCN-L4}	0.791 ± 0.014	0.655 ± 0.022	0.792 ± 0.021	0.418 ± 0.062	5.727 ± 0.095	0.132 ± 0.002
barabasi-albert	\textsc{SybilGCN-L8}	0.504 ± 0.005	0.502 ± 0.004	0.500 ± 0.004	0.700 ± 0.138	10.238 ± 0.167	0.161 ± 0.002
barabasi-albert	\textsc{SybilRGCN-L2}	0.605 ± 0.041	0.499 ± 0.018	0.513 ± 0.069	0.238 ± 0.096	4.017 ± 0.093	0.837 ± 0.009
barabasi-albert	\textsc{SybilGAT-L2}	0.749 ± 0.009	0.525 ± 0.032	0.860 ± 0.092	0.064 ± 0.099	0.594 ± 0.091	0.160 ± 0.003
barabasi-albert	\textsc{SybilGAT-L4}	0.801 ± 0.010	0.623 ± 0.061	0.804 ± 0.085	0.380 ± 0.260	0.544 ± 0.120	0.239 ± 0.006
barabasi-albert	\textsc{SybilGAT-L8}	0.832 ± 0.007	0.695 ± 0.031	0.810 ± 0.073	0.541 ± 0.170	0.642 ± 0.137	0.396 ± 0.018
power-law	\textsc{SybilRank}	0.511 ± 0.052	0.493 ± 0.038	0.492 ± 0.039	0.497 ± 0.038		0.369 ± 0.002
power-law	\textsc{SybilBelief}	0.512 ± 0.008	0.501 ± 0.004	inf ± nan	0.000 ± 0.000		0.233 ± 0.034
power-law	\textsc{SybilSCAR}	0.600 ± 0.018	0.524 ± 0.017	0.539 ± 0.033	0.246 ± 0.103		0.136 ± 0.002
power-law	\textsc{SybilGCN-L2}	0.733 ± 0.018	0.512 ± 0.007	0.907 ± 0.040	0.024 ± 0.011	3.638 ± 0.086	0.120 ± 0.003
power-law	\textsc{SybilGCN-L4}	0.752 ± 0.021	0.604 ± 0.039	0.753 ± 0.030	0.311 ± 0.124	5.664 ± 0.054	0.129 ± 0.006
power-law	\textsc{SybilGCN-L8}	0.551 ± 0.078	0.532 ± 0.054	0.530 ± 0.051	0.691 ± 0.140	10.245 ± 0.101	0.158 ± 0.008
power-law	\textsc{SybilRGCN-L2}	0.608 ± 0.043	0.509 ± 0.026	0.537 ± 0.075	0.207 ± 0.070	3.990 ± 0.057	0.830 ± 0.005
power-law	\textsc{SybilGAT-L2}	0.733 ± 0.017	0.522 ± 0.009	0.890 ± 0.082	0.051 ± 0.030	0.729 ± 0.223	0.164 ± 0.007
power-law	\textsc{SybilGAT-L4}	0.753 ± 0.017	0.618 ± 0.028	0.751 ± 0.030	0.356 ± 0.086	0.625 ± 0.277	0.242 ± 0.008
power-law	\textsc{SybilGAT-L8}	0.739 ± 0.015	0.657 ± 0.009	0.678 ± 0.030	0.608 ± 0.074	0.589 ± 0.103	0.380 ± 0.006

Table A.2: Complete data for the experiments from Section 6.1.2. Runtimes are in seconds. Five runs for each experiment.

Variant	Algorithm	AUC	Accuracy	Precision	Recall	Pretrain Runtime	Runtime
BA-BA	\textsc{SybilRank}	0.807 ± 0.010	0.725 ± 0.007	0.725 ± 0.007	0.725 ± 0.008		1.529 ± 0.145
BA-BA	\textsc{SybilBelief}	0.514 ± 0.001	0.500 ± 0.000	inf ± nan	0.400 ± 0.490		1.263 ± 0.135
BA-BA	\textsc{SybilSCAR}	0.674 ± 0.007	0.601 ± 0.033	0.622 ± 0.021	0.517 ± 0.172		0.764 ± 0.076
BA-BA	\textsc{SybilGCN-L2}	0.739 ± 0.005	0.520 ± 0.011	0.857 ± 0.028	0.050 ± 0.030	9.857 ± 0.630	0.512 ± 0.006
BA-BA	\textsc{SybilGCN-L4}	0.783 ± 0.005	0.685 ± 0.012	0.700 ± 0.035	0.663 ± 0.104	15.629 ± 0.868	0.549 ± 0.012
BA-BA	\textsc{SybilGCN-L8}	0.775 ± 0.140	0.699 ± 0.100	inf ± nan	0.605 ± 0.305	23.368 ± 11.496	0.616 ± 0.017
BA-BA	\textsc{SybilRGCN-L2}	0.623 ± 0.055	0.489 ± 0.020	0.521 ± 0.169	0.098 ± 0.035	11.091 ± 0.332	3.042 ± 0.060
BA-BA	\textsc{SybilGAT-L2}	0.745 ± 0.006	0.552 ± 0.055	0.813 ± 0.104	0.201 ± 0.271	2.388 ± 0.336	0.602 ± 0.013
BA-BA	\textsc{SybilGAT-L4}	0.808 ± 0.006	0.709 ± 0.016	0.719 ± 0.050	0.709 ± 0.099	2.225 ± 0.535	0.792 ± 0.012
BA-BA	\textsc{SybilGAT-L8}	0.859 ± 0.004	0.755 ± 0.005	0.779 ± 0.052	0.729 ± 0.092	2.917 ± 1.308	1.295 ± 0.150
BA-PL	\textsc{SybilRank}	0.631 ± 0.011	0.590 ± 0.010	0.589 ± 0.010	0.595 ± 0.011		1.471 ± 0.017
BA-PL	\textsc{SybilBelief}	0.515 ± 0.003	0.500 ± 0.000	inf ± nan	0.400 ± 0.490		1.709 ± 0.447
BA-PL	\textsc{SybilSCAR}	0.664 ± 0.008	0.598 ± 0.015	0.616 ± 0.012	0.524 ± 0.087		0.840 ± 0.083
BA-PL	\textsc{SybilGCN-L2}	0.728 ± 0.004	0.557 ± 0.035	0.792 ± 0.103	0.244 ± 0.291	10.005 ± 1.029	0.509 ± 0.005
BA-PL	\textsc{SybilGCN-L4}	0.730 ± 0.005	0.634 ± 0.011	0.688 ± 0.023	0.499 ± 0.076	16.592 ± 0.723	0.547 ± 0.018
BA-PL	\textsc{SybilGCN-L8}	0.655 ± 0.149	0.619 ± 0.061	inf ± nan	0.486 ± 0.251	23.388 ± 11.551	0.670 ± 0.099
BA-PL	\textsc{SybilRGCN-L2}	0.611 ± 0.041	0.504 ± 0.024	0.553 ± 0.143	0.131 ± 0.062	11.362 ± 0.661	3.028 ± 0.105
BA-PL	\textsc{SybilGAT-L2}	0.740 ± 0.006	0.571 ± 0.060	0.812 ± 0.116	0.252 ± 0.244	2.457 ± 0.335	0.664 ± 0.103
BA-PL	\textsc{SybilGAT-L4}	0.761 ± 0.007	0.663 ± 0.012	0.715 ± 0.036	0.554 ± 0.086	2.414 ± 0.102	0.842 ± 0.094
BA-PL	\textsc{SybilGAT-L8}	0.745 ± 0.009	0.665 ± 0.008	0.661 ± 0.027	0.691 ± 0.071	2.821 ± 1.141	1.189 ± 0.043
BA-FB	\textsc{SybilRank}	0.399 ± 0.024	0.374 ± 0.035	0.374 ± 0.035	0.374 ± 0.035		1.333 ± 0.012
BA-FB	\textsc{SybilBelief}	0.713 ± 0.036	0.620 ± 0.015	0.604 ± 0.021	0.725 ± 0.172		2.207 ± 0.029
BA-FB	\textsc{SybilSCAR}	0.629 ± 0.011	0.534 ± 0.007	0.530 ± 0.011	0.628 ± 0.137		0.925 ± 0.074
BA-FB	\textsc{SybilGCN-L2}	0.824 ± 0.005	0.691 ± 0.012	0.843 ± 0.080	0.496 ± 0.121	10.372 ± 0.977	0.230 ± 0.004
BA-FB	\textsc{SybilGCN-L4}	0.706 ± 0.024	0.600 ± 0.020	0.625 ± 0.058	0.541 ± 0.081	17.197 ± 1.008	0.333 ± 0.070
BA-FB	\textsc{SybilGCN-L8}	0.587 ± 0.068	0.575 ± 0.041	inf ± nan	0.564 ± 0.380	23.906 ± 11.806	0.453 ± 0.182
BA-FB	\textsc{SybilRGCN-L2}	0.640 ± 0.092	0.548 ± 0.048	0.618 ± 0.080	0.366 ± 0.220	11.528 ± 0.733	3.476 ± 0.089
BA-FB	\textsc{SybilGAT-L2}	0.761 ± 0.011	0.647 ± 0.009	0.676 ± 0.044	0.595 ± 0.129	2.263 ± 0.336	0.352 ± 0.012
BA-FB	\textsc{SybilGAT-L4}	0.593 ± 0.024	0.559 ± 0.028	0.585 ± 0.086	0.572 ± 0.235	2.672 ± 0.570	0.570 ± 0.007
BA-FB	\textsc{SybilGAT-L8}	0.453 ± 0.021	0.585 ± 0.009	0.680 ± 0.178	0.676 ± 0.386	3.981 ± 1.410	1.106 ± 0.206
PL-BA	\textsc{SybilRank}	0.802 ± 0.008	0.719 ± 0.006	0.719 ± 0.006	0.719 ± 0.006		1.415 ± 0.015
PL-BA	\textsc{SybilBelief}	0.512 ± 0.001	0.500 ± 0.000	inf ± nan	0.400 ± 0.490		1.275 ± 0.201
PL-BA	\textsc{SybilSCAR}	0.688 ± 0.013	0.613 ± 0.039	0.639 ± 0.030	0.511 ± 0.153		0.717 ± 0.064
PL-BA	\textsc{SybilGCN-L2}	0.740 ± 0.003	0.529 ± 0.014	0.857 ± 0.035	0.071 ± 0.037	9.871 ± 0.534	0.499 ± 0.007
PL-BA	\textsc{SybilGCN-L4}	0.784 ± 0.005	0.658 ± 0.023	0.773 ± 0.044	0.462 ± 0.117	17.103 ± 1.014	0.540 ± 0.008
PL-BA	\textsc{SybilGCN-L8}	0.733 ± 0.198	0.688 ± 0.095	inf ± nan	0.581 ± 0.296	25.329 ± 12.503	0.606 ± 0.022
PL-BA	\textsc{SybilRGCN-L2}	0.611 ± 0.073	0.496 ± 0.036	0.557 ± 0.171	0.202 ± 0.154	11.236 ± 1.107	2.961 ± 0.029
PL-BA	\textsc{SybilGAT-L2}	0.753 ± 0.004	0.535 ± 0.050	0.863 ± 0.073	0.100 ± 0.154	2.521 ± 0.316	0.607 ± 0.015
PL-BA	\textsc{SybilGAT-L4}	0.813 ± 0.005	0.691 ± 0.019	0.800 ± 0.026	0.513 ± 0.075	2.305 ± 0.832	0.781 ± 0.013
PL-BA	\textsc{SybilGAT-L8}	0.755 ± 0.220	0.711 ± 0.106	0.739 ± 0.122	0.767 ± 0.129	3.303 ± 1.903	1.152 ± 0.030
PL-PL	\textsc{SybilRank}	0.641 ± 0.013	0.598 ± 0.011	0.597 ± 0.011	0.604 ± 0.011		1.577 ± 0.267
PL-PL	\textsc{SybilBelief}	0.513 ± 0.002	0.500 ± 0.000	inf ± nan	0.400 ± 0.490		1.220 ± 0.097
PL-PL	\textsc{SybilSCAR}	0.665 ± 0.015	0.577 ± 0.021	0.591 ± 0.035	0.555 ± 0.196		0.829 ± 0.167
PL-PL	\textsc{SybilGCN-L2}	0.727 ± 0.004	0.532 ± 0.037	0.881 ± 0.076	0.089 ± 0.119	10.613 ± 2.808	0.620 ± 0.217
PL-PL	\textsc{SybilGCN-L4}	0.730 ± 0.003	0.632 ± 0.004	0.691 ± 0.011	0.477 ± 0.034	16.409 ± 2.097	0.616 ± 0.128
PL-PL	\textsc{SybilGCN-L8}	0.639 ± 0.139	0.605 ± 0.054	inf ± nan	0.487 ± 0.247	24.662 ± 12.737	0.618 ± 0.021
PL-PL	\textsc{SybilRGCN-L2}	0.588 ± 0.036	0.490 ± 0.029	0.480 ± 0.070	0.158 ± 0.019	10.584 ± 0.426	2.987 ± 0.036
PL-PL	\textsc{SybilGAT-L2}	0.741 ± 0.009	0.577 ± 0.051	0.801 ± 0.091	0.248 ± 0.196	2.580 ± 0.368	0.656 ± 0.071
PL-PL	\textsc{SybilGAT-L4}	0.756 ± 0.006	0.660 ± 0.006	0.671 ± 0.044	0.656 ± 0.116	3.232 ± 0.838	0.812 ± 0.039
PL-PL	\textsc{SybilGAT-L8}	0.743 ± 0.006	0.659 ± 0.007	0.688 ± 0.013	0.583 ± 0.054	2.841 ± 1.213	1.199 ± 0.065
PL-FB	\textsc{SybilRank}	0.414 ± 0.045	0.386 ± 0.043	0.386 ± 0.043	0.387 ± 0.043		1.316 ± 0.010
PL-FB	\textsc{SybilBelief}	0.730 ± 0.005	0.614 ± 0.013	0.612 ± 0.029	0.643 ± 0.101		2.148 ± 0.013
PL-FB	\textsc{SybilSCAR}	0.646 ± 0.013	0.550 ± 0.017	0.564 ± 0.042	0.530 ± 0.181		0.883 ± 0.036
PL-FB	\textsc{SybilGCN-L2}	0.826 ± 0.014	0.676 ± 0.023	0.882 ± 0.066	0.419 ± 0.101	9.109 ± 0.404	0.229 ± 0.005
PL-FB	\textsc{SybilGCN-L4}	0.716 ± 0.014	0.593 ± 0.021	0.609 ± 0.038	0.546 ± 0.107	15.166 ± 0.520	0.262 ± 0.006
PL-FB	\textsc{SybilGCN-L8}	0.564 ± 0.073	0.580 ± 0.048	0.659 ± 0.154	0.490 ± 0.293	22.216 ± 10.969	0.344 ± 0.003
PL-FB	\textsc{SybilRGCN-L2}	0.621 ± 0.105	0.560 ± 0.071	0.615 ± 0.108	0.452 ± 0.182	10.146 ± 0.265	3.340 ± 0.025
PL-FB	\textsc{SybilGAT-L2}	0.767 ± 0.018	0.645 ± 0.017	0.715 ± 0.096	0.540 ± 0.126	2.288 ± 0.381	0.332 ± 0.006
PL-FB	\textsc{SybilGAT-L4}	0.612 ± 0.007	0.562 ± 0.027	0.558 ± 0.025	0.624 ± 0.223	1.801 ± 0.447	0.562 ± 0.038
PL-FB	\textsc{SybilGAT-L8}	0.485 ± 0.033	0.586 ± 0.010	0.777 ± 0.191	0.461 ± 0.332	3.693 ± 1.424	1.004 ± 0.058

Table A.3: Complete data for the experiments from Section 6.3. Runtimes are in seconds. Five runs for each experiment.

Algorithm	AUC	Accuracy	Precision	Recall	Pretrain Runtime	Runtime
\textsc{SybilRank}	0.738 ± 0.002	0.704 ± 0.003	0.536 ± 0.005	0.836 ± 0.004		19.711 ± 0.611
\textsc{SybilBelief}	0.526 ± 0.029	0.705 ± 0.012	0.887 ± 0.069	0.134 ± 0.040		63.553 ± 77.938
\textsc{SybilSCAR}	0.802 ± 0.005	0.671 ± 0.001	0.718 ± 0.007	0.027 ± 0.004		15.197 ± 2.220
\textsc{SybilGCN-L2}	0.802 ± 0.001	0.514 ± 0.008	0.403 ± 0.004	0.944 ± 0.006	211.061 ± 174.829	6.241 ± 0.125
\textsc{SybilGCN-L4}	0.796 ± 0.003	0.615 ± 0.019	0.458 ± 0.013	0.804 ± 0.030	416.767 ± 354.108	6.503 ± 0.067
\textsc{SybilGCN-L8}	0.702 ± 0.004	0.696 ± 0.016	0.547 ± 0.026	0.544 ± 0.014	484.025 ± 101.993	7.126 ± 0.098
\textsc{SybilRGCN-L2}	0.624 ± 0.045	0.434 ± 0.019	0.363 ± 0.012	0.917 ± 0.046	192.744 ± 84.128	32.761 ± 0.791
\textsc{SybilGAT-L2}	0.821 ± 0.005	0.608 ± 0.009	0.457 ± 0.007	0.913 ± 0.010	220.234 ± 185.319	7.381 ± 0.178
\textsc{SybilGAT-L4}	0.842 ± 0.007	0.666 ± 0.009	0.501 ± 0.008	0.918 ± 0.015	66.853 ± 13.940	10.351 ± 0.682
\textsc{SybilGAT-L8}	0.794 ± 0.015	0.664 ± 0.008	0.499 ± 0.007	0.929 ± 0.010	289.000 ± 347.859	67.897 ± 105.297

Table A.4: Complete data for the experiments from Section 6.4.2. Runtimes are in seconds. Five runs for each experiment.

Variant	Algorithm	AUC	Accuracy	Precision	Recall	Pretrain Runtime	Runtime
BA	\textsc{SybilRank}	0.736 ± 0.344	0.699 ± 0.295	0.706 ± 0.296	0.691 ± 0.289		0.134 ± 0.002
BA	\textsc{SybilBelief}	0.504 ± 0.003	0.500 ± 0.000	inf ± nan	0.000 ± 0.000		0.092 ± 0.011
BA	\textsc{SybilSCAR}	0.727 ± 0.026	0.521 ± 0.022	inf ± nan	0.053 ± 0.055		0.079 ± 0.041
BA	\textsc{SybilGCN-L2}	0.822 ± 0.016	0.504 ± 0.005	1.000 ± 0.000	0.008 ± 0.010	12.863 ± 0.780	0.059 ± 0.007
BA	\textsc{SybilGCN-L4}	0.920 ± 0.015	0.699 ± 0.078	0.940 ± 0.050	0.437 ± 0.190	37.083 ± 31.509	0.067 ± 0.005
BA	\textsc{SybilGCN-L8}	0.536 ± 0.033	0.515 ± 0.022	0.577 ± 0.143	0.573 ± 0.235	31.354 ± 15.515	0.076 ± 0.002
BA	\textsc{SybilRGCN-L2}	0.612 ± 0.116	0.507 ± 0.073	inf ± nan	0.277 ± 0.163	14.335 ± 0.908	0.450 ± 0.014
BA	\textsc{SybilGAT-L2}	0.846 ± 0.012	0.508 ± 0.007	0.964 ± 0.055	0.016 ± 0.013	3.751 ± 0.678	0.074 ± 0.004
BA	\textsc{SybilGAT-L4}	0.959 ± 0.007	0.710 ± 0.064	0.991 ± 0.011	0.425 ± 0.134	52.446 ± 98.750	0.117 ± 0.014
BA	\textsc{SybilGAT-L8}	0.984 ± 0.001	0.841 ± 0.057	0.992 ± 0.008	0.688 ± 0.120	5.250 ± 1.596	0.191 ± 0.019
PL	\textsc{SybilRank}	0.589 ± 0.178	0.558 ± 0.141	0.559 ± 0.141	0.561 ± 0.139		0.164 ± 0.057
PL	\textsc{SybilBelief}	0.507 ± 0.004	0.500 ± 0.000	inf ± nan	0.000 ± 0.000		0.116 ± 0.052
PL	\textsc{SybilSCAR}	0.682 ± 0.027	0.517 ± 0.020	inf ± nan	0.043 ± 0.057		0.117 ± 0.052
PL	\textsc{SybilGCN-L2}	0.796 ± 0.029	0.520 ± 0.010	0.943 ± 0.027	0.043 ± 0.021	14.290 ± 2.767	0.067 ± 0.021
PL	\textsc{SybilGCN-L4}	0.861 ± 0.036	0.666 ± 0.081	0.899 ± 0.061	0.395 ± 0.225	78.128 ± 103.716	0.072 ± 0.024
PL	\textsc{SybilGCN-L8}	0.530 ± 0.022	0.511 ± 0.018	0.518 ± 0.027	0.569 ± 0.216	91.020 ± 118.507	0.088 ± 0.029
PL	\textsc{SybilRGCN-L2}	0.584 ± 0.077	0.494 ± 0.023	inf ± nan	0.147 ± 0.140	15.556 ± 3.103	0.534 ± 0.183
PL	\textsc{SybilGAT-L2}	0.805 ± 0.037	0.539 ± 0.036	0.939 ± 0.078	0.092 ± 0.091	3.846 ± 1.156	0.088 ± 0.025
PL	\textsc{SybilGAT-L4}	0.903 ± 0.021	0.678 ± 0.079	0.964 ± 0.017	0.372 ± 0.173	4.343 ± 1.371	0.132 ± 0.034
PL	\textsc{SybilGAT-L8}	0.965 ± 0.002	0.758 ± 0.046	0.990 ± 0.005	0.523 ± 0.096	7.071 ± 3.509	0.222 ± 0.056

Table A.5: Complete data for the experiments from Section 6.5.1. Runtimes are in seconds. Five runs for each experiment.

Graph Size	Algorithm	AUC	Accuracy	Precision	Recall	Pretrain Runtime	Runtime
2000	\textsc{SybilRank}	0.612 ± 0.037	0.573 ± 0.028	0.572 ± 0.028	0.579 ± 0.029		0.101 ± 0.001
2000	\textsc{SybilBelief}	0.508 ± 0.004	0.500 ± 0.000	inf ± nan	0.400 ± 0.490		0.103 ± 0.017
2000	\textsc{SybilSCAR}	0.664 ± 0.019	0.539 ± 0.030	0.570 ± 0.058	0.581 ± 0.332		0.059 ± 0.032
2000	\textsc{SybilGCN-L2}	0.731 ± 0.009	0.505 ± 0.001	0.951 ± 0.060	0.010 ± 0.003	9.205 ± 0.291	0.054 ± 0.001
2000	\textsc{SybilGCN-L4}	0.739 ± 0.008	0.613 ± 0.036	0.746 ± 0.043	0.362 ± 0.154	15.728 ± 0.885	0.068 ± 0.013
2000	\textsc{SybilGCN-L8}	0.564 ± 0.118	0.538 ± 0.038	inf ± nan	0.417 ± 0.282	22.273 ± 10.945	0.070 ± 0.003
2000	\textsc{SybilRGCN-L2}	0.627 ± 0.048	0.520 ± 0.044	0.555 ± 0.094	0.136 ± 0.111	10.411 ± 0.146	0.299 ± 0.036
2000	\textsc{SybilGAT-L2}	0.734 ± 0.007	0.518 ± 0.022	0.915 ± 0.085	0.045 ± 0.057	2.330 ± 0.251	0.069 ± 0.003
2000	\textsc{SybilGAT-L4}	0.753 ± 0.013	0.583 ± 0.051	0.829 ± 0.068	0.225 ± 0.165	2.095 ± 0.372	0.092 ± 0.004
2000	\textsc{SybilGAT-L8}	0.737 ± 0.011	0.649 ± 0.018	0.686 ± 0.020	0.560 ± 0.101	2.840 ± 2.338	0.172 ± 0.035
4000	\textsc{SybilRank}	0.708 ± 0.034	0.651 ± 0.031	0.650 ± 0.030	0.656 ± 0.031		0.225 ± 0.001
4000	\textsc{SybilBelief}	0.513 ± 0.002	0.500 ± 0.000	inf ± nan	0.600 ± 0.490		0.216 ± 0.023
4000	\textsc{SybilSCAR}	0.667 ± 0.003	0.560 ± 0.026	0.579 ± 0.048	0.606 ± 0.274		0.128 ± 0.048
4000	\textsc{SybilGCN-L2}	0.730 ± 0.010	0.517 ± 0.017	0.922 ± 0.059	0.041 ± 0.042	10.085 ± 0.514	0.119 ± 0.024
4000	\textsc{SybilGCN-L4}	0.733 ± 0.015	0.608 ± 0.009	0.709 ± 0.069	0.437 ± 0.226	16.409 ± 0.818	0.116 ± 0.011
4000	\textsc{SybilGCN-L8}	0.582 ± 0.110	0.553 ± 0.036	inf ± nan	0.405 ± 0.244	23.615 ± 11.691	0.128 ± 0.006
4000	\textsc{SybilRGCN-L2}	0.598 ± 0.050	0.514 ± 0.033	0.539 ± 0.098	0.175 ± 0.073	10.614 ± 0.371	0.560 ± 0.027
4000	\textsc{SybilGAT-L2}	0.738 ± 0.006	0.536 ± 0.021	0.844 ± 0.070	0.096 ± 0.066	2.368 ± 0.459	0.128 ± 0.010
4000	\textsc{SybilGAT-L4}	0.753 ± 0.013	0.637 ± 0.026	0.735 ± 0.052	0.459 ± 0.169	2.121 ± 0.634	0.184 ± 0.015
4000	\textsc{SybilGAT-L8}	0.734 ± 0.018	0.652 ± 0.024	0.677 ± 0.018	0.583 ± 0.097	3.279 ± 1.733	0.296 ± 0.029
8000	\textsc{SybilRank}	0.626 ± 0.009	0.585 ± 0.008	0.584 ± 0.008	0.590 ± 0.008		0.476 ± 0.007
8000	\textsc{SybilBelief}	0.513 ± 0.001	0.500 ± 0.000	inf ± nan	0.400 ± 0.490		0.403 ± 0.051
8000	\textsc{SybilSCAR}	0.674 ± 0.012	0.550 ± 0.027	0.611 ± 0.079	0.460 ± 0.317		0.218 ± 0.059
8000	\textsc{SybilGCN-L2}	0.733 ± 0.008	0.522 ± 0.012	0.907 ± 0.056	0.051 ± 0.029	38.954 ± 53.183	0.210 ± 0.021
8000	\textsc{SybilGCN-L4}	0.734 ± 0.009	0.621 ± 0.018	0.722 ± 0.045	0.416 ± 0.126	18.405 ± 3.840	0.214 ± 0.012
8000	\textsc{SybilGCN-L8}	0.628 ± 0.093	0.583 ± 0.057	inf ± nan	0.412 ± 0.219	48.190 ± 37.169	0.258 ± 0.034
8000	\textsc{SybilRGCN-L2}	0.575 ± 0.041	0.469 ± 0.019	0.412 ± 0.054	0.137 ± 0.022	66.321 ± 111.466	6.493 ± 9.894
8000	\textsc{SybilGAT-L2}	0.740 ± 0.008	0.558 ± 0.048	0.828 ± 0.093	0.178 ± 0.182	7.453 ± 10.197	0.240 ± 0.009
8000	\textsc{SybilGAT-L4}	0.760 ± 0.009	0.635 ± 0.021	0.731 ± 0.075	0.495 ± 0.223	2.335 ± 0.708	0.331 ± 0.015
8000	\textsc{SybilGAT-L8}	0.736 ± 0.020	0.649 ± 0.023	0.708 ± 0.017	0.512 ± 0.101	7.655 ± 9.742	0.501 ± 0.035
16000	\textsc{SybilRank}	0.708 ± 0.011	0.652 ± 0.009	0.651 ± 0.009	0.658 ± 0.009		1.095 ± 0.042
16000	\textsc{SybilBelief}	0.513 ± 0.002	0.500 ± 0.000	0.500 ± 0.000	1.000 ± 0.000		12.516 ± 22.870
16000	\textsc{SybilSCAR}	0.656 ± 0.009	0.571 ± 0.014	0.555 ± 0.018	0.741 ± 0.063		0.592 ± 0.029
16000	\textsc{SybilGCN-L2}	0.725 ± 0.004	0.537 ± 0.022	0.846 ± 0.037	0.094 ± 0.064	11.902 ± 0.976	0.405 ± 0.009
16000	\textsc{SybilGCN-L4}	0.725 ± 0.008	0.637 ± 0.006	0.649 ± 0.024	0.608 ± 0.092	221.307 ± 404.699	0.446 ± 0.011
16000	\textsc{SybilGCN-L8}	0.662 ± 0.082	0.598 ± 0.050	inf ± nan	0.493 ± 0.259	63.278 ± 75.466	0.487 ± 0.010
16000	\textsc{SybilRGCN-L2}	0.594 ± 0.026	0.499 ± 0.014	0.517 ± 0.063	0.138 ± 0.047	81.273 ± 137.233	2.178 ± 0.040
16000	\textsc{SybilGAT-L2}	0.733 ± 0.003	0.558 ± 0.040	0.808 ± 0.071	0.173 ± 0.147	2.420 ± 0.270	0.486 ± 0.011
16000	\textsc{SybilGAT-L4}	0.750 ± 0.009	0.658 ± 0.007	0.658 ± 0.039	0.682 ± 0.102	1.990 ± 0.784	0.660 ± 0.017
16000	\textsc{SybilGAT-L8}	0.732 ± 0.012	0.653 ± 0.011	0.650 ± 0.030	0.677 ± 0.061	3.608 ± 2.456	0.996 ± 0.042
32000	\textsc{SybilRank}	0.639 ± 0.010	0.598 ± 0.007	0.597 ± 0.007	0.603 ± 0.007		2.354 ± 0.116
32000	\textsc{SybilBelief}	0.513 ± 0.000	0.500 ± 0.000	inf ± nan	0.600 ± 0.490		2.177 ± 0.459
32000	\textsc{SybilSCAR}	0.660 ± 0.004	0.567 ± 0.026	0.565 ± 0.029	0.656 ± 0.189		1.222 ± 0.169
32000	\textsc{SybilGCN-L2}	0.728 ± 0.003	0.536 ± 0.017	0.860 ± 0.055	0.091 ± 0.050	10.529 ± 1.244	0.798 ± 0.020
32000	\textsc{SybilGCN-L4}	0.731 ± 0.005	0.640 ± 0.004	0.669 ± 0.028	0.568 ± 0.081	17.518 ± 2.322	0.860 ± 0.028
32000	\textsc{SybilGCN-L8}	0.603 ± 0.114	0.574 ± 0.040	inf ± nan	0.449 ± 0.226	26.591 ± 13.443	0.948 ± 0.019
32000	\textsc{SybilRGCN-L2}	0.601 ± 0.039	0.494 ± 0.020	0.495 ± 0.060	0.151 ± 0.049	11.798 ± 1.054	4.240 ± 0.121
32000	\textsc{SybilGAT-L2}	0.742 ± 0.002	0.596 ± 0.050	0.769 ± 0.089	0.321 ± 0.199	2.868 ± 0.637	0.931 ± 0.016
32000	\textsc{SybilGAT-L4}	0.759 ± 0.002	0.663 ± 0.007	0.691 ± 0.036	0.610 ± 0.105	2.296 ± 0.344	1.256 ± 0.023
32000	\textsc{SybilGAT-L8}	0.529 ± 0.218	0.575 ± 0.069	inf ± nan	0.440 ± 0.380	3.078 ± 2.148	2.029 ± 0.139
64000	\textsc{SybilRank}	0.700 ± 0.007	0.646 ± 0.006	0.645 ± 0.006	0.652 ± 0.006		4.934 ± 0.057
64000	\textsc{SybilBelief}	0.513 ± 0.001	0.500 ± 0.000	inf ± nan	0.400 ± 0.490		4.504 ± 0.610
64000	\textsc{SybilSCAR}	0.667 ± 0.005	0.591 ± 0.023	0.595 ± 0.037	0.630 ± 0.136		2.951 ± 0.315
64000	\textsc{SybilGCN-L2}	0.726 ± 0.002	0.566 ± 0.022	0.794 ± 0.035	0.186 ± 0.081	9.211 ± 0.476	1.566 ± 0.019
64000	\textsc{SybilGCN-L4}	0.725 ± 0.004	0.635 ± 0.004	0.667 ± 0.014	0.542 ± 0.044	15.599 ± 0.855	1.630 ± 0.024
64000	\textsc{SybilGCN-L8}	0.632 ± 0.124	0.577 ± 0.065	inf ± nan	0.394 ± 0.321	17.544 ± 13.872	1.837 ± 0.028
64000	\textsc{SybilRGCN-L2}	0.596 ± 0.031	0.510 ± 0.034	0.511 ± 0.062	0.215 ± 0.102	10.477 ± 0.228	8.282 ± 0.064
64000	\textsc{SybilGAT-L2}	0.743 ± 0.004	0.596 ± 0.036	0.778 ± 0.069	0.292 ± 0.130	2.501 ± 0.240	1.837 ± 0.027
64000	\textsc{SybilGAT-L4}	0.759 ± 0.005	0.665 ± 0.004	0.688 ± 0.031	0.619 ± 0.083	2.710 ± 0.509	2.498 ± 0.030
64000	\textsc{SybilGAT-L8}	0.743 ± 0.004	0.663 ± 0.005	0.680 ± 0.021	0.619 ± 0.060	3.571 ± 1.166	3.922 ± 0.079
128000	\textsc{SybilRank}	0.642 ± 0.005	0.600 ± 0.004	0.599 ± 0.004	0.605 ± 0.004		10.574 ± 0.062
128000	\textsc{SybilBelief}	0.513 ± 0.001	0.500 ± 0.000	inf ± nan	0.600 ± 0.490		15.391 ± 11.823
128000	\textsc{SybilSCAR}	0.662 ± 0.005	0.593 ± 0.016	0.594 ± 0.018	0.607 ± 0.134		13.568 ± 13.466
128000	\textsc{SybilGCN-L2}	0.722 ± 0.003	0.567 ± 0.027	0.788 ± 0.048	0.197 ± 0.111	9.548 ± 0.917	3.049 ± 0.033
128000	\textsc{SybilGCN-L4}	0.724 ± 0.006	0.638 ± 0.006	0.649 ± 0.014	0.604 ± 0.057	15.598 ± 1.168	3.241 ± 0.026
128000	\textsc{SybilGCN-L8}	0.683 ± 0.094	0.620 ± 0.061	inf ± nan	0.509 ± 0.257	76.945 ± 110.956	3.660 ± 0.050
128000	\textsc{SybilRGCN-L2}	0.606 ± 0.048	0.537 ± 0.030	0.570 ± 0.081	0.320 ± 0.140	10.551 ± 0.918	16.459 ± 0.090
128000	\textsc{SybilGAT-L2}	0.741 ± 0.002	0.618 ± 0.042	0.734 ± 0.077	0.414 ± 0.183	2.869 ± 0.964	3.830 ± 0.159
128000	\textsc{SybilGAT-L4}	0.759 ± 0.004	0.670 ± 0.004	0.676 ± 0.012	0.656 ± 0.022	2.453 ± 0.674	94.050 ± 177.445
128000	\textsc{SybilGAT-L8}	0.694 ± 0.103	0.633 ± 0.066	0.666 ± 0.008	0.535 ± 0.268	3.246 ± 1.643	8.577 ± 0.181

Table A.6: Complete data for the experiments from Section 6.5.2. Runtimes are in seconds. Five runs for each experiment.

Honest Size	Sybil Size	Algorithm	AUC	Accuracy	Precision	Recall	Pretrain Runtime	Runtime
1000	2000	\textsc{SybilRank}	0.718 ± 0.028	0.651 ± 0.021	0.815 ± 0.020	0.616 ± 0.016		0.157 ± 0.003
1000	2000	\textsc{SybilBelief}	0.227 ± 0.004	0.667 ± 0.000	0.667 ± 0.000	1.000 ± 0.000		0.100 ± 0.006
1000	2000	\textsc{SybilSCAR}	0.629 ± 0.007	0.665 ± 0.001	0.667 ± 0.001	0.995 ± 0.005		0.062 ± 0.003
1000	2000	\textsc{SybilGCN-L2}	0.677 ± 0.008	0.361 ± 0.012	0.953 ± 0.023	0.045 ± 0.020		10.990 ± 0.434
1000	2000	\textsc{SybilGCN-L4}	0.470 ± 0.008	0.459 ± 0.026	0.617 ± 0.010	0.490 ± 0.075		18.783 ± 0.794
1000	2000	\textsc{SybilGCN-L8}	0.211 ± 0.008	0.328 ± 0.021	inf ± nan	0.045 ± 0.044		203.438 ± 365.875
1000	2000	\textsc{SybilRGCN-L2}	0.733 ± 0.031	0.368 ± 0.011	0.969 ± 0.014	0.053 ± 0.018		13.215 ± 0.387
1000	2000	\textsc{SybilGAT-L2}	0.695 ± 0.074	0.376 ± 0.018	0.913 ± 0.118	0.083 ± 0.056		4.559 ± 2.114
1000	2000	\textsc{SybilGAT-L4}	0.701 ± 0.020	0.521 ± 0.072	0.875 ± 0.053	0.344 ± 0.166		1.886 ± 0.416
1000	2000	\textsc{SybilGAT-L8}	0.566 ± 0.103	0.473 ± 0.107	inf ± nan	0.282 ± 0.222		2.208 ± 0.515
1000	4000	\textsc{SybilRank}	0.482 ± 0.032	0.485 ± 0.017	0.782 ± 0.017	0.493 ± 0.011		0.281 ± 0.004
1000	4000	\textsc{SybilBelief}	0.074 ± 0.002	0.800 ± 0.000	0.800 ± 0.000	1.000 ± 0.000		0.146 ± 0.002
1000	4000	\textsc{SybilSCAR}	0.566 ± 0.016	0.800 ± 0.000	0.800 ± 0.000	1.000 ± 0.000		0.101 ± 0.002
1000	4000	\textsc{SybilGCN-L2}	0.528 ± 0.008	0.242 ± 0.007	0.884 ± 0.017	0.060 ± 0.012		15.548 ± 1.788
1000	4000	\textsc{SybilGCN-L4}	0.096 ± 0.006	0.246 ± 0.107	0.518 ± 0.149	0.218 ± 0.201		138.445 ± 196.344
1000	4000	\textsc{SybilGCN-L8}	0.066 ± 0.002	0.203 ± 0.003	inf ± nan	0.004 ± 0.006		114.742 ± 124.233
1000	4000	\textsc{SybilRGCN-L2}	0.723 ± 0.018	0.293 ± 0.082	0.981 ± 0.020	0.120 ± 0.109		213.595 ± 395.312
1000	4000	\textsc{SybilGAT-L2}	0.620 ± 0.066	0.477 ± 0.131	0.861 ± 0.048	0.423 ± 0.217		7.202 ± 1.576
1000	4000	\textsc{SybilGAT-L4}	0.613 ± 0.031	0.430 ± 0.051	0.928 ± 0.028	0.314 ± 0.080		480.871 ± 490.873
1000	4000	\textsc{SybilGAT-L8}	0.500 ± 0.000	0.200 ± 0.000	inf ± nan	0.000 ± 0.000		873.387 ± 171.329
1000	6000	\textsc{SybilRank}	0.431 ± 0.024	0.462 ± 0.012	0.818 ± 0.011	0.479 ± 0.008		0.399 ± 0.008
1000	6000	\textsc{SybilBelief}	0.038 ± 0.001	0.857 ± 0.000	0.857 ± 0.000	1.000 ± 0.000		0.204 ± 0.013
1000	6000	\textsc{SybilSCAR}	0.533 ± 0.012	0.857 ± 0.000	0.857 ± 0.000	1.000 ± 0.000		0.293 ± 0.286
1000	6000	\textsc{SybilGCN-L2}	0.405 ± 0.008	0.199 ± 0.019	0.790 ± 0.025	0.092 ± 0.035		20.063 ± 1.122
1000	6000	\textsc{SybilGCN-L4}	0.039 ± 0.001	0.142 ± 0.004	0.676 ± 0.238	0.014 ± 0.010		314.026 ± 389.133
1000	6000	\textsc{SybilGCN-L8}	0.035 ± 0.001	0.144 ± 0.002	inf ± nan	0.001 ± 0.002		161.785 ± 199.593
1000	6000	\textsc{SybilRGCN-L2}	0.706 ± 0.037	0.264 ± 0.074	0.986 ± 0.017	0.145 ± 0.092		21.384 ± 3.542
1000	6000	\textsc{SybilGAT-L2}	0.561 ± 0.141	0.506 ± 0.212	0.846 ± 0.064	0.499 ± 0.286		223.217 ± 401.489
1000	6000	\textsc{SybilGAT-L4}	0.441 ± 0.135	0.357 ± 0.156	0.882 ± 0.068	0.295 ± 0.226		519.030 ± 489.979
1000	6000	\textsc{SybilGAT-L8}	0.449 ± 0.065	0.218 ± 0.096	inf ± nan	0.114 ± 0.144		1090.575 ± 930.070
2000	1000	\textsc{SybilRank}	0.625 ± 0.032	0.583 ± 0.025	0.418 ± 0.025	0.639 ± 0.038		0.168 ± 0.005
2000	1000	\textsc{SybilBelief}	0.226 ± 0.003	0.667 ± 0.000	inf ± nan	0.000 ± 0.000		0.100 ± 0.007
2000	1000	\textsc{SybilSCAR}	0.633 ± 0.017	0.665 ± 0.003	0.204 ± 0.194	0.002 ± 0.002		0.311 ± 0.493
2000	1000	\textsc{SybilGCN-L2}	0.681 ± 0.006	0.671 ± 0.003	0.653 ± 0.108	0.027 ± 0.021		11.797 ± 0.923
2000	1000	\textsc{SybilGCN-L4}	0.471 ± 0.010	0.510 ± 0.063	0.270 ± 0.009	0.272 ± 0.106		19.674 ± 0.949
2000	1000	\textsc{SybilGCN-L8}	0.214 ± 0.005	0.460 ± 0.169	inf ± nan	0.494 ± 0.410		38.789 ± 21.532
2000	1000	\textsc{SybilRGCN-L2}	0.725 ± 0.023	0.667 ± 0.002	0.639 ± 0.329	0.008 ± 0.008		13.611 ± 0.912
2000	1000	\textsc{SybilGAT-L2}	0.719 ± 0.054	0.667 ± 0.000	inf ± nan	0.000 ± 0.001		4.429 ± 2.112
2000	1000	\textsc{SybilGAT-L4}	0.717 ± 0.010	0.631 ± 0.066	0.480 ± 0.052	0.318 ± 0.332		1.610 ± 0.158
2000	1000	\textsc{SybilGAT-L8}	0.669 ± 0.024	0.609 ± 0.027	0.439 ± 0.020	0.603 ± 0.101		2.067 ± 0.193
4000	1000	\textsc{SybilRank}	0.454 ± 0.021	0.463 ± 0.017	0.168 ± 0.017	0.428 ± 0.042		0.324 ± 0.041
4000	1000	\textsc{SybilBelief}	0.074 ± 0.002	0.800 ± 0.000	inf ± nan	0.000 ± 0.000		0.162 ± 0.009
4000	1000	\textsc{SybilSCAR}	0.580 ± 0.028	0.800 ± 0.000	inf ± nan	0.000 ± 0.000		0.124 ± 0.016
4000	1000	\textsc{SybilGCN-L2}	0.529 ± 0.006	0.713 ± 0.172	inf ± nan	0.141 ± 0.277		17.598 ± 2.785
4000	1000	\textsc{SybilGCN-L4}	0.099 ± 0.007	0.364 ± 0.249	inf ± nan	0.159 ± 0.186		30.461 ± 4.482
4000	1000	\textsc{SybilGCN-L8}	0.066 ± 0.001	0.558 ± 0.297	inf ± nan	0.331 ± 0.406		50.558 ± 4.253
4000	1000	\textsc{SybilRGCN-L2}	0.713 ± 0.027	0.798 ± 0.001	inf ± nan	0.001 ± 0.001		18.308 ± 3.516
4000	1000	\textsc{SybilGAT-L2}	0.710 ± 0.031	0.800 ± 0.000	inf ± nan	0.000 ± 0.000		18.654 ± 11.795
4000	1000	\textsc{SybilGAT-L4}	0.551 ± 0.117	0.629 ± 0.183	0.150 ± 0.055	0.235 ± 0.294		126.779 ± 37.142
4000	1000	\textsc{SybilGAT-L8}	0.572 ± 0.014	0.543 ± 0.029	0.221 ± 0.018	0.520 ± 0.116		185.012 ± 44.434
6000	1000	\textsc{SybilRank}	0.421 ± 0.017	0.449 ± 0.007	0.099 ± 0.007	0.355 ± 0.026		0.426 ± 0.013
6000	1000	\textsc{SybilBelief}	0.036 ± 0.000	0.857 ± 0.000	inf ± nan	0.000 ± 0.000		0.213 ± 0.012
6000	1000	\textsc{SybilSCAR}	0.534 ± 0.015	0.857 ± 0.000	inf ± nan	0.000 ± 0.000		0.157 ± 0.009
6000	1000	\textsc{SybilGCN-L2}	0.409 ± 0.007	0.793 ± 0.123	0.023 ± 0.029	0.035 ± 0.068		20.824 ± 1.886
6000	1000	\textsc{SybilGCN-L4}	0.039 ± 0.001	0.285 ± 0.286	inf ± nan	0.716 ± 0.363		41.518 ± 9.597
6000	1000	\textsc{SybilGCN-L8}	0.034 ± 0.000	0.288 ± 0.285	inf ± nan	0.729 ± 0.368		75.919 ± 24.526
6000	1000	\textsc{SybilRGCN-L2}	0.708 ± 0.039	0.855 ± 0.003	inf ± nan	0.000 ± 0.000		21.631 ± 2.295
6000	1000	\textsc{SybilGAT-L2}	0.647 ± 0.032	0.785 ± 0.145	inf ± nan	0.133 ± 0.266		30.193 ± 23.962
6000	1000	\textsc{SybilGAT-L4}	0.466 ± 0.105	0.505 ± 0.228	inf ± nan	0.471 ± 0.362		157.012 ± 38.632
6000	1000	\textsc{SybilGAT-L8}	0.519 ± 0.046	0.485 ± 0.076	0.147 ± 0.036	0.588 ± 0.242		260.332 ± 113.108