DETECTING MULTIPLE EPIDEMIC SOURCES IN NETWORK EPIDEMICS USING GRAPH NEURAL NETWORKS

Rodrigo Gonçalves Haddad

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Daniel Ratton Figueiredo

Rio de Janeiro
Maio de 2023

DETECTING MULTIPLE EPIDEMIC SOURCES IN NETWORK EPIDEMICS USING GRAPH NEURAL NETWORKS

Rodrigo Gonçalves Haddad

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientador: Daniel Ratton Figueiredo

RIO DE JANEIRO, RJ – BRASIL
MAIO DE 2023

Gonçalves Haddad, Rodrigo

Detecting Multiple Epidemic Sources in Network Epidemics using Graph Neural Networks/Rodrigo Gonçalves Haddad. – Rio de Janeiro: UFRJ/COPPE, 2023.

XIV, 60 p.: il.; 29, 7cm.

Orientador: Daniel Ratton Figueiredo

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2023.

Referências Bibliográficas: p. 48 – 52.

1. Graph Neural Network. 2. Network Epidemic. 3. Graph Embedding. I. Ratton Figueiredo, Daniel. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

A minha mãe por me
proporcionar todas as
ferramentas para que eu pudesse
realizar meus sonhos.

Agradecimentos

Primeiramente, expresso minha gratidão por tudo o que meu pais fizeram por mim; o seu apoio e encorajamento me moldaram na pessoa que sou hoje. Mãe, o seu amor e dedicação durante minha criação foram a base da minha vida e pavimentaram minha jornada. Você sempre se fez presente como porto seguro e sou grato a isso.

Aos meus avós, sou grato por ter contato com sua sabedoria e lições de vida. Desde pequeno, sempre pude contar com seu apoio mesmo com a distância. Vocês têm sido fonte de inspiração e acolhimento e sou afortunado pelo privilégio de tê-los comigo.

Aos meus amigos e namorada, seu apoio, conselhos e incentivos significam muito para mim. Nossos momentos de descontração e desabafo foram valiosos para ajudar a me recompor durante as dificuldades.

A todos os professores que tive contato desde os primeiros dias de escola, sou grato pela dedicação, paciência e ensinamentos. Obrigado por serem parte de uma classe tão importante na nossa formação.

Ao meu orientador, obrigado pela seus ensinamentos e acompanhamento ao longo do mestrado. Seu apreço pela pesquisa é inspirador para mim e suas contribuições me ajudaram a aprimorar minhas habilidades e a me tornar um melhor profissional.

A Universidade Federal do Rio de Janeiro, sou grato pela educação, oportunidades e desafios proporcionados a mim. A sua excelência no ensino e pesquisa me preparam para os desafios da vida e guardarei com agrado as memórias das experiências que vivenciei na instituição desde a graduação.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

DETECÇÃO DE MÚLTIPLAS FONTES DE EPIDEMIA EM REDES EPIDÊMICAS UTILIZANDO REDES NEURAIS EM GRAFOS

Rodrigo Gonçalves Haddad

Maio/2023

Orientador: Daniel Ratton Figueiredo

Programa: Engenharia de Sistemas e Computação

Grafos são abstrações matemáticas usadas para representar entidades e relacionamentos entre eles. Um dos processos mais importantes que podem ser modelados por meio de grafos são as epidemias em redes. Nas epidemias em redes, as informações se espalham pela rede e eventualmente atingem os nós localizados longe de seu local de origem. Em muitos cenários, o processo de disseminação começa em um único ou em um número relativamente pequeno de nós, conhecidos como fontes epidêmicas. Identificar o nó fonte (ou vários nós fonte) após o desdobramento de uma epidemia é um problema fundamental, pois revela sua origem. Este trabalho trata da detecção de fontes epidêmicas em redes artificiais e do mundo real de diferentes tipos e tamanhos com nós suscetíveis e infectados. É proposto um algoritmo para aprimoramento de atributos de nós seguido pelo treinamento de redes neurais de grafos. O modelo treinado é então aplicado a novos padrões de infecção e estruturas de rede para encontrar os nós responsáveis por iniciar a epidemia.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

DETECTING MULTIPLE EPIDEMIC SOURCES IN NETWORK EPIDEMICS USING GRAPH NEURAL NETWORKS

Rodrigo Gonçalves Haddad

May/2023

Advisor: Daniel Ratton Figueiredo

Department: Systems Engineering and Computer Science

Graphs are mathematical abstractions used to represent entities and relationships between them. One of the most important process that can be modeled through graphs is network epidemics. In network epidemics, information spreads throughout the network and eventually reaches nodes located far from its beginning place. In many scenarios, the dissemination process starts on a single or a relative small number of network nodes, known as epidemic sources. Identifying the source node (or multiple source nodes) after an epidemic has unfolded is a fundamental problem, since it reveals its origin. This work deals with the detection of epidemic's sources problem in artificial and real world networks from different types and sizes with susceptible and infected nodes. It is proposed an algorithm for node attribute enhancement followed by graph neural network training. The trained model is then applied to new infection patterns and network structures to find the nodes responsible for starting the epidemic.

Contents

List of Figures

List of Tables

List of Abbreviations

BA	Barabási-Albert, p. 28, 30, 33, 36, 46
BFS	Breadth First Search, p. 21, 23
DRI	Depth Ring Infection, p. 21, 23, 24
ER	Erdős–Rényi, p. 31, 46
GCNSI	Graph Convolutional Networks based Source Identification, p. 19, 20, 44
GCN	Graph Convolutional Network, p. 5, 9, 10, 12, 19–21, 26, 43, 44
GNN	Graph Neural Network, p. 4, 6–9, 12, 46, 47
IC	Independent Cascade, p. 16, 20, 28, 47
LPSI	Label Propagation based Source Identification, p. 18, 19, 44
LSTM	Long Story Short Memory, p. 12
LT	Linear Threshold, p. 16
MDL	Minimum Description Length, p. 18
MSD	Multiple Source Detection, p. 19, 32, 36, 44, 47
RI	Ring Infection, p. 21, 23, 24
SD	Source Detection, p. 2
SIR	Susceptible-Infected-Recovered, p. 15, 16, 18, 20, 28, 47
SIS	Susceptible-Infected-Susceptible, p. 16, 47
SI	Susceptible-Infected, p. 15, 18, 20, 28, 46, 47

Chapter 1

Introduction

Due to the number of parties involved and the nature of their connection, relationships between individuals can be quite complex, lacking regularity or simple rules that can be difficult to model effectively. For that matter, graphs are mathematical abstractions used to represent relationships between entities and provide a starting point for mathematical analysis. The entities involved are represented by nodes and edges represent relationships between entities. Nodes are used to represent people, objects, geographic points, human cells and more. Edges are used to depict a variety of relations such as interactions between cells in an organism, interactions among people in a social network, submarine internet cable connections between countries among others.

Among various processes that take place on graphs, the spread of information is among the most important ones. Information can propagate through the network reaching nodes far away from its starting point after some time. The propagation takes form in natural and human related events that happen inside networks such as social networks (rumors and false information spread), medical networks (contagious diseases spread) and financial networks (financial failures) [11]. However, the spread of information does not necessarily has a negative connotation; more frequently, non-harmful information propagate by files, news and advertisements. Indeed, different sorts of information travel through various networks in various ways.

In many scenarios, the dissemination process starts on a single or a very small number of network nodes, known as the epidemic source. Thus, identifying the source node (or multiple source nodes) after an epidemic has unfolded is a fundamental problem, since it reveals the origin of the epidemic. In spite of that, identifying the source nodes is not an easy task and the difficulty lies on how much the epidemic has advanced in the network after the first contagion. Usually, a single snapshot of the network after the information has spread is available, making it difficult to identify the initial infected nodes, then the more advanced the diffusion, the harder it is to identify where it started. Network structure namely its degree distribution and diameter play a fundamental role on how information travels.

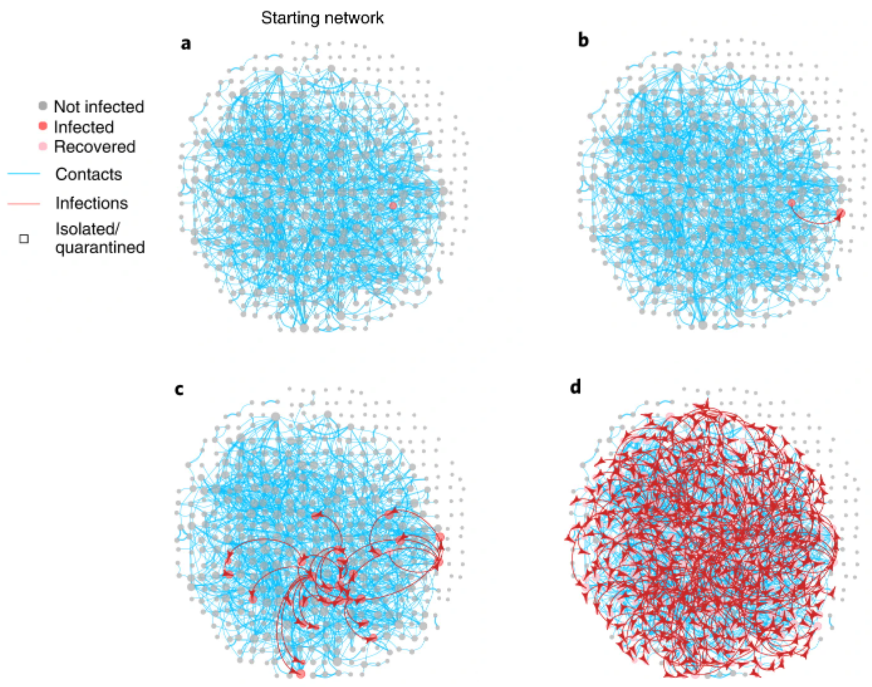


Figure 1.1: Advance of SARS-CoV-2 in a group of individuals from Haslemere-UK. From **a-b** 10 days have passed, **b-c** 20 days have passed and **c-d** 70 days have passed. The social network contains 468 nodes and 1257 edges weighted by the contact strength. Extracted and adapted from [1].

For over a decade, efforts have been made to address the problem of identifying the source of an epidemic. The pioneering methods started by classifying single sources in networks that would contain only susceptible and infected nodes [12], which evolved to multiple source detection on different diffusion patterns [13]. They range from maximizing likelihood while trying to find nodes with great probability of being sources [13] to iteratively propagate node labels to accumulate values from its neighbors [14]. More recently, learning-based approaches began being used to tackle the problem. In this case, the model is bound to the same graph structure which the model was trained with and learns to identify the source of an epidemic by creating latent representations for each node and classifying them as sources or non-sources [10].

Besides the existence of said algorithms, the SD (Source Detection) problem remains being challenging mainly when the epidemic starts in multiple nodes in a large network. This thesis discusses the implementation of a learning-based method which is not structure bound, i.e., it is capable of tackling the problem for unseen networks and different contagions.

1.1 Motivation

Numerous types of epidemics ranging from viral spread to information dissemination have occurred throughout human history with severe impact for mankind. The most recent was the COVID-19 wave; Figure 1.1 depicts a simulated advance of COVID-19 in a group of individuals from Haslemere-UK. This simulation shows how quickly an epidemic that started on a single node is capable of spreading when no control scenarios, such as quarantine and isolation, are taken. Network shown in Figure 1.1**(d)** is the average of 75% of individuals infected after several simulations, whereas more controlled scenario would reduced it to 16%. Also, while the COVID-19 virus was physically spreading among humans, rumours and *fakenews* were being disseminated through digital platforms. Overall, the study of epidemics allows predicting the impact of an epidemic or evaluate the effectiveness of control measures. The spread of information between individuals in a network can be approximated by mathematical models. There are numerous studies about different types of epidemic models that try to simulate different kinds of diffusion; a model that nicely captures the characteristics of a virus spread may not be the best to capture the diffusion of rumours in a digital social network.

An important issue that naturally arises when considering epidemics is to find where it started. In a network, this is equivalent to finding the node (or group of nodes) responsible for the first contagion. This problem has real life applications e.g., which account started a false message trend or in which computer a virus first entered a corporate network.

One key factor is the network observed after the epidemic has started and what information is provided in this observation. If the infection time of every node is available, then identifying the source node is trivial. However, if no time data is available and only the set of infected nodes is observed at a certain point in time, then identifying the source node is much more challenging.

1.2 Problem Definition

Let $\mathcal{G} = (V, E)$ represent a connected undirected graph containing a set of vertices V and a set of edges E and let Y in V denote the set of source nodes responsible for initiating the spread of information.

Consider an epidemic model that infects uninfected nodes from the V set. After a percentage of the graph is infected a snapshot of the graph denoted by $\mathcal{O}$ is taken. $\mathcal{O}$ contains the same structure of $\mathcal{G}$ such that $\mathcal{O} = (V, E, I)$, where I is the set of infected nodes, $I \subseteq V$.

The goal is to identify Y from $\mathcal{O}$ by designing a function F that returns the set of nodes where the epidemic started $Y^* = F(\mathcal{O})$. As the ground truth is denoted by Y, the quality of the function F can be measured by the Jaccard similarity between the two sets Y^* and Y, which is given by:

$$q(F, \mathcal{O}) = \frac{|Y^* \cap Y|}{|Y^* \cup Y|} \tag{1.1}$$

Observe that when $q(F, \mathcal{O})$ depends also on the observation $\mathcal{O}$ and $q(F, \mathcal{O}) = 1$ only when the exact set of source nodes is identified by F.

1.3 Objective and Contribution

This thesis addresses the problem of identifying multiple epidemic sources in different types of networks under different observations (infection percentage) of the epidemic. The main contributions are:

- The design and implementation of a framework using Graph Convolutional Networks to identify the set of source nodes of an epidemic. The framework is inductive and a trained model can be used in different unseen networks.
- Novel attributes for node features that reflect their local epidemic state, taking into consideration neighborhoods at different distances.
- Empirical evaluation of the proposed framework on real networks and synthetic network models along with an in depth discussion of the model performance while varying the infection snapshot and number of source nodes. Curiously, results indicate that the effectiveness of the framework improves as the number of sources increases, despite the probable appearance of infection clusters that overlap.

1.4 Structure

The following chapters are organized as follows:

- Chapter 2 presents the necessary background and related work. In particular, epidemic models are introduced as well as algorithms used to generate node embedding including a discussion about Graph Neural Network.
- In Chapter 3, it is discussed the novel metrics implemented to generate attributes for nodes from the observation of a network with infected nodes. This attributes are an important part of GNN application.

- In Chapter 4, it is presented the technical implementation of infection propagation and GCN construction. It is further described metrics and baselines used to compare to other works.
- Chapter 5 presents an empirical evaluation of the proposed framework under various conditions such as different networks, observations, and numbers of sources.
- Chapter 6 presents a conclusion of the work along with a path for future work.

Chapter 2

Preliminaries and Related Work

This chapter presents related work and important concepts and is organized as follows: Section 2.1 presents structural node embeddings which are used to create vector representations for nodes of graphs in Euclidian space. Section 2.2 introduces Graph Neural Networks (GNN) and its derivatives and how it is used to generate graph embeddings leveraging node attributes. Section 2.3 introduces epidemics on networks; the different models used to characterize different epidemics and different models for observation after contagion. Section 2.4 discusses approaches that tackle the source identification problem from pioneering works to those considered state-of-art.

2.1 Node Embedding

Graph embedding is the task of generating a vector representation for nodes of the graph with a desired number of dimensions. Embeddings have received a lot of attention because they transform network information in latent data that can be easily employed in mathematical models.

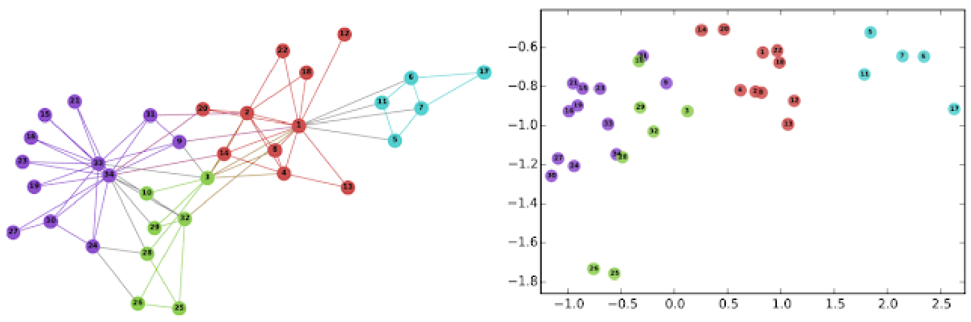


Figure 2.1: On the left, the well known Karate graph representing a social network. On the right, a continuous two dimensional space embedding representation of the nodes using DeepWalk. Extracted from [2].

Algorithms that generate embeddings take different features in consideration:

structural embeddings take into account nodes neighbors; while GNN approaches leverage node and edge attributes by using functions to aggregate structural and attribute data.

2.1.1 Structural node embedding

The technique based on network structure generates a representation (embedding) on d-dimensions leveraging primarily structural network characteristics. Because it takes into consideration structural data, neighboring nodes will likely have similar representation while distant nodes will differ. Some of the more prominent algorithms to generate embedding are discusses below.

Word2vec [15] is a natural language processing algorithm that is not directly connected to graph embedding. It aims to generate an embedding for every word so it is possible to predict words inside the same context. At first glance, *word2vec* seems useful only when dealing with text documents, however researchers were able to establish a parallel for networks. The same way a piece of text is an ordered sequence of words, one could apply random walks to sample nodes from a graph extracting an ordered sequence of nodes. For that reason, *word2vec* inspired a series of articles concerning structural embeddings.

DeepWalk [16] learns latent representations sampling neighborhoods for every node in the graph by drawing random walks of certain size starting on the target node. *DeepWalk* aims to provide a social representation of a graph's vertex as it captures neighborhood similarity and community membership. After sampling, the node sequence is used as input to a SkipGram algorithm. As SkipGram [17] maximizes the co-occurrence probability among the words that appear within the same window, nodes close to each other will likely have a similar representation and thus embedding. Figure 2.1 represents a embedding generated by DeepWalk. *Node2vec* [18] is a semi-supervised algorithm for scalable feature learning in networks and is based on DeepWalk. However, *node2vec* proposes a biased second order random walk aiming at transitioning between DFS and BFS when walking the graph by introducing two new parameters (called return and in-out) that control how fast the walk explores and leaves the starting node neighborhood. *DeepWalk* is described by *node2vec* creators as a special case of *node2vec* since similar algorithm can be achieved by adjusting two parameters p (which controls the likelihood of immediately revisiting a node in the walk) and q (which allows the search to differentiate between visiting more frequently further away nodes from closer nodes).

A more recent approach *struct2vec* [19] makes use of structural identity. To reach its objective, it calculates the structural similarity between each vertices pair present in the graph for different neighborhood sizes. Next, a multi-layered weighted graph is constructed where layers represent the k-distance neighborhood. A biased random walk is used to generate node sequences in the multi-layered graph (the sequences are likely to have nodes structurally similar). Node sequence is then used as input to a learning technique such as SkipGram, as *node2vec*.

Another algorithm example, *subgraph2vec* [20] learns latent representations of rooted subgraphs from large graphs. First, it generates rooted subgraphs around every node in a given graph applying Weisfeiler-Lehman kernel. After the subgraph is extracted, it is used as input to a radial SkipGram model which differs from the vanilla algorithm by learning the embedding of a target subgraph using its surrounding radial context.

Noticeable, random walks are largely used to generate random sequences of nodes of a graph while also reducing memory usage when extracting random samples of large graphs (i.e., not all paths are considered). Also, structural embedding, as the name suggests, does not seek to leverage node attributes when learning latent representations leaving room for other algorithms that do.

2.2 Graph Neural Networks

Graph Neural Networks (GNN) were created to extend existing neural network methods for processing data in graph domains by mapping nodes into Euclidean space applying supervised or unsupervised learning algorithms [21]. Bronstein *et al.*[3] classifies GNN as one of the most general class of deep learning architectures as other deep learning architectures can be interpreted as a special case of GNN. GNN in itself can be divided in two functions: *permutation equivariant* and *permutation invariant.* Equivariant functions are constructed by applying shared invariant functions over local neighborhoods. These local functions are usually referred as "diffusion", "propagation" or "message passing", and the overall computation as a "GNN layer". Most GNN layers may be derived from three flavours of layers: convolutional, attentional and message-passing. These flavours govern the extent to which invariant functions transforms the neighborhood features. It is worth noting that the presented types are nested as follows: *convolution* $\subseteq$ *attention* $\subseteq$ *message-passing.*

In all of them, permutation invariance is ensured by aggregating transformed features with some permutation invariant function and then updating the features of the target node by means of some function. Usually, the aggregator function is non-parametric and not learnable, as it represents an operation such as sum, mean or maximum.

Figure 2.2 visually presents the differences in each layer of the three types of GNN. **Convolutional** envolves aggregating features of neighborhood nodes with fixed weights (denoted by the constant c_{uv}) that specify the importance of node v to

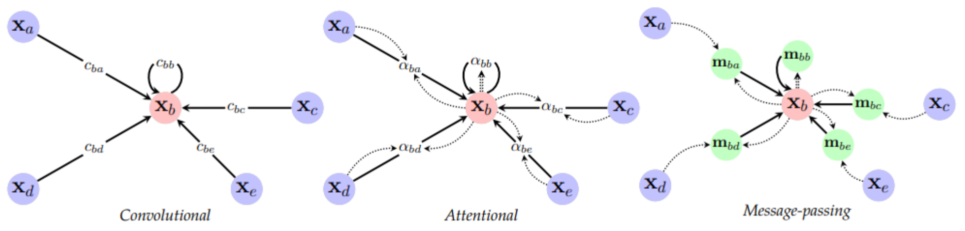


Figure 2.2: A schematic of information flow through GNN's different flavours. Extracted from [3].

node u depending on the adjacency matrix. **Attentional** on the other hand computes the weight in a learnable fashion by $\alpha_{uv} = a(\mathbf{x}_u, \mathbf{x}_v)$ while **message-passing** aggregates vectors generated from neighboring nodes computed by $\mathbf{m}_{uv} = \psi(\mathbf{x}_u, \mathbf{x}_v)$. More emphasis will be given to **message-passing** since it better describes the algorithm used later in this work, which consists on computing arbitrary vector across edges. Described as:

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} \psi(\mathbf{x}_u, \mathbf{x}_v)\right) \tag{2.1}$$

Where a vector message v is sent to node u by its neighbors and aggregated by $\bigoplus$ function and updated by ϕ function. ψ is the learnable message function.

As per Hamilton[6], it is also important to disclose how different node classification settings work during training. *Transductive* nodes are unlabeled nodes which are involved in the message-passing operations and receive GNN hidden representations $\mathbf{h}_v^{(k)}$ but are not used in the loss function computation. Training a GNN on a citation network graph with some unlabeled nodes and then testing it on these unlabeled nodes is an example of transductive node classification.

On the other hand, *inductive* nodes are not used in the GNN message passing operations neither in the loss computation functions. In other words, these nodes and all of its edges are never seen by the GNN model during training. An example of inductive node classification would be training a GNN on one subgraph of a citation network and then testing it on a completely disjoint subgraph of that network.

2.2.1 Graph Convolutional Networks

Graph Convolutional Networks (GCN) were introduced by Kipf *et al.*[4] as a subclass of GNN designed for semi-supervised learning in the transductive setting. This setting required the graph's Laplacian matrix to be fully known beforehand and no structural changes to the network are supported during testing.

Kipf *et al.*[4] proposes a semi-supervised convolutional neural network which operates directly and efficiently on graphs because it scales linearly in the number of edges. In his work, the graph is encoded directly using a neural network model and trained on a supervised target for all nodes with labels, learning hidden layer representation that encode both graph structure and node attributes. Conditioning the model on the adjacency matrix of the graph allows the model to distribute gradient information and enables it to learn representation of nodes with and without labels.

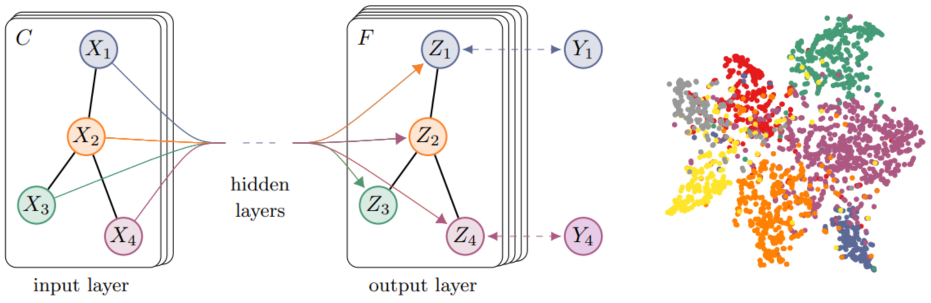


Figure 2.3: Schematic of a multi-layer GCN for supervised learning with C input channels and F feature maps. Labels are denoted by Y_i. Extracted from [4].

Consider a multi-layer GCN with the following propagation rule:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{2.2}$$

$\tilde{A}$ is the adjacency matrix of a undirected graph with added self-connections, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, $W^{(l)}$ is a layer-specific trainable weight matrix, σ denotes an activation function, and $H^{(l)}$ is the matrix of activations in the l^{th} layer.

A neural network model based on graph convolutions can therefore be built by stacking multiple convolutional layers. Considering a two-layer GCN for semi-supervised node with symmetric adjacency matrix A, the forward model takes the form:

$$Z = f(X, A) = \text{softmax}\left(\hat{A}\,\text{ReLU}\left(\hat{A}XW^{(0)}\right)W^{(1)}\right) \tag{2.3}$$

Where $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$. $W^{(0)}$ is an input-to-hidden weight matrix, whose dimensions depend on the number of input channels and hidden layer's feature maps, and $W^{(1)}$ a hidden-to-output weight matrix, whose dimensions depend on hidden layer's feature maps and number of feature maps in the output layer. Note that $W^{(0)}$ and $W^{(1)}$ are the parameters of this model that must be tuned during training.

Further, for multi-class classification, the cross-entropy error over all labeled examples is computed by:

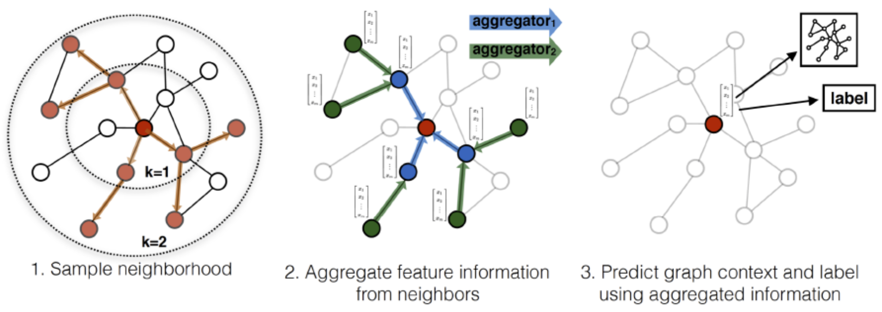
$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf} \tag{2.4}$$

Kipf *et al.*[4] performs batch gradient descent using the full dataset for every training iteration which would be a problem in the case of very large datasets. Alternatively, a memory-efficient approach would adopt mini-batch stochastic gradient descent.

Further research implemented a variant of the algorithm which enabled more memory-efficiency with inductive setting as the previous GCN method relied on the same graph structure being used both in training and in testing, i.e., no new nodes could be added.

2.2.2 GraphSAGE embedding

GraphSAGE [5] is an inductive node embedding algorithm which incorporates nodes features and topological structure in order to learn an embedding function that generalizes embeddings to unseen nodes. The unseen nodes may comprehend new nodes added to the graph structure as well as whole new graphs; one could train an embedding generator on protein interaction network derived from a reference organism, and then produce node embeddings for data collected on different organisms using the trained model.



k=1
k=2
1. Sample neighborhood
aggregator1
aggregator2
2. Aggregate feature information from neighbors
label
3. Predict graph context and label using aggregated information

Figure 2.4: Illustration of GraphSAGE sample, aggregation and prediction. Extracted from [5].

Instead of training an embedding for each specific node which would make the model bound to a certain graph structure, a trainable set of aggregator functions that aggregate information from neighborhoods at different distances from the target node is used.

GraphSAGE generates embeddings by making each node aggregate the representation of the nodes in its immediate neighborhood into a single vector. This step depends on the representation generated at the previous iteration and must have a base case (the initial representation). After aggregating the neighboring feature vectors, GraphSAGE then concatenates the node's current representation with the aggregated neighborhood vector and uses it as the input to a fully connected layer with nonlinear activation functions. The result is the representation that is then used in next iteration.

Hamilton *et al.*[5] describes three types of aggregator functions: **mean, LSTM and pooling aggregators**. LSTM and pooling reportedly outperformed GCN and mean aggregator. However, LSTM is significantly slower than pooling by a factor of two giving pooling aggregator a slight edge.

2.2.2.1 Message Passing

Message passing describes the whole process of how the target node receives information from neighboring nodes, aggregates and updates its own value at each iteration.

The information comes in two forms as specified by Hamilton[6]. One of them is the *structural* information e.g., encoded information about degrees of all the nodes in the k-hop neighborhood. The other kind of information absorbed by GNN node embedding is *feature based*. Equation 2.5 summarizes the whole message passing to generate embedding for a target node u.

$$\mathbf{h}_u^{(k+1)} = \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \text{AGGREGATE}^{(k)}\left(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\}\right)\right) \tag{2.5}$$

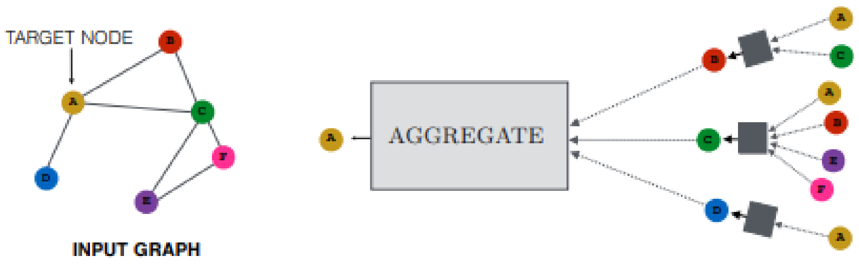


Figure 2.5: Overview of a single node aggregating messages from its local neighborhood via an *aggregate* operator. Note that the message aggregated on the target node also contains its own information from the perspective of its neighborhood. Extracted from [6].

2.2.2.2 Aggregation

In general, the aggregation function aggregates information from other nodes with information. There are different types of aggregation functions and messages; vectors are an example of messages and *mean* is an example of an aggregation function. Figure 2.5 shows the aggregation step in a two-layer version of a message-passing model. The computation graph forms a tree structure by placing the neighborhood around the target node.

Bronstein *et al.*[3] states that the aggregator function is not usually learnable. However, in GraphSAGE, a set of aggregator functions are trained in order to merge feature information from a node's local neighborhood. Since it is computed from the features of nodes, the aggregator function may be applied to new nodes and even whole new graphs. Ideally, the property of symmetry (i.e., invariant to permutation of inputs) of aggregators should be ensured in order to guarantee that the neural network model can be trained and applied to arbitrarily ordered node neighborhood feature sets. The max pooling aggregator is an example of a both symmetric and trainable operator. It is characterized by the following equation:

$$\mathbf{h}^k_{\mathcal{N}(v)} \leftarrow \max\left(\left\{\sigma\left(\mathbf{W}_{\text{pool}}\, \mathbf{h}^{k-1}_u + \mathbf{b}\right), \forall u \in \mathcal{N}(v)\right\}\right) \tag{2.6}$$

Where σ represents a nonlinear activation function and *max* is an element wise operator. For each depth k, each node v aggregates representations for every node in its local neighborhood.

2.2.2.3 Update

Finally, after aggregating the information vectors, aggregation result from the previous step $\mathbf{h}^{k-1}_{\mathcal{N}(u)}$ and the current representation for the node, $\mathbf{h}^{k-1}_u$, are fed through a fully connected layer with nonlinear activation function σ, as follows:

$$\mathbf{h}^k_u \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}\left(\mathbf{h}^{k-1}_u, \mathbf{h}^k_{\mathcal{N}(u)}\right)\right) \tag{2.7}$$

This is then the representation for node u at layer k, and its final representation depends on the total number of layers used in the model. Often, a few layers are used in practice (e.g., two or three).

2.3 Network Epidemics

The study of mathematical models that characterize epidemics is at least two centuries old. One of the oldest registered studies [22] began discussing epidemic models back in 1766. More recently, a series of articles written by Kermack–McKendrick proposed simple compartmental models that are the basis for various modern epidemic models, including network epidemics.

The aforementioned focused on analysing epidemic diseases. Kephart *et al.*[23]

reports that there are compelling analogies between diseases spread and computer viruses spread. This led to their effort of adapting existent diseases spreading mathematical models to computer virus spread. More specifically, Wang *et al.*[24] gathers information about the dissemination of worms, a sub-class of computer viruses, across computers. Worms search for new targets to transmit themselves and have the ability to travel from host to host that are connected by a network. While in the propagation phase, hosts in the network have three different states: susceptible, infectious and removed. A host that is vulnerable to infection is called susceptible; infectious a host that has been infected and can spread the infection to others; removed host is immune or has died and is no longer able to be infected or spread the infection.

Doerr *et al.*[25] discusses why rumours and news spread much faster in existent online social networks than in other network topologies. They support this claim by proliferating information in networks with power law degree distribution that require sub-logarithmic time to spread news to all nodes while in other networks topologies at least logarithmic time is needed. To simulate the epidemic, a *push-pull* model [26] was adopted, where each node randomly communicates with one of its neighbors, excluding the one contacted immediately before.

2.3.1 Propagation Models

David *et al.*[27] proposes a division of the models in two big groups: influence and infection epidemic models. While infection models were used to describe disease contagion among individuals, influence models were adopted to describe "social contagion" because they spread from one individual to another similar to a biological epidemic. Social and biological contagion have fundamentally different underlying mechanisms; whereas biological contagion is based on the possibility of contracting a disease-causing pathogen through contact with another peer, social contagion typically involves decision-making on the part of the affected individuals. However, the dynamics at the network level are comparable, and knowledge gathered from the study of biological epidemics can be generalized when considering how diffusion occurs on networks.

Starting from 1927, a series of three articles published the basis of modern epidemic modelling. Kermack *et al.*[28] theorized a partial differential-equation model that structured the infected population in terms of time, while using the now popular labels susceptible (S), infected/sick (I), and recovered/removed (R) [28]. In 1932, a second article by the same authors was published to extend the model and remove previous limitations [29]. They also added the effect of partial immunity and fresh susceptible individuals to the system by birth and immigration. In 1933, minor improvements concerning new death rates to accommodate deaths by other causes were included [30].

On the *Susceptible-Infected* (SI) model, nodes are initially susceptible and can be infected. Once a susceptible node is infected, it never recovers. At the beginning of the epidemic, considering $t = 0$, only one individual, or a set of individuals, are infected nodes which are also the set of epidemic source nodes.

At each time slot, an infected node v tries to infect each one of its neighbors w with probability $p_{v,w}$. In the most basic form of this model, every infected node has the same probability of infecting a susceptible node, therefore, $p_{v,w} = p$. Supposing w is surrounded by n infected nodes, the probability of w not being infected in the current time step is $p_{inf} = (1 - p)^n$. Further, if the infection fails, the probability remains the same in the next time step, given that no new nodes adjacent to w were infected. Also, infected nodes try to infect all their susceptible adjacent nodes at each time step making it possible that more than one new infection happens at time step.

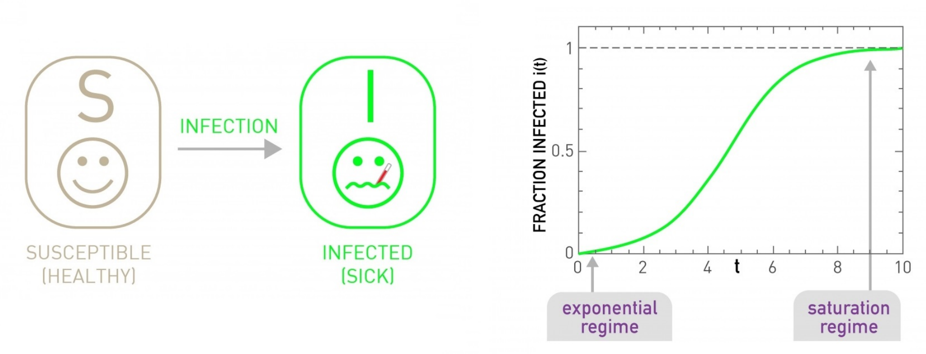


Figure 2.6: States of SI model followed by graph representation of the fraction of infected nodes varying with time. Extracted from [7].

Figure 2.6 plots the fraction of infected nodes which follows the pattern of a logistic growth curve. It shows rapid contagion growth in the first half as exponential regime followed by saturation regime when most of the network is already infected.

The *Susceptible-Infected-Recovered* (SIR) model is similar to SI, the difference resides on the Recovered state; at each time step, infected node v may recover with probability q. Once recovered, it can not infect other nodes and neither be infected again. Figure 2.7 shows the evolution of the fraction of nodes in each category; in this example all nodes were infected and will thus recover.

Figure 2.8 shows that different compartmental models follow similar trend in the exponential regime; when the number of affected individuals is small, the disease spreads freely and the number of affected individuals increases exponentially. Nevertheless, after some time the difference between the models becomes clear: in

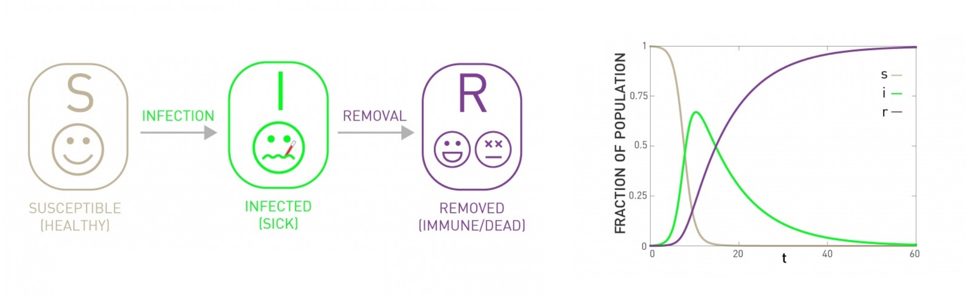


Figure 2.7: States of SIR model followed by graph representation of the fraction of the population of each state varying with time. Extracted from [7].

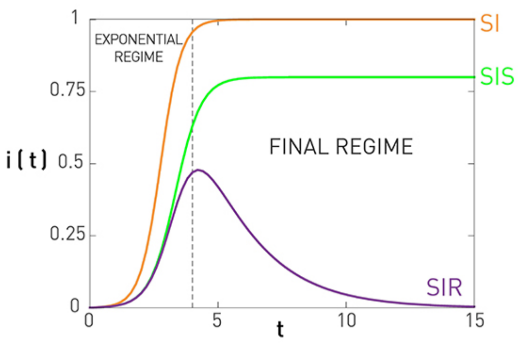


Figure 2.8: Comparing SI, SIS and SIR infection rate; their biggest difference reside in the saturation regime. Extracted from [7].

the SI model every individual becomes infected, in the SIS model a fraction of individuals remains infected (which depends on model parameters), in the SIR model the endemic states becomes dominated by recovered nodes. In the SIR model, the epidemic dies out when the basic reproductive number is smaller than 1 (a number that depends on the recovery rate and infection rate).

Influence models such as *Independent Cascade* (IC) and *Linear Threshold* (LT) are other models for contagion. On IC [31] an initial set of active nodes unfolds the process in discrete time steps according to the following rule: when node v first becomes active (infected) in step t, it is given a single chance to activate (infect) each currently inactive neighbor w; it succeeds with probability $p_{v,w}$. The probability may be different for each neighbor connection, or the same for all the connections inside the network. If w has multiple newly activated neighbors, each attempts to active w in random order within the same time step. If the activation succeeds, w will become an active node in the next step. If it does not succeed, v cannot make any more attempts at activating w, while w can still be activated when other non-active neighbors become active.

2.3.2 Categories of observations

In the source identification problem, one must determine the source nodes of an epidemic given some data that is observed from the epidemic process. Thus, a fundamental premise is the observation state of nodes during the propagation process [8]. Since each type of observation changes the problem, it is important to establish the observation category model under consideration.

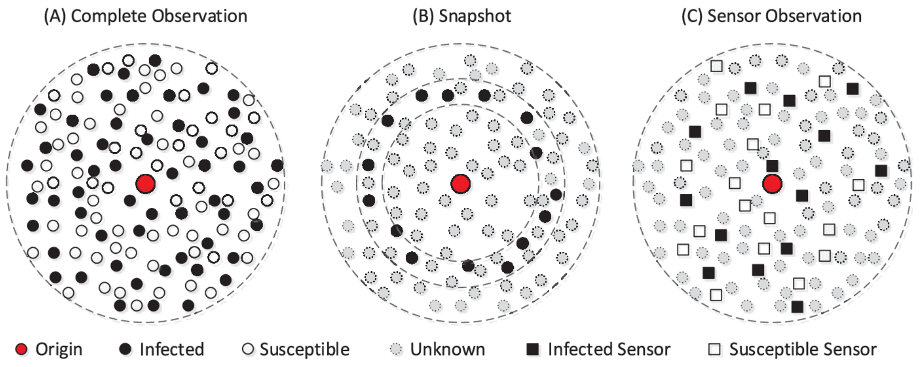


Figure 2.9: Illustration of three categories of observation in networks. Extracted from [8].

The *complete observation* reports the epidemic state of all nodes in the population distinguishing susceptible and recovered (if applicable) at time t. An example of complete observation is shown in Figure 2.9(A).

Snapshots provide partial knowledge of the sate of nodes at a given time t and are presented in three forms that might intersect: **(i)** each node reveals its epidemic state with probability p, **(ii)** only a fraction of nodes are known i.e., they are reportedly infected or not **(iii)** only a set of nodes are observed when taking the snapshot. An example of snapshot is presented in Figure 2.9(B).

Alternatively *sensor observations* are nodes that collect information about their state, state transitioning time and infection directions. This representation is the richest one in terms of information (since the time of infection is available), although it is not feasible in most real world network applications because it depends on the implementation of sensors before an epidemic starts. An example of complete observation is shown in Figure 2.9(C).

The proposed framework assumes the input is a complete observation model but does not require any additional information concerning the epidemic propagation model.

2.4 Epidemic Source Detection

Epidemic source detection is the problem of identifying the source of an epidemic given an observation model. Different approaches have been applied to this problem, from using entropy related ideas [32] to neural networks [10] and label propagation [14]. Some of them are used to detect sources when a specific type of epidemic model is used, while others are used to detect sources without any information about the underlying epidemic model. The latter is specially useful in real world applications where it may not be feasible to determine the model and its parameters value in advance.

Shah *et al.*[12] provides a rigorous study on finding the single source of rumor in a network affected by a SI epidemic. It involves finding the node with the maximum likelihood of being the source, and thus assumes knowledge of the epidemic model. The same authors continue their research working on a similar but evolved problem, taking into consideration “how many” and “which” nodes caused the epidemic called *NetSleuth* [13]. *NetSleuth* is a well-known Susceptible-Infected source detector: the model starts by applying a submatrix-laplacian method to find out the best seed sets given a certain number of seeds. The idea is that the nodes on the edge of the infected snapshot are unlikely to be the sources due to the large number of uninfected nodes surrounding them. After that, given the seed-sets, an algorithm computes the Minimum Description Length (MDL) scores.

MDL states that given a set of models, the best model is the one that minimizes the sum of length in bits of the model description and the length of the description of the data encoded with the model. MDL is employed to identify the set of sources and the propagation ripple that starting from those nodes would better describe the given snapshot (input to the problem).

Wang *et al.*[14] applies the idea of *source prominence* through a detection method called LPSI (Label Propagation based Source Identification). First, positive labels $(+1)$ are assigned to the infected nodes while negative labels (-1) are assigned to the uninfected ones. Before iterating, a weight matrix is built to represent the propagation probability between nodes (when neighbors). During the iteration phase, each node receives a fraction of label information from its neighborhood while retaining some of its previous label information. Finally, the node is identified as source if it satisfies two conditions: it was initially infected and its final label value is bigger than those of its neighbors. See Figure 2.10 for a visual representation of the aforementioned framework.

Zang *et al.*[9] addresses the problem for networks under SIR epidemic model by applying the *divide and conquer* approach. The first part of the algorithm aims to solve the problem of identifying recovered nodes that are not explicitly observed in this problem formulation (only infected nodes are observed correctly). The score based reverse propagation assigns 1 for infected observed nodes and fills the remaining with zeros. Similar to LPSI, for every iteration, nodes are assigned the score sum from its neighbors. After N iterations, nodes score are compared to a pre-defined

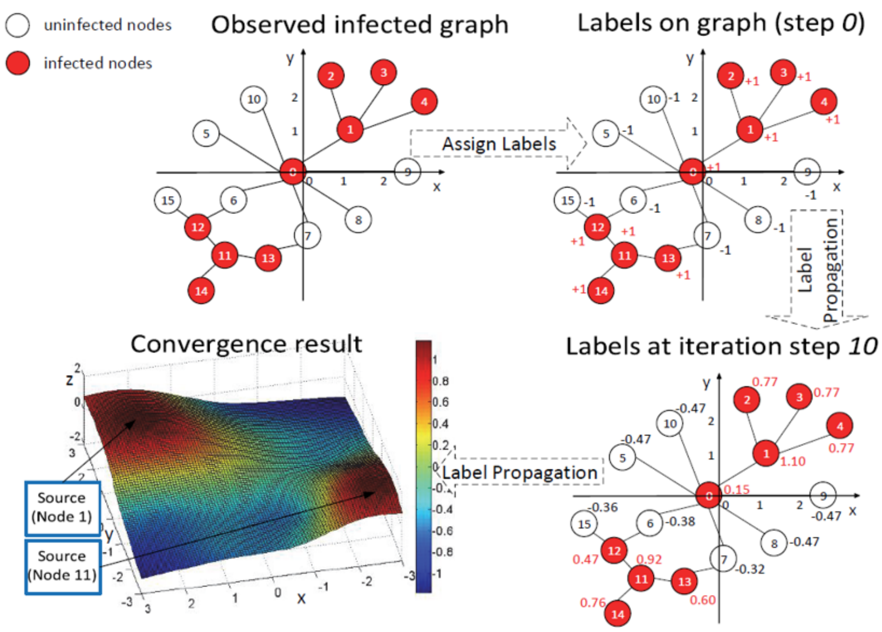


Figure 2.10: LPSI's framework consists on label propagation till convergence and posterior label value analysis. Extracted from [9]

base score. If the node score is greater than the base score, the node is added as part of what is called the "extended infected node set" which comprehends infected nodes and nodes that are suspect of being recovered. Once this process finishes, the extended infected network is partitioned using a leading eigenvector based partition solution.

Also, a simple heuristic for estimating the number of sources is proposed. For each number of sources k from 1 to K the modularity value is computed for the extended infected network: if there is a significantly increase in its value, the new k is preferred. Finally, a variation of the betweenness centrality is used to rank the nodes of each partition and determine the source node.

MSD (Multiple Sources Detection) problem is known to be a supervised learning problem and therefore the model can be trained with known sources. Dong *et al.*[10] proposes a modified GCN that is used as supervised model to detect sources.

In the approach *Graph Convolutional Networks based Source Identification* (GCNSI)[10], every node is assigned a four element vector as features. The first element comes from LPSI calculation and the remaining three come from superimposing vectors. After adding this information to the nodes, the graph is given as input to the GCN model where the batch-sized loss between input and output is calculated to update the weight-matrix when a batch of samples is acquired. The sigmoid cross-entropy is adopted as the loss function for this model, and is given by:

$$L(y', y) = -\log \sigma(y') \times y - \log(1 - \sigma(y')) \times (1 - y) + \lambda ||w||_2 \tag{2.8}$$

Where y' is the output of the model, y is the true label value and σ is the sigmoid function. Observe that w represents the weights in the GCNSI and $||w||_2$ is the L2 regularization with a coefficient λ. This loss function is similar to the one applied in this work which will be later discussed.

GCNSI [10] is trained with samples generated by SI, SIR and IC epidemic models in equal proportion and multiple types of networks are used. It is worth noting that for each type of network, a different configuration of GCN is used in order to achieve better results. Thus, the model is bound to the network structure used during training; it is not possible to reuse it to indicate sources in a network structure different from training.

An algorithm based on *entropy* proposed by Liu *et al.*[32] for the SI epidemic model claims that it would also be successfully applied on other models, such as SIR, when the complete observation is present. The approach indicates the possibility of a node being the source based on its neighborhood entropy. The greater the entropy, the more infection information the node carries in the network, and thus the higher the chance of being the source. The algorithm is composed by two parts: one with the lowest computational complexity but lacking more rigorous foundations and the other which improves source identification recall at the cost of increased computational complexity.

Chapter 3

Proposed Framework

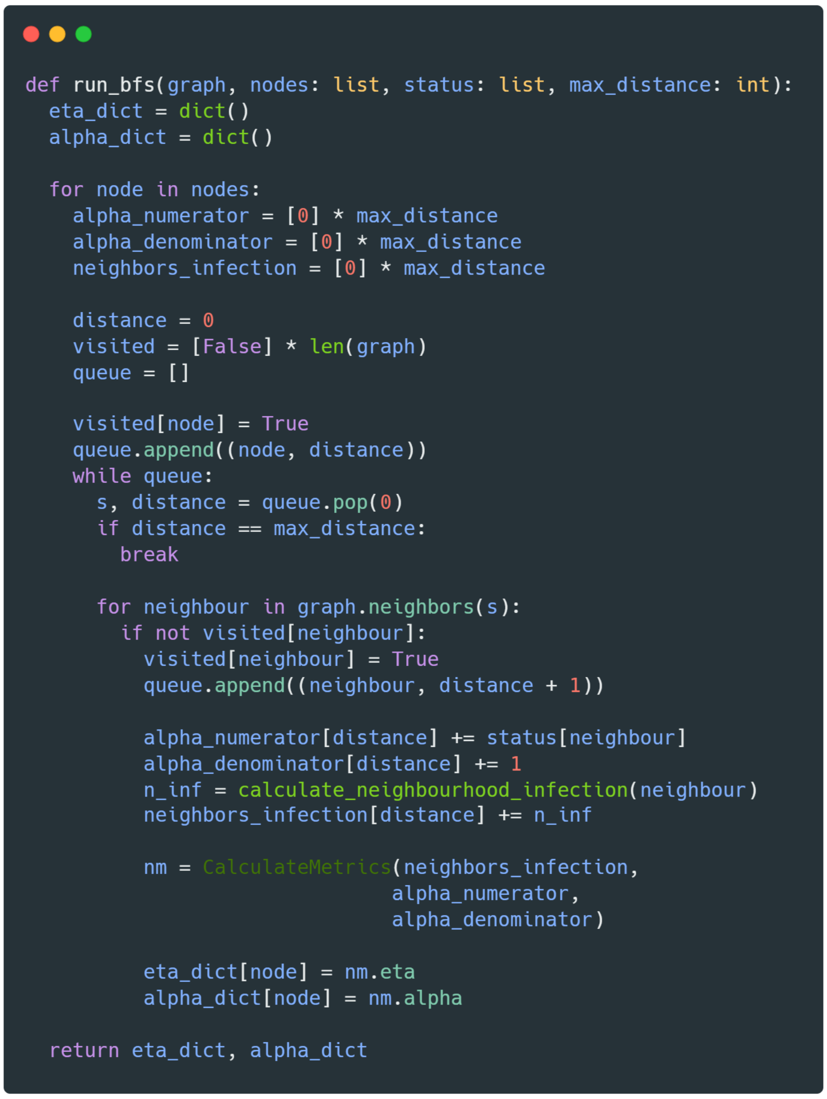
This chapter describes the network preprocessing algorithm that generates nodes attributes by using the observed epidemic state of network nodes. Also, GCN building blocks involving stacking of neural network layers, loss function and sampling algorithm are presented. Section 3.1 describes the algorithm that generates new attributes and Section 3.2 describes GCN model used in this work.

3.1 Node Feature Generation

Although the observed network contains the epidemic state of all nodes when a certain fraction of nodes is infected, it still lacks attributes to provide better inputs to the GCN. These attributes should enrich nodes with information that characterize its neighboring epidemic state. To accomplish that, two new attributes called *ring infection* (RI) and *depth ring infection* (DRI) are proposed.

RI and DRI are calculated for every node by considering its neighborhood infection state up to a certain distance. Since the aim is to calculate attributes against nearest nodes, BFS (Breadth First Search), which running time is $O(|V| + |E|)$, is used. However, this upper bound is rarely reached because the metric requires visiting nodes until a pre-established distance; it starts by visiting every node within one hop distance from the target node (the one that the attributes are being calculated for) and added to a queue so that they are not visited again. For the next step, neighbors of the previous one hop distance nodes are visited until a certain distance is reached or until all nodes are visited. After that, the information of infection state and distance discovered for each node is used to calculate both attributes for the given node. This process repeats for every node in the graph. Algorithm 3.1 shows the implementation of BFS.

Whereas the processing for every node is independent, it is possible to improve performance by making better use of vacant computational resources. Nodes are divided in k equal parts and each of these sets of nodes are used as the input to



```python
def run_bfs(graph, nodes: list, status: list, max_distance: int):
  eta_dict = dict()
  alpha_dict = dict()

  for node in nodes:
    alpha_numerator = [0] * max_distance
    alpha_denominator = [0] * max_distance
    neighbors_infection = [0] * max_distance

    distance = 0
    visited = [False] * len(graph)
    queue = []

    visited[node] = True
    queue.append((node, distance))
    while queue:
      s, distance = queue.pop(0)
      if distance == max_distance:
        break

      for neighbour in graph.neighbors(s):
        if not visited[neighbour]:
          visited[neighbour] = True
          queue.append((neighbour, distance + 1))

          alpha_numerator[distance] += status[neighbour]
          alpha_denominator[distance] += 1
          n_inf = calculate_neighbourhood_infection(neighbour)
          neighbors_infection[distance] += n_inf

          nm = CalculateMetrics(neighbors_infection,
                                alpha_numerator,
                                alpha_denominator)

          eta_dict[node] = nm.eta
          alpha_dict[node] = nm.alpha

  return eta_dict, alpha_dict
```

Figure 3.1: Breadth First Search used to compute the metrics that are used as node attributes.

a different process that runs concurrently. In practice, the nodes of the graph are divided and submitted to a different process that runs the BFS algorithm. Observe that nodes belonging to a partition do not need to be adjacent. Despite making calculations faster, the memory usage in this approach is high because every process carries a copy of the graph.

The implementation for calculating the two attributes for all nodes takes as parameters *max distance* and *number of workers*; *max distance*, d_{max}, indicates the distance from the target node until which the attributes are calculated, and the dimension of the attribute vector for each metric (if depth equals one, only the direct neighbours of the target node will be taken into account and each metric will be an one-dimension vector). The parameter *number of workers* represents the number of cores in the CPU and is used to determine the number of sets to divide the nodes of the network.

3.1.1 Attributes

The node attributes are calculated to enhance node characterization based on its neighboring labels. The RI leverages the epidemic state at k-hop distance while DRI leverages the neighbors epidemic state of a node at k-hop distance from the target node.

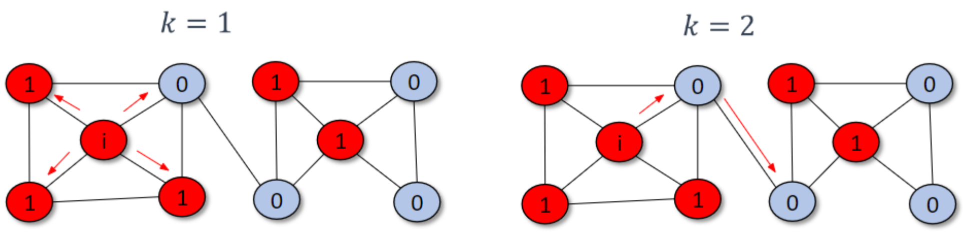


Figure 3.2: Graph example of how the ring infection is calculated for $k = 1$ and $k = 2$. For $k = 2$, the result is $\alpha^i = [3/4 \quad 0]$.

Ring infection (RI) is given by Equation 3.1 where $\mathcal{N}_k(i)$ represents the neighborhood of node i in k-hop distance (i.e., set of nodes that are at distance k from node i) and I_j the epidemic state of node j, where 1 indicates infected and 0 indicates susceptible.

$$\alpha_k^i = \frac{\sum_{j \in \mathcal{N}_k(i)} I_j}{|\mathcal{N}_k(i)|} \tag{3.1}$$

Note that α_k^i denotes the fraction of values computed for each $1 \leq k \leq d_{\max}$ from a vector that is taken to be a feature to node i. For $d_{\max} = 3$, the vector would be:

$$\alpha^i = [\alpha_1^i \quad \alpha_2^i \quad \alpha_3^i] \tag{3.2}$$

Figure 3.2 shows an example on how to calculate α for node i in a small graph. The red arrows indicate which nodes are taken into account when calculating α_k^i for a certain distance k.

The second metric *depth ring infection* (DRI) is calculated according to Equation 3.3. Observe that DRI depends on the epidemic state of the neighbors of nodes that are at distance k from node i. In particular, it computes the average of RI at distance 1 among its nodes that are at distance k. Note that a single node can be counted multiple times in this metric, since it can be a neighbor of many nodes at distance k from i. Also, observe that DRI provides information that is significantly different from RI.

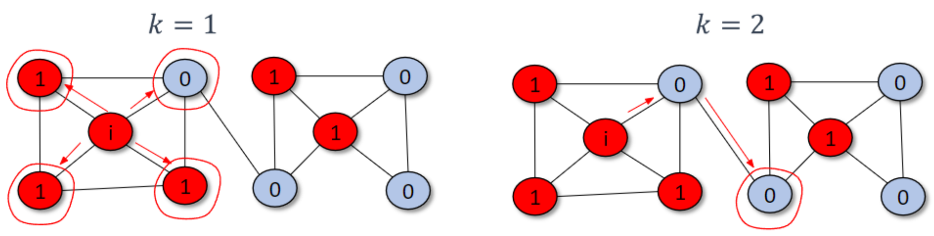


Figure 3.3: Graph example of how the depth ring infection is calculated for $k = 1$ and $k = 2$. For $k = 2$, the result is $\eta^i = [37/48 \quad 2/3]$.

Again, values computed for each k form a vector that is taken to be a feature of node i:

$$\eta_k^i = \frac{\sum_{j \in \mathcal{N}_k(i)} \alpha_1^j}{|\mathcal{N}_k(i)|} \tag{3.3}$$

Figure 3.3 shows how to calculate η^i for node i on the same small graph. The red arrows indicate which nodes are taken into account at a given distance and the circles around the nodes indicate the edges of the neighborhood for each node taken into account.

3.2 GCN model and implementation

GCNs are built by stacking layers that encode graph information at different distances. A well-known and well-documented framework called PyTorch Geometric [33] uses PyTorch [34] to create a variety of GNN models, including SAGE layers, which are used in this work.

Layers generate output that is used as input to the next layers until a final layer is reached at which point a node embedding is generated. While a larger number of layers makes the information scatter among neighboring nodes, a reduced number of layers would prevent the information from spreading far. Thus, empirical studies suggest using neural networks with three layers or so, as is the case in this work.

The first layer needs to adjust to the number of inputs according to node features; in this case, it comprehends nine dimensions, assuming $d_{\max} = 4$: four for each attribute and one for the infection state of the node. The output dimension is the desired latent representation dimension. In this case, the dimension is 128 to avoid overfitting while also being a relative large number compared to the size of the networks to be tested.

The middle layer has the input dimension equal to the output dimension and, in this work, equals to 128. On the last layer, the latent representation must converge all of its information to a single neuron output. This is done because the source identification problem consists of a binary classification problem and one output being labeled 0 (not source) or 1 (source) is enough.

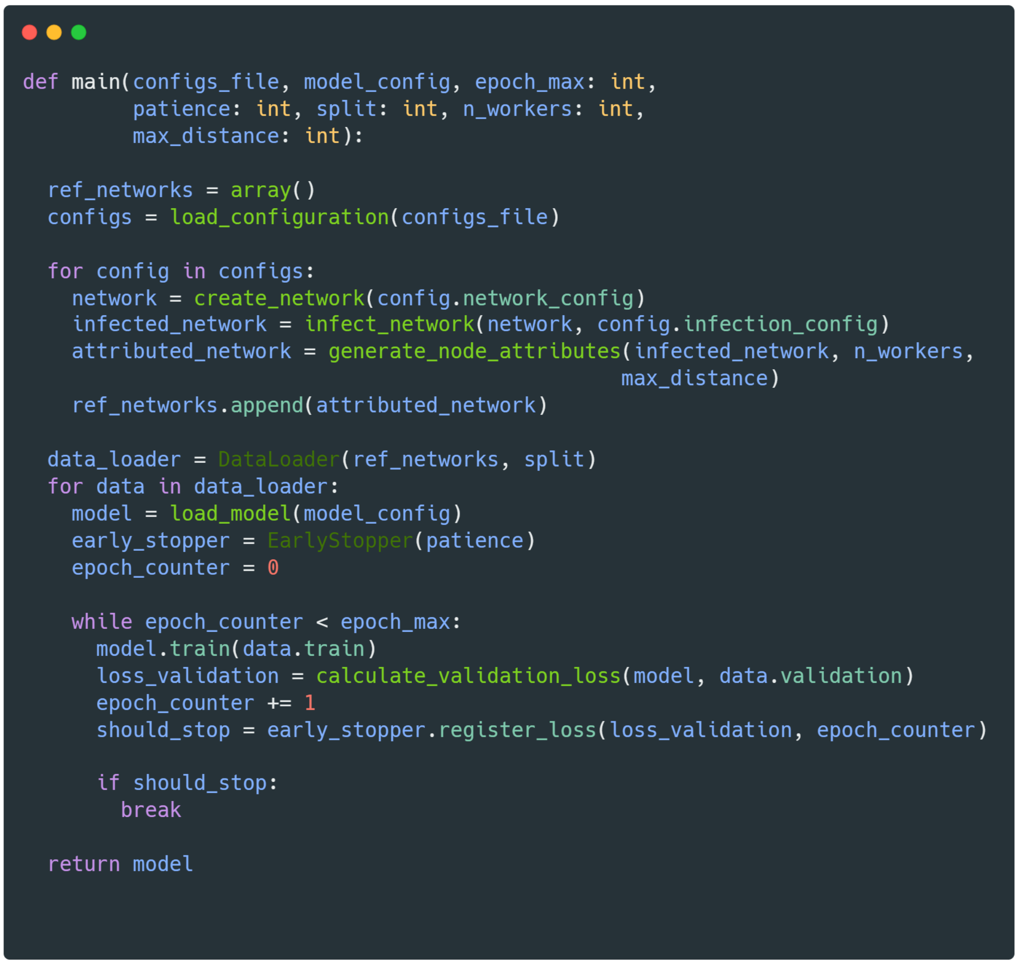
The first and second layers use the *ReLU* as activation function together with a dropout layer to help avoid overfitting. The last layer output is passed to a *sigmoid* function to output values between 0 and 1. Adam optimizer is the preferred optimizer used with a low *learning rate* of 0.001.

Although smaller batch sizes may be used when dealing with large networks, all network nodes were considered as a single batch at each learning epoch. As for sampling neighboring nodes, nodes were picked up to a distance 3 of the target node. This sampling strategy is used to balance computational cost and network coverage as in this problem nearer nodes carry more important information.

Naturally, epidemic source nodes are underrepresented in all networks and for that matter, a weight balancer is used in the loss function to calculate weight compensation. The model is trained on 500 epochs and an *early stopper* is implemented so that if no loss improvements are made in the validation network on 75 consecutive epochs (*patience*), the training with the input network stops. The experiments were carried on a computer with an Intel Core i7-11800H, GPU NVIDIA GeForce 3070 8GB and 16GB of RAM.

Algorithm 3.4 shows the general steps taken by the whole process to output a trained GCN model that can identify epidemic source nodes. The code is divided in two big loops that comprehend the data preparation and training. First, a bulk of real world and synthetic networks (from different network models) are created leveraging configuration parameters for training, validation and testing. For every network, a epidemic is simulated according to an epidemic model until a certain fraction of nodes become infected. Afterwards, a function to compute the node attributes is executed. Note that algorithm 3.1 is called inside the attribute generator function. The network configuration encompass, if synthetic, a network model, number of nodes, number of edges and edge probability, while infection configuration encompass number of sources, infection fraction and node infection probability.

Once the dataset with the ground truth has been generated (network and node attributes), the data is split for test, training and validation. The model is trained in two different ways: using only one specific network infection configuration and network model in the same training session or using multiple network infection configurations on the same network model. In order to avoid overfitting to a certain network instance, a validation network is evaluated in every epoch and skips learning with the current training network if it does not reduce the validation loss after a certain number of iterations, denoted by *patience* in the code.



```
def main(configs_file, model_config, epoch_max: int,
         patience: int, split: int, n_workers: int,
         max_distance: int):

  ref_networks = array()
  configs = load_configuration(configs_file)

  for config in configs:
    network = create_network(config.network_config)
    infected_network = infect_network(network, config.infection_config)
    attributed_network = generate_node_attributes(infected_network, n_workers,
                                                   max_distance)
    ref_networks.append(attributed_network)

  data_loader = DataLoader(ref_networks, split)
  for data in data_loader:
    model = load_model(model_config)
    early_stopper = EarlyStopper(patience)
    epoch_counter = 0

    while epoch_counter < epoch_max:
      model.train(data.train)
      loss_validation = calculate_validation_loss(model, data.validation)
      epoch_counter += 1
      should_stop = early_stopper.register_loss(loss_validation, epoch_counter)

      if should_stop:
        break

  return model
```

Figure 3.4: General framework to build a ground truth dataset and train a GCN model to classify epidemic source nodes.

3.2.1 Loss Function

Recall that the GCN model proposed in this work has a single output neuron. Thus, the loss function uses this single output in a supervised scenario in order to train neural network parameters. In particular, the Binary Cross Entropy function was used and is given by:

$$l_i = -w_i \left[y_i \cdot \log \sigma \left(x_i \right) + \left(1 - y_i \right) \cdot \log \left(1 - \sigma \left(x_i \right) \right) \right] \tag{3.4}$$

Where y_i indicates if node i is a source node in the epidemic observed in the network (ground truth), x_i is the output of the neural network for node i, and w_i is a weight associated to node i. Despite being possible to associate different weights for each node, a different weight is used for each class (source node or not source node). A *sigmoid* function σ is applied to the output value of the neural network such that the output value is between zero and one, taking advantage of the log-sum-exp trick for numerical stability. Representing classes weights, W_0 and W_1 are calculated for each network before training by:

$$W_c = K_c \cdot \frac{n}{\sum_{i=0} \left[y_i = c \right]} \tag{3.5}$$

The amount of network nodes is denoted by n and K_c is a constant used to increase class weight beyond balance value, the proposed framework opted to use $K_0 = 1$ and $K_1 = 100$. Weight classes are then broken into node sample weights w_i, with each node having the weight of its infection state class. Finally, the loss of all nodes are aggregated into a single loss through a weighted average using the same weights as before, and given by function ℓ as:

$$\ell = \frac{\sum_i l_i}{\sum_i w_i} \tag{3.6}$$

Chapter 4

Methodology and Evaluation

This chapter evaluates the model outputs when using different networks as inputs; synthetic networks and real networks. It also discusses the creation of datasets and training strategies. Section 4.1 describes how network samples were generated, Section 4.2 explains metrics used to evaluate results and Section 4.3 displays results.

4.1 Infected Graph and Epidemic Factory

There are various approaches to train the proposed model, ranging from specialist to generalist approach. In the former, the training set are networks samples of the same network model that have the same size, while epidemics have the same parameters and the same number of sources. In the latter, the training set is very diverse. Both approaches will be evaluated in what follows. In any case, it is necessary to generate networks and epidemics to build the training sets. For this purpose, a framework was developed to simulate different types of epidemics through different networks; for example, SI epidemic with multiple sources with different parameters on different networks generated by the BA model. The framework generates multiple networks by sampling random network models or reading an edge list and simulate the epidemic using a network epidemic library according to the configuration file.

For generating random networks and manipulating them in general, *NetworkX* [35] is the go to option as its is a well maintained Python package. *NDLib* (Network Diffusion Library) [36] is a network epidemics library used to simulate epidemics on networks according to different compartmental models such as SI, SIR and IC. Each network is initialized with and epidemic with its respective parameters, SI uses β for infection probability, and the initially infected nodes (sources), which may be randomly or deterministically selected. For each network where the epidemic sources have been chosen, iterations have to occur in order do propagate the epidemic to other nodes. The initial number of infected nodes greatly influence the final result and the speed at which the propagation occurs on the network, and the epidemic stops when a certain fraction of nodes have been infected. Notice that in order to characterize the performance of the proposed methodology, network epidemics with $3, 5, 10$ and 15 sources were considered until around 10%, 20% and 30% of the nodes were infected. Observe that it is not possible to guarantee that an exact percentage of the nodes are infected since each infected node is able to infect more than one neighboring node in each interaction. However, it is guaranteed that at least that percentage of nodes are infected. The infected network is added as a single sample into the dataset. Different samples are generated independently by repeating the entire process; generating another random network, randomly selecting the source nodes, and simulating the random epidemic until a percentage of the nodes are infected.

4.2 Evaluation

The dataset with the infected networks are divided in three different sets: training, testing and validation. The implementation of early stopping algorithm previously discussed relies on the training and validation datasets having the same number of samples. The experiments are divided in two categories: **(i)** datasets where all networks have the same network topology, number of sources and infection percentage, **(ii)** datasets where all networks have the same model but different number of source and infection percentages. It is important to note that epidemics on real networks differ only on the choice of source nodes, since the network is always the same compared to random network models where each sampled network is different.

In the experiments that follows, the results are divided in two: **(i)** true positives are only the epidemic source nodes and every other node identified by the framework as a source is considered a false positive **(ii)** true positives are the epidemic sources and their neighbors. Observe that the latter criteria measures the performance of the framework on identifying the epidemic source or a neighbor of the source as the epidemic source.

The evaluation also does not directly determine the number of epidemic sources. Instead, every network node is taken as the input to the model which outputs the probability that the node is an epidemic source. Nodes are then ranked according to this probability and the top-k nodes are identified as the epidemic sources. Thus, k is a parameter of the evaluation methodology.

4.2.1 Metrics

Note that accuracy is not a good metric to capture the model's performance, since having a very unbalanced dataset is a characteristic inherent (very few source nodes) to the problem. Thus, it is more useful to use metrics that capture the model's performance on correctly identifying the sources, hence metrics that take into account true positives (TP), false positives (FP) and false negatives (FN) are preferred.

Using TP, FP and FN values, three evaluation metrics are considered: **precision**, **recall** and **F-score**. **Precision** is denoted by the number of true positives identified by the framework divided by the sum of true and false positives identified by the framework. **Recall** is denoted by the number of true positives identified by the framework divided by the sum of true positives and false negatives in the dataset. Both metrics are combined to build **F-score**. The F-score used applies the same weight to the two metrics and is given by:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{4.1}$$

Observe that the number of true positives plus the number of false positives identified by framework is given by k since these are the top-k most probable epidemic source nodes.

4.2.2 Network models and real networks

The evaluation is divided in using random network models and real networks. Two models are used: Barabási-Albert (BA) and Erdős–Rényi (ER)[37, 38], and two real world networks are used: Facebook Ego Network [39] and the Power grid Network [40]. Facebook Ego is a social network consisting of 10 ego-networks and 4039 users as 10 Facebook users were asked to manually classify their friends. The power grid network depicts the electrical power grid of the western United States. Vertices represent generators, transformers and substations, and edges represent high-voltage transmission lines between them. The Erdős–Rényi (ER) graph is also known as $G_{n,p}$. Each possible edge is present in the graph with probability p.

The Barabási-Albert (BA) model is based on preferential attachment and generates a scale free network. It provides a better model for some real networks where nodes tend to link to the more connected nodes [7]. For each newly added node, m new edges are connected to m nodes already in the network, chosen at random according to the preferential attachment rule.

The random networks are parameterized such that their average degree is $2log_2n$ where n is the number of network nodes. This is done so that the experiments are not so biased due to very different average degrees. Table 4.1 shows some characteristics for the networks used. Average degree values marked with an asterisk represent the expected value, since edges are random.

On the BA model, the average degree is given by Equation 4.2 where m, which is a parameter for network model, can be isolated.

$$
\begin{aligned}
d &= \frac{2m(n-1)}{n} \\
2\log_2 n &= \frac{2m(n-1)}{n} \\
m &= \frac{n\log_2 n}{n-1}
\end{aligned}
\tag{4.2}
$$

For $n = 1500$, $m = 10.558$ and for $n = 5000$, $m = 12.290$. On the other hand, on the ER model, the average degree is calculated by Equation 4.3 where the goal is to isolate p that represents the probability for edge creation.

$$
\begin{aligned}
d &= (n-1)p \\
2\log_2 n &= (n-1)p \\
p &= \frac{2\log_2 n}{(n-1)}
\end{aligned}
\tag{4.3}
$$

For $n = 1500$, $p = 0.01408$ and for $n = 5000$, $p = 0.00492$.

Network	Nodes	Edges	Avg. Degree	Diameter
BA 1500	1500	15827	21.102	4
BA 5000	5000	61438	24.575	4
ER 1500	1500	15827*	21.102*	4
ER 5000	5000	61438*	24.575*	4
Power grid	4941	6594	2.669	46
Facebook Ego	4039	88234	43.691	8

Table 4.1: Structure information about the networks used in the evaluation.

Regarding datasets size and split for training, validation and testing; for every line in Table 4.1 concerning synthetic networks, 30 networks are generated and split equally in three sets. For the real world networks, 50 networks are generated, from which 1/5 are used for training, 1/5 for validation and the rest for testing. A larger dataset is used for testing to guarantee a more stable average of values to be compared with other articles results.

4.3 Results

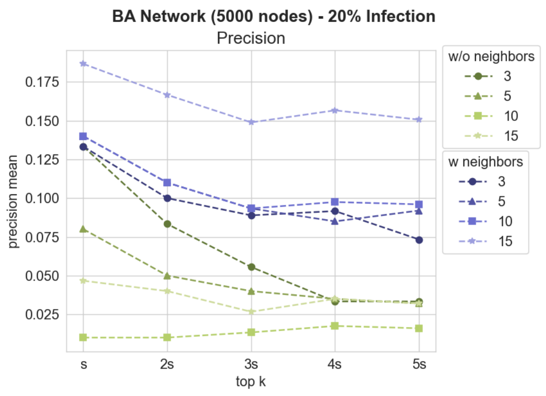
The results are presented in three types of graphs discussed on the following paragraphs. The first two focus on comparing the performance of identifying the source node or a neighbor of the source, and the last one compares the models trained on a more diverse dataset.

4.3.1 Performance on artificial network models

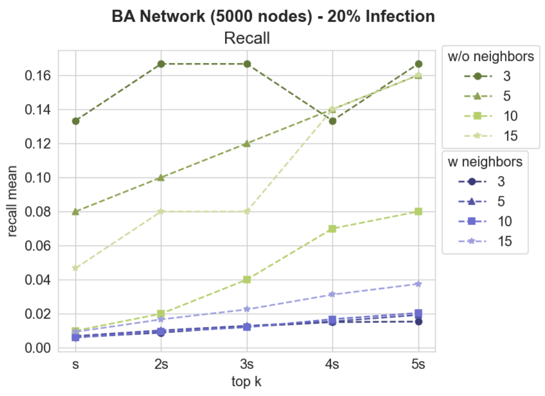
On the first graph type, each line represents the average metric value for the test set of the corresponding dataset. Whilst **F-score** is largely used alone to compare MSD models, precision and recall are also shown here in order to better characterize the results. The plot lines are divided in groups: **(i)** epidemic sources neighbors are considered true positive and **(ii)** epidemic sources neighbours are considered false positives. Even though the main goal of the model is to identify the epidemic source, identifying a neighbor of an epidemic source also indicates that the model has come close to identifying the source.

The second graph type compares precision, recall and F-score for the three fractions of infected nodes. It allows to compare the evolution of results as the epidemic infects a larger fraction of the network.

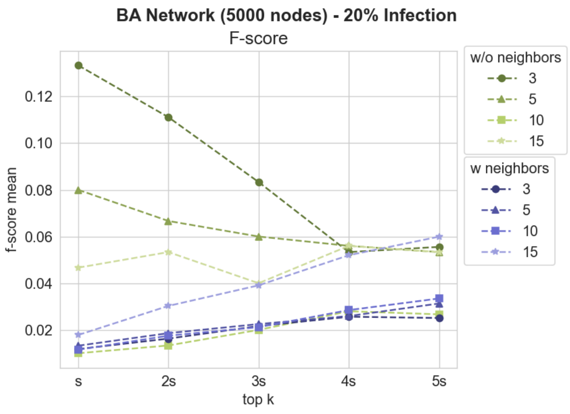
On both graphs aforementioned, the x axis represents the number of epidemic sources identified by the framework as given by the top-k nodes (x axis corresponds to k). The values are multiples of 2, 3, 4 and 5 of the true number of epidemic sources. Thus, $k = 30$ if the number of sources is 10 and the multiple is 3.



(a) Precision



(b) Recall

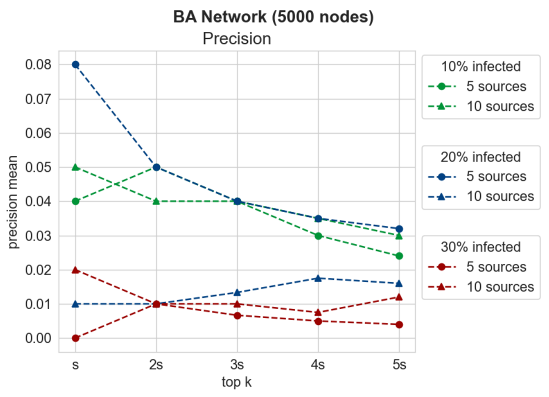


(c) F-score

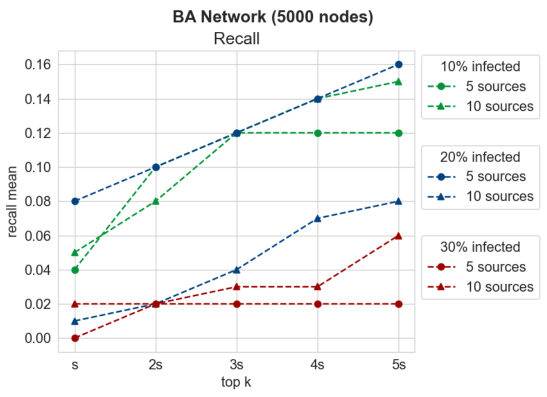
Figure 4.1: Results for source identification considering the BA model network with 5000 nodes when 20% of them are infected.

Figure 4.1 compares artificial network generated by BA model. On average, **with neighbors** precision values decrease with the increase of top-k, while some values of **without neighbors**, namely 10 and 15 sources, increase and decrease, respectively, very little. **With neighbors** recall values increase very little with the increase of top-k, while values of **without neighbors** have a greater value increase. At last, F-score on **without neighbors** two smallest source values decrease with top-k, whilst the other two increase, **with neighbors** show similar values, with the exception of 15 sources, that follow an increase trend.

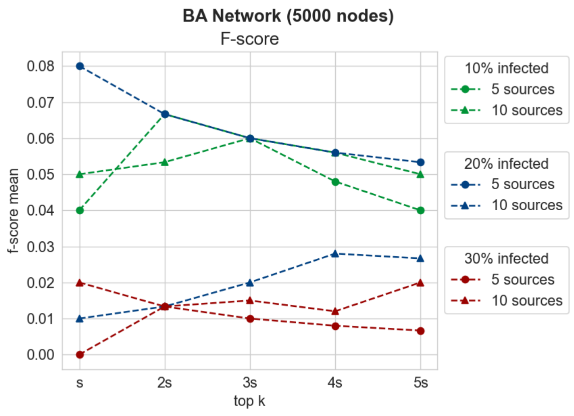
With neighbors precision values are larger than the **without neighbors** values, while **without neighbors** recall values are larger than the **with neighbors** values. Therefore, **without neighbors** F-score values are on average larger because of the larger recall values and not so small precision values.



(a) Precision



(b) Recall



(c) F-score

Figure 4.2: Results comparison across different infection rates for BA model with 5000 nodes.

When the neighbors of source nodes ground truth are labeled as sources, it is expected that the model achieves higher precision than when they are not. On the other hand, it is expected that recall achieves lower values due to more false negatives being present enforced by the low amount of predicted sources in comparison to the amount of neighbours now considered true positives.

Precision **with neighbors** is on average larger indicating that mistaking the source neighbors for source occurs frequently. **With neighbors** reaches lower recall values due to the increase in the number of false negatives that are represented by the number of sources multiplied by average node degree. The value for F-score of **without neighbors** ends up being higher on average because of **with neighbors** recall value. The Appendix Section A.1.1 presents results for other fractions of infected nodes. However, Figure 4.1 is illustrative of the general trend among these results.

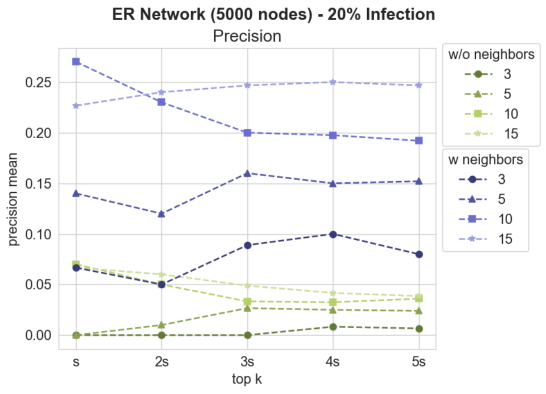
Figure 4.2 compares metrics values across different infection percentages. Note that, in general, as the fraction of infected nodes increases, the performance decreases for both precision and recall. Moreover, the performance when having **5 or 10 epidemic** sources are relatively similar for **20% and 30% infection rate**, with slightly better performance for **10 sources**. Individually, the mean precision for **10% infected 5 sources** increases to its largest value when top-$k = 2s$, but decreases beyond its initial top-k value, for **10% 10 sources** the largest value is reached when top-$k = s$ and then it decreases, for **20% infected 5 sources** the largest value happens when top-$k = s$ and it decreases from this point on.

While **10% infected** case intuitively reaches larger F-score values and **30% infection** lower values, **20% infected** results depend much more on the amount of sources; the smaller the amount of sources the greater the F-score; **20% infected 5 sources** reaches even larger values than the **10% infected** case.

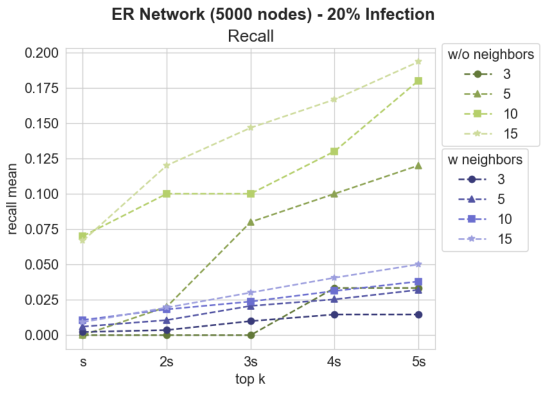
Figure 4.3 compares networks generated by $G(n, p)$ model. On average, **with neighbors** precision values increase with top-k, **3 and 5 sources** increase very little, while **10 and 15 sources** decrease. **With neighbors** recall values increase very little with top-k compared to **without neighbors** values. At last, F-score for **without neighbors 15 sources** case reaches its peak at top-$k = 2s$ and then decreases, **10 sources** values decreases but starts to show larger values after $3s$, whilst the other two (**3 and 5 sources**) increase, **with neighbors** show a increase trend for all source number cases.

Similar to the previous experiment, the model mistakes the neighbor's source for source. Also, it is noticeable the drop in precision values when decreasing the amount of sources. The **3 sources** case on **without neighbors** stands out negatively, being far below the other examples in the same label bracket. The values for F-score show a expected behavior: **with neighbors** values surpass all of their **without neighbors** counterparts with the increase of top-k. Thus, having more epidemic sources helps improving the performance. The Appendix Section A.1.2 presents the results for other infection ratios.

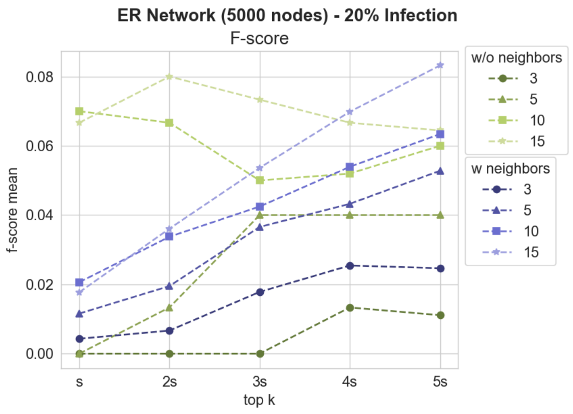
Figure 4.4 compares $G(n, p)$ network metrics values across different infection per-



(a) Precision



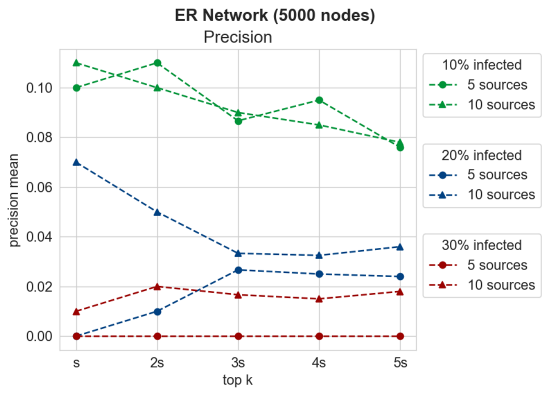
(b) Recall



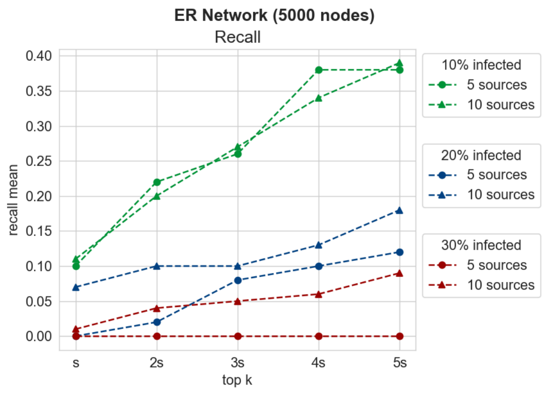
(c) F-score

Figure 4.3: Results for source identification considering the ER model network with 5000 nodes when 20% of them are infected.

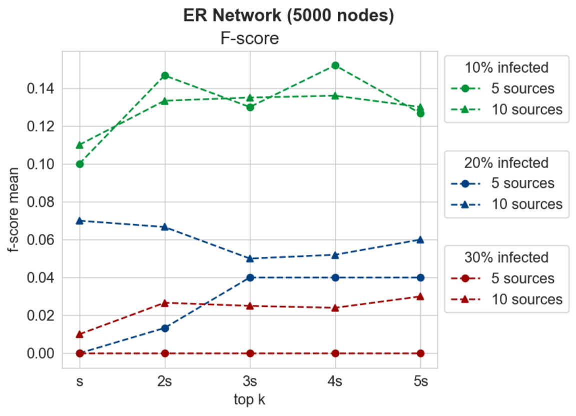
centages. **10% 20% and 30% infected** cases remain at the same relative position in every graph. For **10% infection rate**, having **5 or 10 epidemic sources** yield similar results. However, on **20% and 30% infection rates**, performance with **10 sources** is significantly better that with **5 sources**. In such scenarios, having more epidemic sources improves precision and recall. Nevertheless, there is a clear gap between **10% infection rate** results and the other two, showing that infection rate variation greatly influences the model performance. This group of graphs illustrates the intuition that the higher the infection, the lower the F-score.



(a) Precision



(b) Recall



(c) F-score

Figure 4.4: Results comparison across different infection rates for ER type network with 5000 nodes.

4.3.1.1 Model comparison on artificial networks

The following results consider different kinds of datasets used for training. Two scenarios are considered: models trained in a specific configuration are called **individual** e.g. trained on the BA network model, 20% of the network infected and 5 source nodes. The other scenario is called **general** because the models were trained using datasets comprising of networks of the same model but different number of sources and infection rate. In order to verify the impact of network size on the task of MSD, two network sizes are considered.

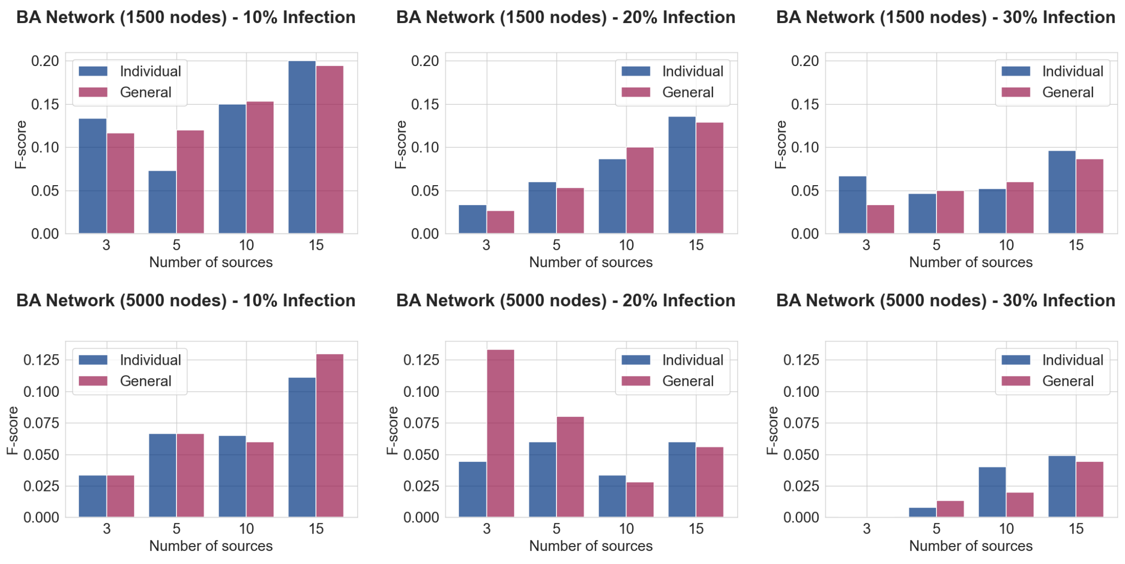


Figure 4.5: Comparison of performance when training the model with BA type network with different datasets.

Figure 4.5 **BA 1500 10% infection** shows an increase on F-score values when increasing the number of sources, except on the **individual 5 sources** case where it decreases. **BA 1500 20% infection** shows an increase on F-score values when increasing the number of sources. **BA 1500 30% infection** also shows an increase on F-score values when increasing the number of sources, except on **individual 3 sources** where it manages to reach the second largest value.

Group of graphs on Figure 4.5 **BA 1500** portrays the trend that the larger the infection percentage, the lower the F-score. A satisfying surprise comes when comparing **individual** and **general** classification models. Despite being trained in a more heterogeneous dataset, **general** F-score values are very close to **individual**, even performing better in some cases. Group of graphs on Figure 4.5 **BA 5000** also portrays the same trend with the exception of **20% infection 3 sources** outlier case. The **30% infection rate** scenario depicts one of the worst results among the experiments with a F-score of 0 for **3 sources**.

Despite intuition indicating that with more epidemic sources the harder is to find them, since the infection clusters might mix and their intersection be identified as sources, F-score increases together with the number of sources in most case scenarios. Nevertheless, the expectation that it would be harder to find sources when the infection rate increases remains true.

When comparing **BA 1500** and **BA 5000** results, it is noticeable that a determining factor for greater F-score values is the network size. Results for **BA 1500** are on average two times larger than **BA 5000** as it has 3.3 more nodes than **BA 1500**.

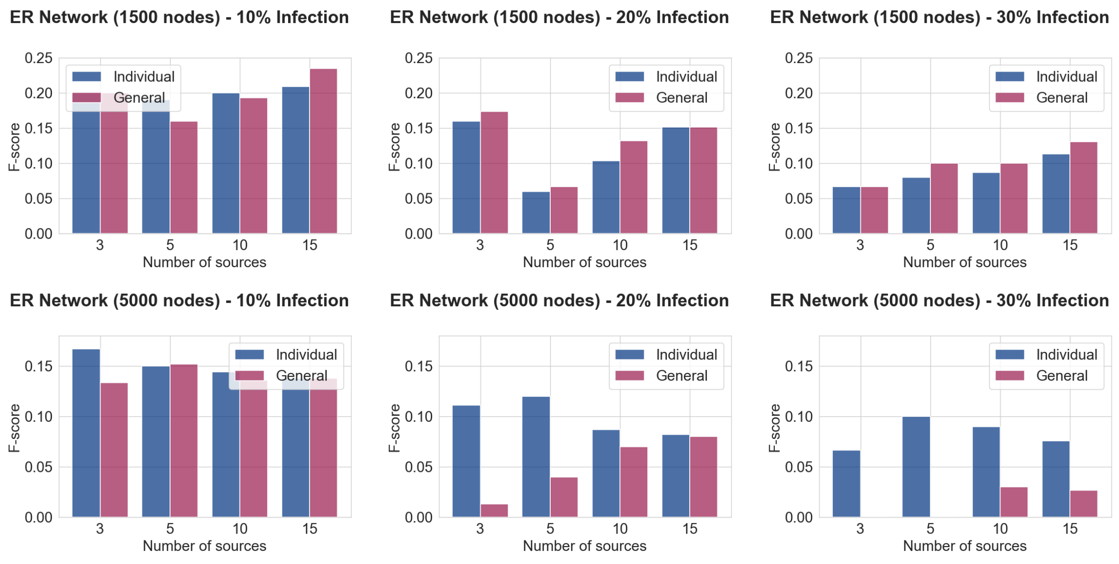


Figure 4.6: Comparison of performance when training the model with ER type network with different datasets.

Figure 4.6 **ER 1500 and ER 5000 10% infection** show an increase on F-score values when increasing the number of sources. **ER 5000 20% infection** shows a decrease on F-score values when increasing the number of sources, except on **5 sources**, on the **individual** case, while on **general** it increases. **ER 5000 30% infection** also shows an increase on F-score on **individual 5 sources** and **general** increases from zero with **10 and 15 sources**.

Figure 4.6 **ER 1500** shows that **general** surpass **individual** trained model on average. While F-score value decreases with the infection percentage increase, it presents slightly better values than BA with the same amount of vertices.

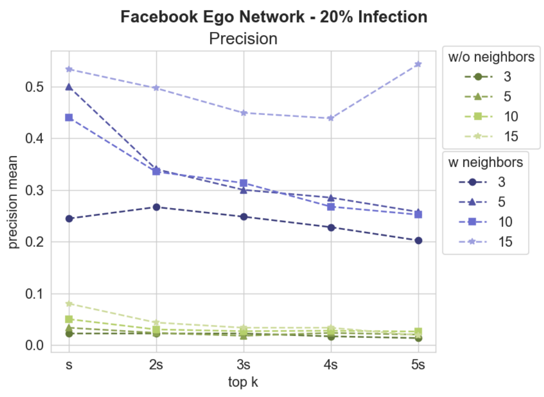
Figure 4.6 **ER 5000** results frustrate the use of **general** trained model from **20% infection** onward as **individual** reaches much better performance even when **general** shows a F-score of zero (in **30% infection rate**). Again, comparing to **ER 1500**, the F-score decreases more significantly for **general** than for **individual** that managed to score similar values on the **30%** case.

4.3.2 Performance on real networks

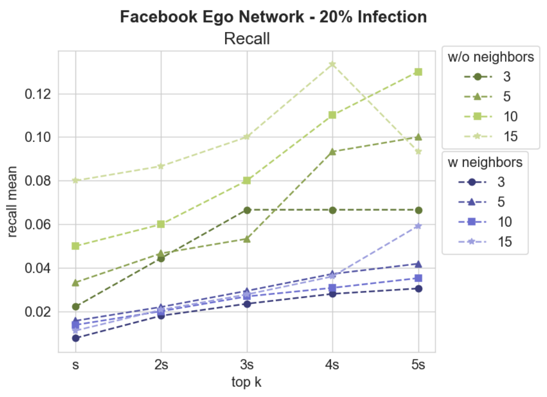
Facebook Ego network results are shown in Figure 4.7 where **with neighbors** precision values decrease for **5 and 10 sources** with the increase of top-k, **15 sources** starts decreasing before raising to its largest value where top-$k = 5s$ and **3 sources** values slightly increase to its largest value where top-$k = 2s$ before decreasing. **Without neighbors**, namely **5, 10 and 15 sources**, largest precision values are when top-$k = s$ and then they decrease, whilst **3 sources** remains constant.

With neighbors precision values are much larger than the **without neighbors**

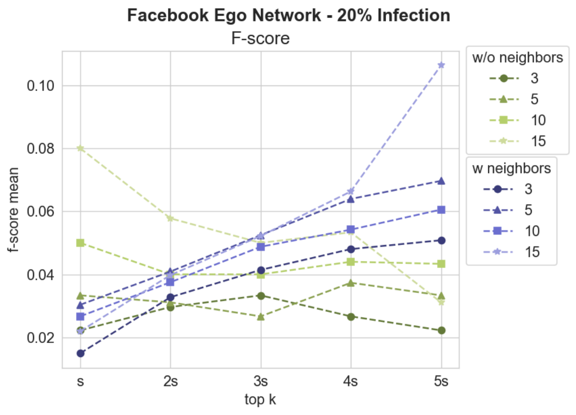
values, while **without neighbors** recall values are larger than the **with neighbors** values. Therefore, **with neighbors** F-score values increase with top-k while **without neighbors** decrease.



(a) Precision



(b) Recall



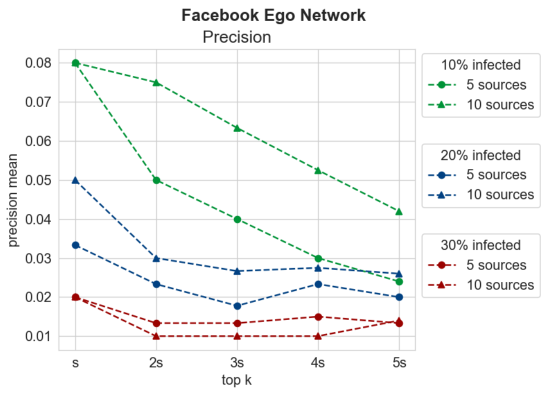
(c) F-score

Figure 4.7: Results for source identification considering the Facebook ego network when 20% of them are infected.

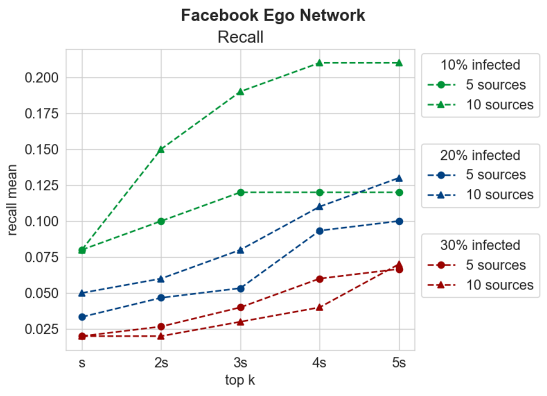
There is a big gap between **with neighbors** and **without neighbors** precision values. On recall graph, a behavior worth analysing happens from $4s$ to $5s$ for the **15 sources without neighbors** case; the big drop on recall shows that, on average, there was no improvement when increasing top-k, as false negatives increased more than true positives. The Appendix Section A.2.2 presents results for other infection rates.

On Figure 4.8, **10% infected** precision values decrease with top-k, **20% infected** values decrease from the first x value top-$k = s$, **30% infected** values also decrease from the first x value top-$k = s$. Recall graph shows that all values increase from the first top-k value, despite the **10% infected 5 sources** case where it remains constant from $3s$ to $5s$. **10% 20% and 30% infected** cases remain at the same relative position in every graph.

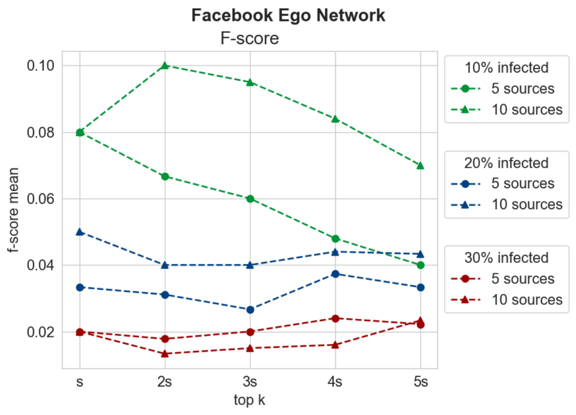
Precision graph displays a unprecedented decline on the **10%** case, although the



(a) Precision



(b) Recall



(c) F-score

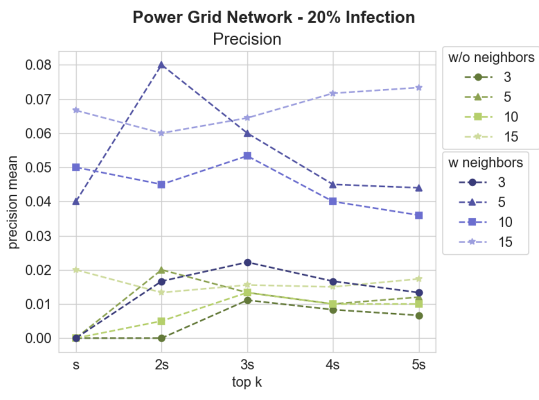
Figure 4.8: Results comparison across different infection rates for Facebook ego network.

same infection case with **5 sources** shows a large increase on recall results. Thus, on average, the model was not so efficient on correctly identifying sources with the increase of top-k, however it did not fail on the same scale on not identifying true sources.

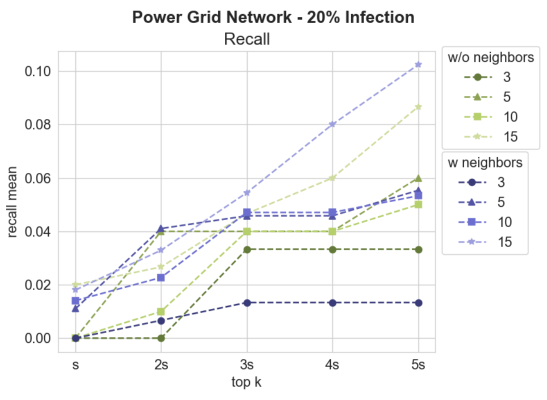
Figure 4.9 shows results for the Power grid network experiments. On the precision graph, considering **with neighbors 3 sources** increases to its peak when top-$k = 3s$.

Precision results are again superior for **with neighbors** in comparison to **without neighbors** under the same number of sources, but interestingly, recall results **with neighbors** are similar to **without neighbors**, differently from previous scenarios. Thus, F-score for **with neighbors** is significantly higher in comparison to **without neighbors** for this network.

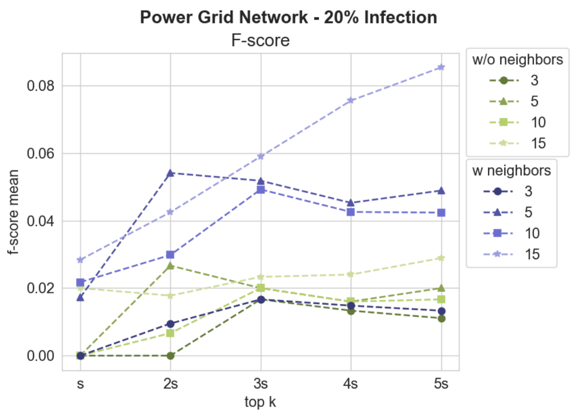
It shows the same trend considering **with neighbors** and **without neighbors**, except **with neighbors** values with **3 sources** that are so low that entangle with **without neighbors** values. Appendix Section A.2.1 presents the remaining infection cases.



(a) Precision



(b) Recall

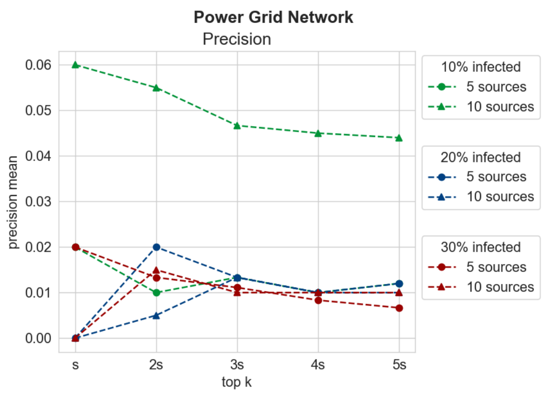


(c) F-score

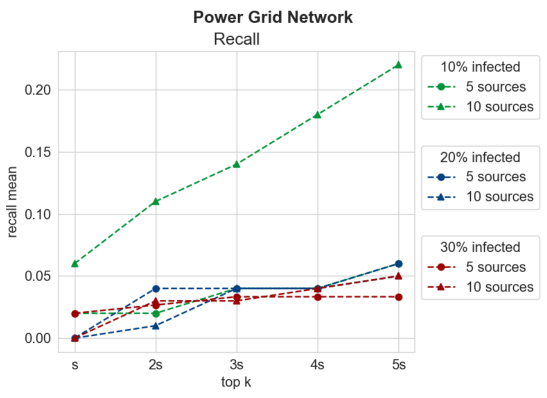
Figure 4.9: Results for source identification considering the Power Grid network when 20% of them are infected.

On Figure 4.10, precision graph shows that **10% case** values decrease compared to its first value where top-$k = s$, **20% case 5 sources** reaches its peak where top-$k = 2s$ and **10 sources** increases until it reaches its peak where top-$k = 3s$, **30% case 5 sources** decrease from the first x value top-$k = s$ and **30% infected 10 sources** case increases until it reaches its peak where top-$k = 2s$. **10% 20% and 30% infected** cases remain at the same relative position in every graph.

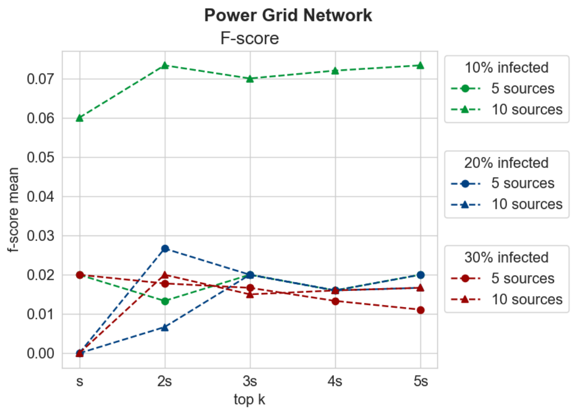
Results concerning different infection rates are very similar, with the exception of **10% infected** with **10 sources** which stands out. This could be happening because the Power grid network structure has very low average degree, hampering the emergence of infection clusters.



(a) Precision



(b) Recall



(c) F-score

Figure 4.10: Results comparison across different infection rates for Power grid network.

4.3.2.1 Model comparison on real networks

Figure 4.11 **10% infection general** shows an increase on F-score values when increasing the number of sources, on the **individual** case there is a big decrease on **5 sources** value before reaching its largest value with **10 sources**.

On Figure 4.11, aside from the 30% infection **3 and 5 sources**, all the other results from **individual** trained model far outweigh results from **general**. Results for **10 and 15 sources** are on average larger than **3 and 5 sources** for all the depicted cases reinforcing the idea that the model improves performance when increasing the number of sources. The F-score greatly decreases on the **30% infection** case, where the first zero result for **individual** trained model happens (not shown due to averaging).

Figure 4.12 for Power grid results **10% infection individual** shows an increase on F-score values when increasing the number of sources, on the **general** case there is a decrease on **5 sources** before increasing again. The same pattern from Facebook Ego network is observed; **individually** trained model achieves much better results, in many cases, up to twice the value of **general**. On **30% infection** case, despite

individual still having better results, they are much lower than the **20%** case. In addition, on **general** model scores very poor results on three and five sources.

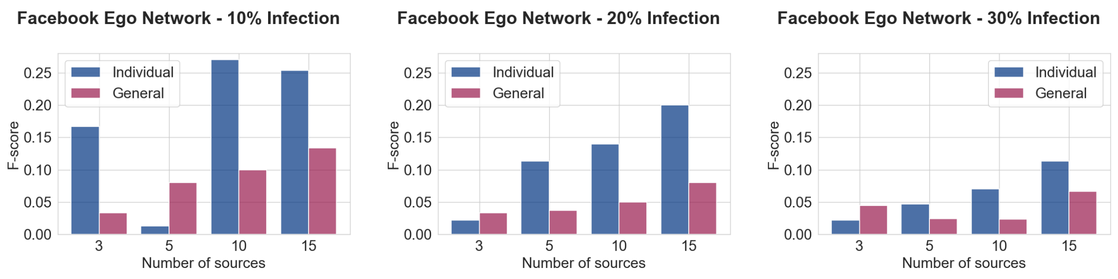


Figure 4.11: Comparison of performance when training the model with Facebook ego network with different datasets.

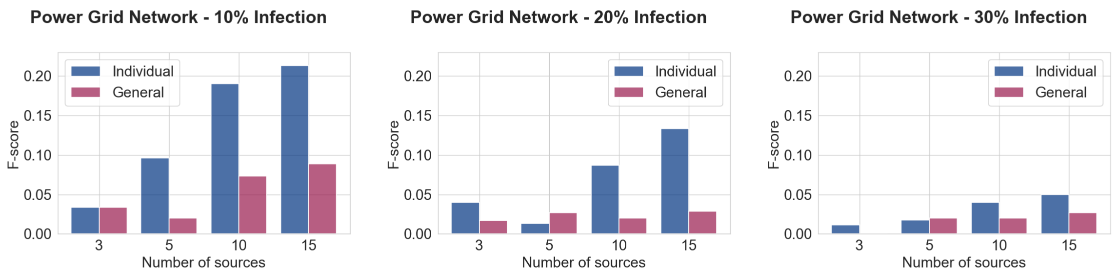


Figure 4.12: Comparison of performance when training the model with Power grid network with different datasets.

4.4 Ablation Study

The proposed framework is divided in two parts; the attribute generator algorithm and the GCN model. While the attribute generator is the responsible for generating β and η, GCN uses said attributes and structural data to generate embeddings, while using modifications proposed by this work, such as different sampling strategy, loss function and class weights.

To better understand the impact of the new attributes and GCN modifications, infected networks are used to train two different group of models, one group that does not leverage node attributes (β and η) and the other that does. The model is then trained and tested with the same dataset (with the exception of β and η).

Table 4.2 shows F-score results of models trained individually in specific network configurations. For each group **without generated attributes** and **with generated attributes**, four GCNs are trained with the Power grid dataset with 30% of network infected and one of the number of epidemic sources. Then, both groups are tested in the same dataset, generating the results presented.

Besides the 5 sources case where results on graphs **without generated attributes** are larger, results are expected. Both average and max F-score values are greater on **with generated attributes** in 3 out of the 4 cases.

Without generated attributes F-score			
No. sources	Max	Average	Variance
3	0.003704	0.000741	3×10^{-6}
5	0.040000	0.030800	9.9×10^{-5}
10	0.048000	0.041822	1.1×10^{-4}
15	0.056296	0.052593	1.5×10^{-5}
With generated attributes F-score			
No. sources	Max	Average	Variance
3	0.011111	0.008815	1×10^{-5}
5	0.017778	0.014089	1.1×10^{-5}
10	0.051111	0.049356	1.2×10^{-5}
15	0.065926	0.059556	2.6×10^{-5}

Table 4.2: Comparison of F-score values of GCNs trained and tested **without** the generated attributes and **with** the generated attributes (Power grid network).

4.5 Comparison with Baselines

Although this work has focused on synthetic networks and two real networks, other articles such as the one that implemented GCNSI [10] focused only on real world networks. Table 4.3 compares F-score results for 3, 5 and 10 epidemic source on 30% infection rate with results generated by GCNSI, LPSI and NetSleuth (results extracted from [10]). From these three frameworks, the only one that also uses GCN is the GCNSI. Note that experiments were not carried on the exact same datasets across SAGE and the other three algorithms. Although, the number of sources, infection rate and network topology were the same. Results presented on Table 4.3 are from models trained in heterogeneous datasets, containing various types of infection configurations.

Individually, it is noticeable that results are poor for all table entries since they are much closer to zero than to one. That shows that MSD is rather a difficult problem that still has a long way to being solved more effectively. In general, the results improve with the increase of the number of sources. The framework here proposed (SAGE in the table) stands out by having the best results across the table for any scenario. Also, the gap between SAGE and the other frameworks increases with the number of sources; SAGE's result on Facebook Ego with 10 sources are 5 times better than the second best framework.

$s = 3$		
	Power grid	Facebook ego
SAGE (Max)	0.011	0.022
SAGE (Mean)	0.009	0.018
GCNSI	0.006	0.012
LPSI	0.003	0.009
NetSleuth	0.001	0.005
$s = 5$		
	Power grid	Facebook ego
SAGE (Max)	0.018	0.047
SAGE (Mean)	0.014	0.039
GCNSI	0.009	0.017
LPSI	0.008	0.013
NetSleuth	0.003	0.006
$s = 10$		
	Power grid	Facebook ego
SAGE (Max)	0.066	0.138
SAGE (Mean)	0.060	0.112
GCNSI	0.017	0.022
LPSI	0.013	0.020
NetSleuth	0.006	0.009

Table 4.3: Side by side comparison of F-score from other frameworks with 3, 5 and 10 sources, and SAGE indicates the framework here proposed. Results extracted from [10].

Chapter 5

Conclusion and Future work

Identifying epidemic source nodes is a fundamental problem that has been tackled by multiple algorithms. The problem started by investigating the detection of single source epidemics and them moved on to the more challenging scenario of multiple sources. A good portion of the algorithms used for source detection included node label propagation and are applicable to only a specific type of infection model and can not be directly applied to different infection models. The most prominent and recent algorithms for multiple source detection were discussed in this work [9, 10, 13, 32].

The framework here proposed starts by applying a node feature enrichment to enhance the information associated with each network node since it is initially a binary number denoting the infection status. The concept of neighborhood infection is used to create attribute vectors that characterize the nearest neighbors infection state. The key idea behind the proposed algorithm is to capture a pattern that should repeat around other epidemic sources. The attributes generated from an epidemic network serve as better input to a GNN model that not only leverages network structure but also node attributes in order to classify nodes as epidemic sources. Thus, the GNN model has a single output that is used for classification. The proposed framework offers a general model that can be trained with any dataset considering different networks, different number of nodes, and different infection rates.

In order to measure and compare results, the BA and ER random graph models were used to generate networks, along with Facebook and Power Grid real networks, providing a rich and diverse set of network structures. The classic SI epidemic was used to simulate epidemics on these networks with different number of sources. The precision, recall and F-score were the metrics chosen to evaluate performance. Moreover, the trained model can be applied to any epidemic network to perform identification of epidemics sources, although it might not score the best results when those are too different from the training set.

The empirical evaluation considered the identification of the epidemic source or identification of the neighbor of the epidemic source. The performance of the proposed framework was extensively evaluated in different scenarios.

Although the results are not satisfactory individually (low F-scores in almost all scenarios) a direct comparison with prior works shows that this framework is competitive with other published frameworks. In all scenarios available for comparison on real networks, this framework showed significantly better results. Interestingly, results indicate that this framework has better performance with increased amount of epidemic sources and lower infection rate. While the latter is intuitive and has been previously established, the former is an interesting finding of this works and is possibly related to the GNN approach taken by the proposed framework.

5.1 Future work

There is still plenty of unexplored areas in this work and on the MSD problem as a whole. The reader might have observed that experiments only considered SI epidemic model, and thus there are many other epidemic models left to explore. Although developing models that can be applied to a specific real scenario is challenging, it is common that the underlying epidemic model is not known beforehand when trying to identify epidemic sources. Thus, training the proposed framework with different epidemic models could improve its applicability.

There are also other types of GNN that could be tested as embedding model for source identification. Below is a short list of possible future works for this thesis:

- Evaluate the performance of the framework when the number of sources is increased to larger values (beyond 15) as this model obtained better results with larger number of sources.

- Consider other epidemic models with different states and behavior, such as SIR, IC and SIS. In particular, train the framework with datasets of different epidemic models to evaluate its capacity to generalize.

- Enhance the node attributes using other methods such as label propagation and possibly other epidemic information.

References

[1] FIRTH, J., HELLEWELL, J., KLEPAC, P., et al. "Using a real-world network to model localized COVID-19 control strategies", *Nature Medicine*, v. 26, 10 2020. doi: 10.1038/s41591-020-1036-8.

[2] EPASTO, A., PEROZZI, B. "Innovations in Graph Representation Learning". 2019. Disponível em: <`https://ai.googleblog.com/2019/06/innovations-in-graph-representation.html`>.

[3] BRONSTEIN, M. M., BRUNA, J., COHEN, T., et al. "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges". 2021. Disponível em: <`https://arxiv.org/abs/2104.13478`>.

[4] KIPF, T. N., WELLING, M. "Semi-Supervised Classification with Graph Convolutional Networks". 2016. Disponível em: <`https://arxiv.org/abs/1609.02907`>.

[5] HAMILTON, W. L., YING, R., LESKOVEC, J. "Inductive Representation Learning on Large Graphs". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, p. 1025–1035, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN: 9781510860964.

[6] HAMILTON, W. L. "Graph Representation Learning", *Synthesis Lectures on Artificial Intelligence and Machine Learning*, v. 14, n. 3, pp. 1–159, 2020.

[7] BARABÁSI, A.-L., PÓSFAI, M. *Network science*. Cambridge, Cambridge University Press, 2016. ISBN: 9781107076266 1107076269. Disponível em: <`http://barabasi.com/networksciencebook/`>.

[8] JIANG, J., WEN, S., YU, S., et al. "Identifying Propagation Sources in Networks: State-of-the-Art and Comparative Studies", *IEEE Communications Surveys and Tutorials*, v. 19, n. 1, pp. 465–481, 2017. doi: 10.1109/COMST.2016.2615098.

[9] ZANG, W., ZHANG, P., ZHOU, C., et al. "Locating multiple sources in social networks under the SIR model: A divide-and-conquer approach", *Journal of Computational Science*, v. 10, pp. 278–287, 2015. ISSN: 1877-7503. doi: https://doi.org/10.1016/j.jocs.2015.05.002. Disponível em: <`https://www.sciencedirect.com/science/article/pii/S1877750315000824`>.

[10] DONG, M., ZHENG, B., QUOC VIET HUNG, N., et al. "Multiple Rumor Source Detection with Graph Convolutional Networks". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, p. 569–578, New York, NY, USA, 2019. Association for Computing Machinery. ISBN: 9781450369763. doi: 10.1145/3357384.3357994. Disponível em: <`https://doi.org/10.1145/3357384.3357994`>.

[11] SHELKE, S., ATTAR, V. "Source detection of rumor in social network – A review", *Online Social Networks and Media*, v. 9, pp. 30–42, 2019. ISSN: 2468-6964. doi: https://doi.org/10.1016/j.osnem.2018.12.001. Disponível em: <`https://www.sciencedirect.com/science/article/pii/S2468696418300934`>.

[12] SHAH, D., ZAMAN, T. "Rumors in a Network: Who's the Culprit?" *IEEE Transactions on Information Theory*, v. 57, n. 8, pp. 5163–5181, 2011. doi: 10.1109/TIT.2011.2158885.

[13] PRAKASH, B. A., VREEKEN, J., FALOUTSOS, C. "Spotting Culprits in Epidemics: How Many and Which Ones?" In: *2012 IEEE 12th International Conference on Data Mining*, pp. 11–20, 2012. doi: 10.1109/ICDM.2012.136.

[14] WANG, Z., WANG, C., PEI, J., et al. "Multiple Source Detection without Knowing the Underlying Propagation Model", *Proceedings of the AAAI Conference on Artificial Intelligence*, v. 31, n. 1, Feb. 2017. doi: 10.1609/aaai.v31i1.10477. Disponível em: <`https://ojs.aaai.org/index.php/AAAI/article/view/10477`>.

[15] MIKOLOV, T., CHEN, K., CORRADO, G., et al. "Efficient Estimation of Word Representations in Vector Space". 2013. Disponível em: <`https://arxiv.org/abs/1301.3781`>.

[16] PEROZZI, B., AL-RFOU, R., SKIENA, S. "DeepWalk: Online Learning of Social Representations", *CoRR*, v. abs/1403.6652, 2014. Disponível em: <`http://arxiv.org/abs/1403.6652`>.

[17] MIKOLOV, T., CHEN, K., CORRADO, G. S., et al. "Efficient Estimation of Word Representations in Vector Space". 2013. Disponível em: <http://arxiv.org/abs/1301.3781>.

[18] GROVER, A., LESKOVEC, J. "node2vec: Scalable Feature Learning for Networks", *CoRR*, v. abs/1607.00653, 2016. Disponível em: <http://arxiv.org/abs/1607.00653>.

[19] RIBEIRO, L. F., SAVERESE, P. H., FIGUEIREDO, D. R. "Struc2vec: Learning Node Representations from Structural Identity". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, p. 385–394, New York, NY, USA, 2017. Association for Computing Machinery. ISBN: 9781450348874. doi: 10.1145/3097983.3098061. Disponível em: <https://doi.org/10.1145/3097983.3098061>.

[20] NARAYANAN, A., CHANDRAMOHAN, M., CHEN, L., et al. "subgraph2vec: Learning Distributed Representations of Rooted Sub-graphs from Large Graphs", *CoRR*, v. abs/1606.08928, 2016. Disponível em: <http://arxiv.org/abs/1606.08928>.

[21] SCARSELLI, F., GORI, M., TSOI, A. C., et al. "The Graph Neural Network Model", *IEEE Transactions on Neural Networks*, v. 20, n. 1, pp. 61–80, 2009. doi: 10.1109/TNN.2008.2005605.

[22] BERNOULLI, D. "Essai d'une nouvelle analyse de la mortalite causee par la petite verole, et des avantages de l'inoculation pour la prevenir", *Histoire de l'Acad., Roy. Sci. (Paris) avec Mem*, pp. 1–45, 1760. Disponível em: <https://cir.nii.ac.jp/crid/1571417125051275776>.

[23] KEPHART, J., WHITE, S. "Directed-graph epidemiological models of computer viruses". In: *Proceedings. 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 343–359, 1991. doi: 10.1109/RISP.1991.130801.

[24] WANG, Y., WEN, S., XIANG, Y., et al. "Modeling the Propagation of Worms in Networks: A Survey", *IEEE Communications Surveys and Tutorials*, v. 16, n. 2, pp. 942–960, 2014. doi: 10.1109/SURV.2013.100913.00195.

[25] DOERR, B., FOUZ, M., FRIEDRICH, T. "Why Rumors Spread so Quickly in Social Networks", *Commun. ACM*, v. 55, n. 6, pp. 70–75, jun 2012. ISSN: 0001-0782. doi: 10.1145/2184319.2184338. Disponível em: <https://doi.org/10.1145/2184319.2184338>.

[26] DEMERS, A., GREENE, D., HAUSER, C., et al. "Epidemic Algorithms for Replicated Database Maintenance". In: *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '87, p. 1–12, New York, NY, USA, 1987. Association for Computing Machinery. ISBN: 089791239X. doi: 10.1145/41840.41841. Disponível em: <`https://doi.org/10.1145/41840.41841`>.

[27] DAVID, E., JON, K. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. USA, Cambridge University Press, 2010. ISBN: 0521195330.

[28] KERMACK, W. O., MCKENDRICK, A. G., WALKER, G. T. "A contribution to the mathematical theory of epidemics", *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, v. 115, n. 772, pp. 700–721, 1927. doi: 10.1098/rspa.1927.0118. Disponível em: <`https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1927.0118`>.

[29] KERMACK, W. O., MCKENDRICK, A. G., WALKER, G. T. "Contributions to the mathematical theory of epidemics; The problem of endemicity", *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, v. 138, n. 834, pp. 55–83, 1932. doi: 10.1098/rspa.1932.0171. Disponível em: <`https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1932.0171`>.

[30] KERMACK, W. O., MCKENDRICK, A. G., WALKER, G. T. "Contributions to the mathematical theory of epidemics. III.&#x2014;Further studies of the problem of endemicity", *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, v. 141, n. 843, pp. 94–122, 1933. doi: 10.1098/rspa.1933.0106. Disponível em: <`https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1933.0106`>.

[31] KEMPE, D., KLEINBERG, J., TARDOS, E. "Maximizing the Spread of Influence through a Social Network". In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, p. 137–146, New York, NY, USA, 2003. Association for Computing Machinery. ISBN: 1581137370. doi: 10.1145/956750.956769. Disponível em: <`https://doi.org/10.1145/956750.956769`>.

[32] LIU, Y., LI, W., YANG, C., et al. "Multi-source detection based on neighborhood entropy in social networks", *Scientific Reports*, v. 12, 03 2022. doi: 10.1038/s41598-022-09229-2.

[33] FEY, M., LENSSEN, J. E. "Fast Graph Representation Learning with PyTorch Geometric". In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[34] PASZKE, A., GROSS, S., MASSA, F., et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: Wallach, H., Larochelle, H., Beygelzimer, A., et al. (Eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. Disponível em: <`http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`>.

[35] HAGBERG, A. A., SCHULT, D. A., SWART, P. J. "Exploring Network Structure, Dynamics, and Function using NetworkX". In: Varoquaux, G., Vaught, T., Millman, J. (Eds.), *Proceedings of the 7th Python in Science Conference*, pp. 11 – 15, Pasadena, CA USA, 2008.

[36] ROSSETTI, G., MILLI, L., RINZIVILLO, S., et al. "NDlib: a Python Library to Model and Analyze Diffusion Processes Over Complex Networks", *CoRR*, v. abs/1801.05854, 2018. Disponível em: <`http://arxiv.org/abs/1801.05854`>.

[37] ERDÖS, P., RÉNYI, A. "On Random Graphs I", *Publicationes Mathematicae Debrecen*, v. 6, pp. 290, 1959.

[38] GILBERT, E. N. "Random Graphs", *The Annals of Mathematical Statistics*, v. 30, n. 4, pp. 1141 – 1144, 1959. doi: 10.1214/aoms/1177706098. Disponível em: <`https://doi.org/10.1214/aoms/1177706098`>.

[39] LESKOVEC, J., MCAULEY, J. "Learning to Discover Social Circles in Ego Networks". In: Pereira, F., Burges, C., Bottou, L., et al. (Eds.), *Advances in Neural Information Processing Systems*, v. 25. Curran Associates, Inc., 2012. Disponível em: <`https://proceedings.neurips.cc/paper/2012/file/7a614fd06c325499f1680b9896beedeb-Paper.pdf`>.

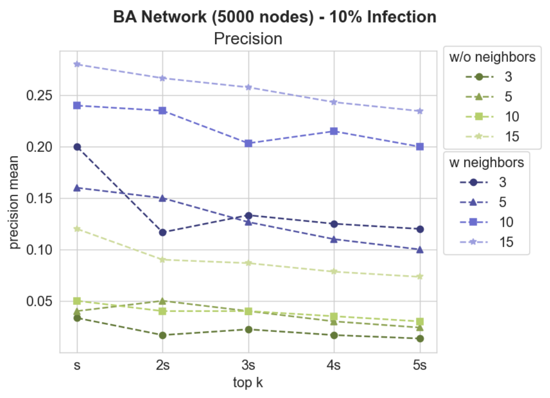
[40] WATTS, D. J., STROGATZ, S. H. "Collective dynamics of 'small-world' networks", *Nature*, v. 393, n. 6684, pp. 440–442, 1998. doi: 10.1038/30918.

Appendix A

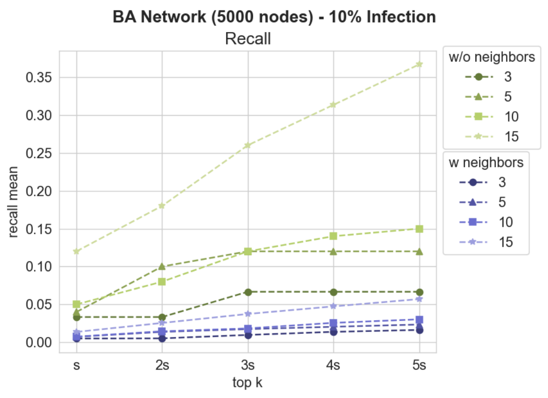
Results for different infection rates

A.1 Artificial networks

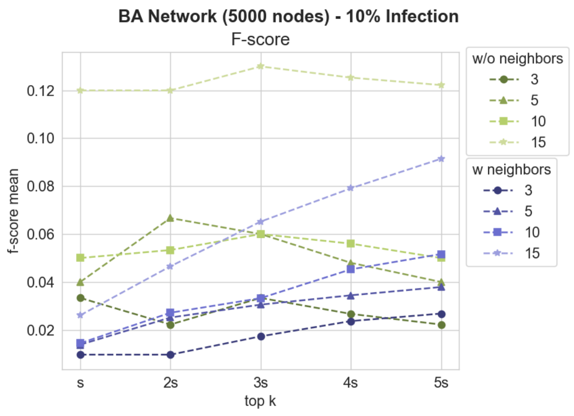
A.1.1 BA network



(a) Precision

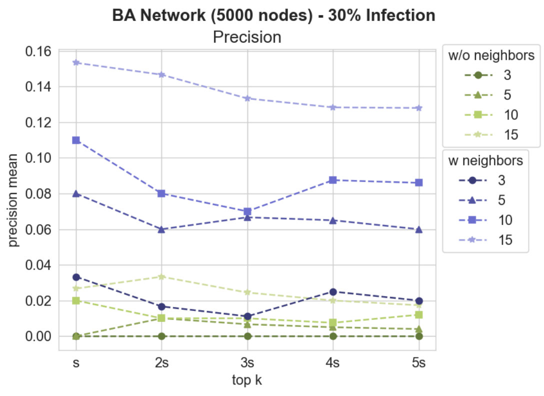


(b) Recall

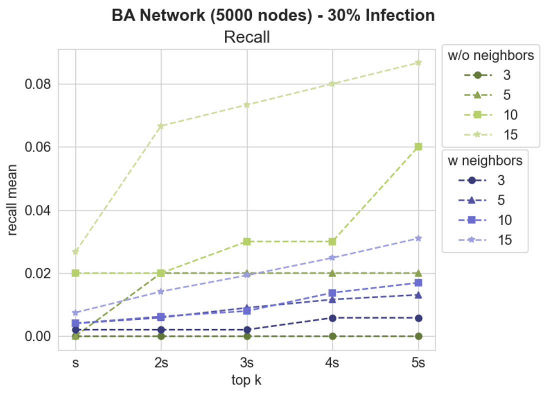


(c) F-score

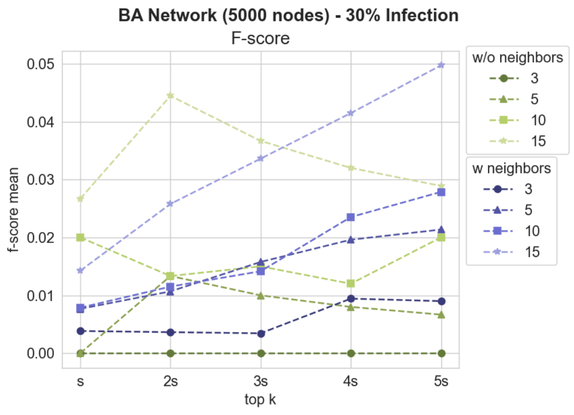
Figure A.1: Results for source identification considering the BA network when 10% of them are infected.



(a) Precision



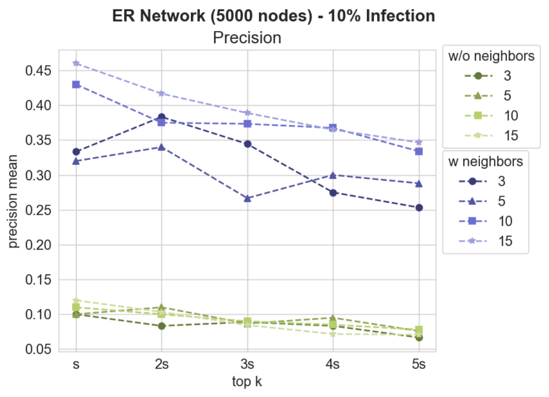
(b) Recall



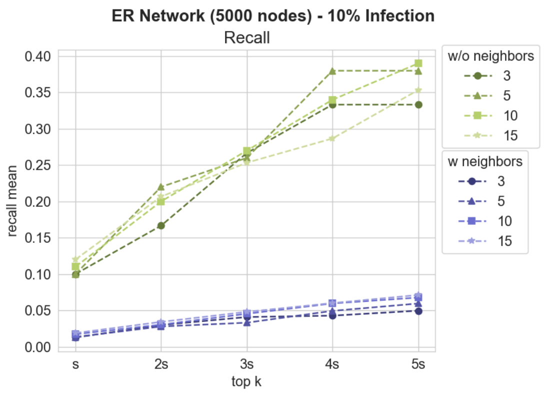
(c) F-score

Figure A.2: Results for source identification considering the BA network when 30% of them are infected.

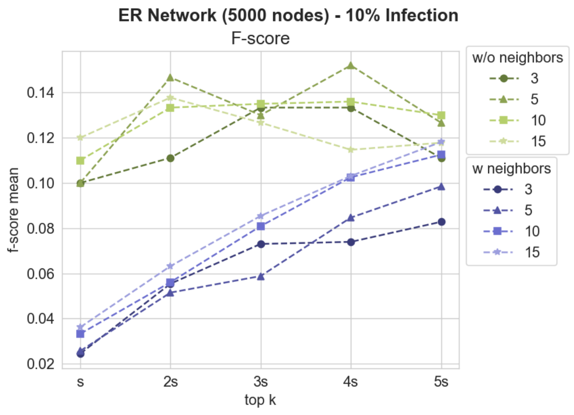
A.1.2 ER network



(a) Precision

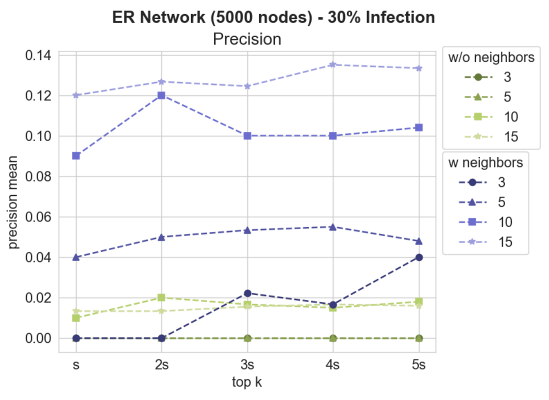


(b) Recall

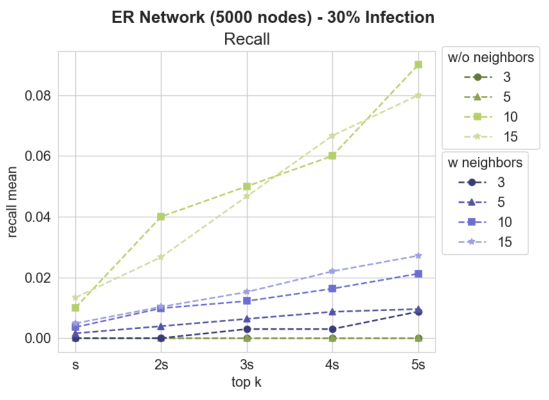


(c) F-score

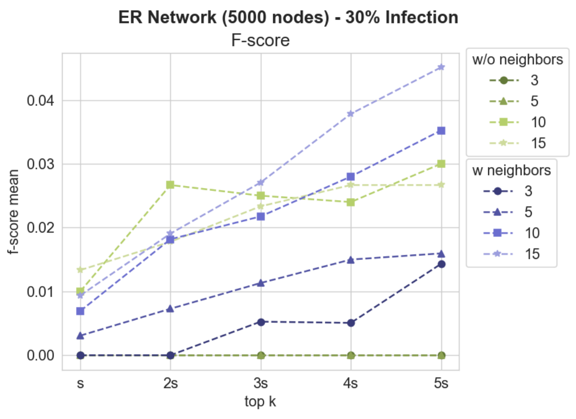
Figure A.3: Results for source identification considering the ER network when 10% of them are infected.



(a) Precision



(b) Recall

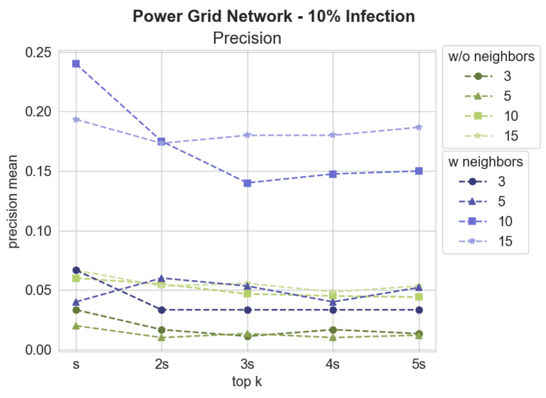


(c) F-score

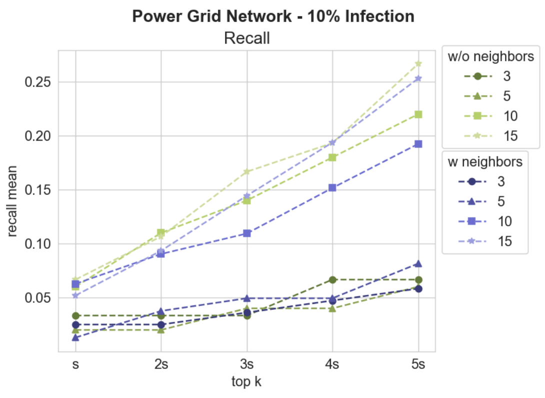
Figure A.4: Results for source identification considering the ER network when 30% of them are infected.

A.2 Real Networks

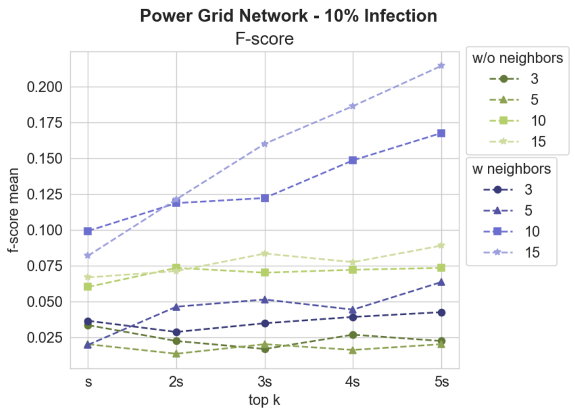
A.2.1 Power grid network



(a) Precision

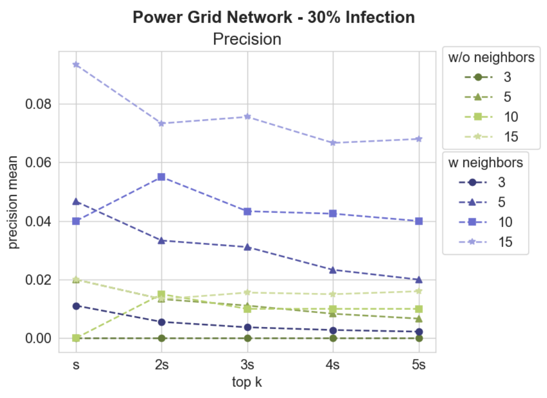


(b) Recall

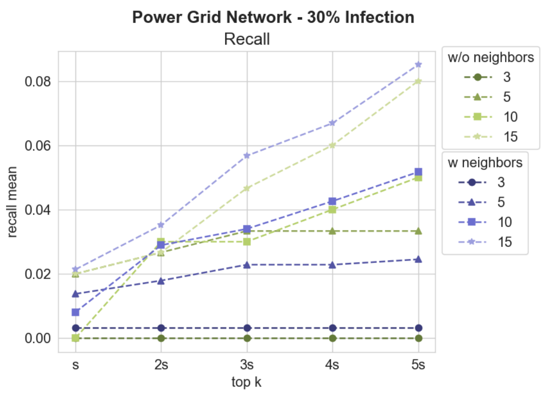


(c) F-score

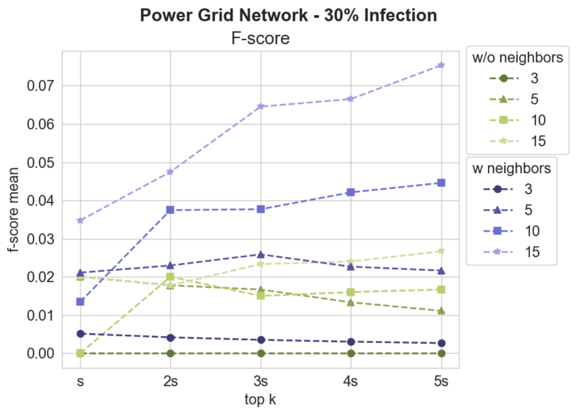
Figure A.5: Results for source identification considering the Power grid network when 10% of them are infected.



(a) Precision



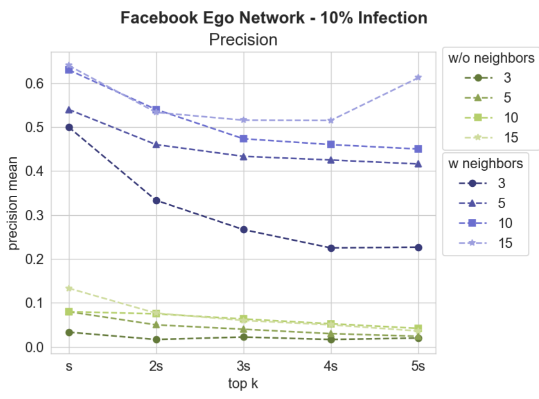
(b) Recall



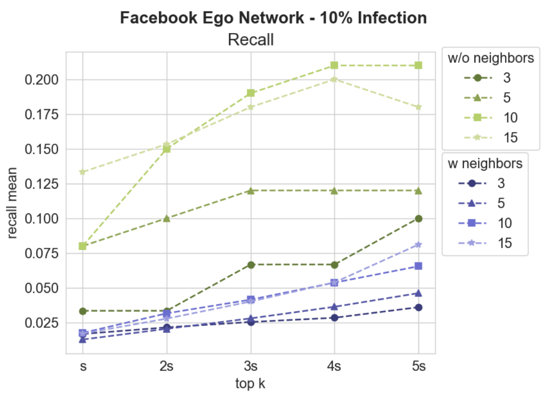
(c) F-score

Figure A.6: Results for source identification considering the Power grid network when 30% of them are infected.

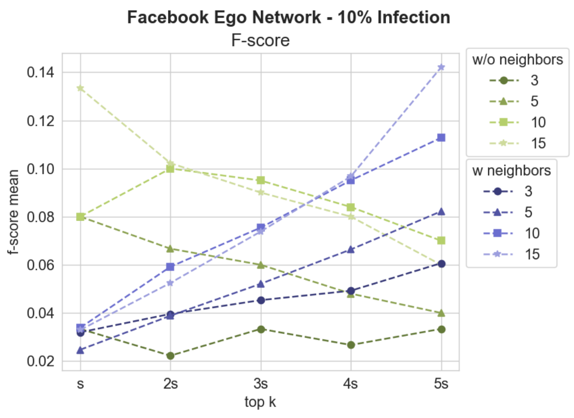
A.2.2 Facebook ego network



(a) Precision

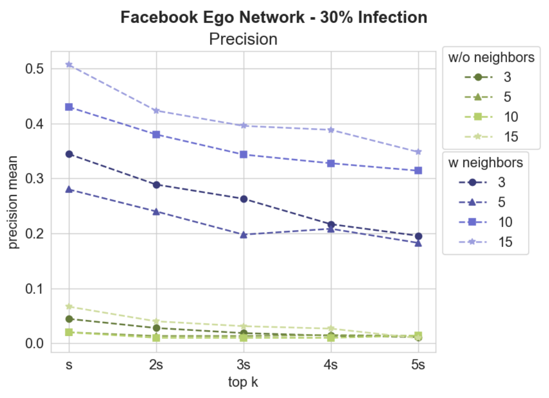


(b) Recall

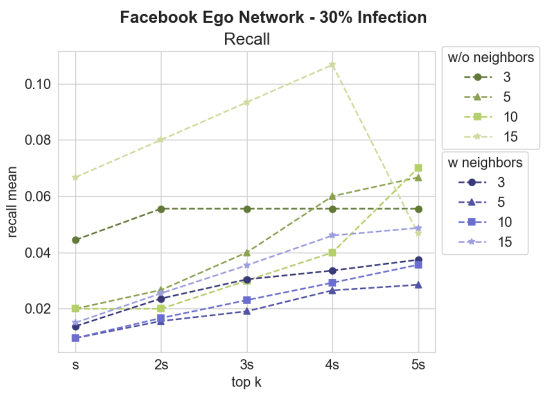


(c) F-score

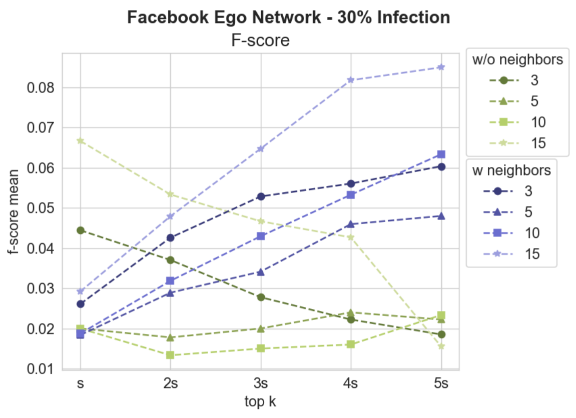
Figure A.7: Results for source identification considering the Facebook ego network when 10% of them are infected.



(a) Precision



(b) Recall



(c) F-score

Figure A.8: Results for source identification considering the Facebook ego network when 30% of them are infected.