School of Electrical and Computer Engineering
Technical University of Crete

Recommendation driven opinion de-polarization in social networks

Diploma Thesis
by

Konstantinos Mylonas
2018030151

DIPLOMA
IN
ELECTRICAL AND COMPUTER ENGINEERING

Chania, Greece

October, 2024

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Πολυτεχνείο Κρήτης

Μείωση Πόλωσης στα Κοινωνικά Δίκτυα Μέσω Προτεινόμενου Περιεχομένου

Διπλωματική Εργασία
από

Κωνσταντίνο Μυλωνά
2018030151

ΔΙΠΛΩΜΑ
ΗΛΕΚΤΡΟΛΟΓΟΥ ΜΗΧΑΝΙΚΟΥ ΚΑΙ ΜΗΧΑΝΙΚΟΥ
ΥΠΟΛΟΓΙΣΤΩΝ

Χανιά, Ελλάδα

Οκτώβριος 2024

Abstract

The Internet has acquired its role in the everyday life of people. Social media are now the ground for political debate and exchange of opinions. Initially, we might think that this over exposure to information would lead to open-minded, inclusive societies with less polarized members. However, there is a significant amount of work that suggests the opposite. People tend to organise into groups and communities that share common beliefs and interact with each other. Such communities are known as echo chambers. This phenomenon is known as homophily and has been studied for years by sociologists. Inside an echo chamber environment, the pre-existing beliefs of individuals are reinforced and the overall polarisation in the social network increases, which have negative impact to our society. In order to de-polarise the network, one can try to convince a small set of network members (users) to adopt more moderate positions around a topic. However, choosing a proper set of users is not an easy task, especially in the modern social networks that consist of billions of users. Should we focus on users with extreme opinions, but few connections? Or should we focus on less extreme, but famous, users, hoping that their neutral views will eventually reach and affect a larger fraction of the overall network? In this diploma thesis, we propose an efficient algorithm to choose a set of users from a network, such that when their opinions are moderated the overall polarisation of the network is reduced significantly, based on the famous Friedkin and Johnsen opinion formation model. We use Graph Neural Networks - specifically a Graph Convolutional Network - in order to create an algorithm able to work with large graphs. We compare our algorithm with a greedy algorithm, named GreedyExt, which has been proposed in the past. Our results show that our algorithm is much faster than GreedyExt and achieves similar performance in terms of depolarisation. We evaluate our algorithm in both synthetic and real graphs with ground truth communities.

Περίληψη

Το Διαδίκτυο έχει αποκτήσει πλέον σημαντικό ρόλο στην καθημερινή ζωή των ανθρώπων. Τα μέσα κοινωνικής δικτύωσης αποτελούν πλέον το έδαφος για πολιτική συζήτηση και ανταλλαγή απόψεων. Αρχικά, θα μπορούσε κανείς να υποθέσει ότι αυτή η υπερέκθεση στην πληροφορία θα οδηγούσε σε ανοιχτόμυαλες, χωρίς πόλωση κοινωνίες. Ωστόσο, υπάρχει σημαντικός όγκος ερευνών που υποστηρίζουν το αντίθετο. Οι άνθρωποι τείνουν να οργανώνονται σε ομάδες και κοινότητες που μοιράζονται κοινές πεποιθήσεις και αλληλεπιδρούν μεταξύ τους. Τέτοιες κοινότητες είναι γνωστές ως echo chambers. Το φαινόμενο αυτό είναι γνωστό ως ομοφιλία και έχει μελετηθεί για χρόνια από κοινωνιολόγους. Μέσα στο περιβάλλον ενός echo chamber, οι προϋπάρχουσες πεποιθήσεις των ατόμων ενισχύονται και η συνολική πόλωση στο κοινωνικό δίκτυο αυξάνεται, γεγονός που έχει αρνητικές συνέπειες για την κοινωνία μας. Για να μειωθεί η πόλωση στο δίκτυο, μπορούμε να προσπαθήσουμε να πείσουμε ένα μικρό σύνολο χρηστών να υιοθετήσουν πιο ουδέτερες θέσεις πάνω σε ένα θέμα. Ωστόσο, η επιλογή του κατάλληλου συνόλου χρηστών δεν είναι εύκολη υπόθεση, ειδικά στα σύγχρονα κοινωνικά δίκτυα που αποτελούνται από δισεκατομμύρια χρήστες. Είναι καλύτερο να επικεντρωθούμε σε χρήστες με ακραίες απόψεις αλλά λίγες συνδέσεις; Ή να εστιάσουμε σε λιγότερο ακραίους αλλά διάσημους χρήστες, ελπίζοντας ότι οι ουδέτερες απόψεις τους θα φτάσουν και θα επηρεάσουν μεγαλύτερο μέρος του συνολικού δικτύου; Σε αυτή τη διπλωματική εργασία, προτείνουμε έναν αποδοτικό αλγόριθμο για την επιλογή ενός συνόλου χρηστών από ένα δίκτυο, ώστε όταν μετριαστούν οι απόψεις τους, να μειωθεί σημαντικά η συνολική πόλωση του δικτύου, βασιζόμενοι στο γνωστό μοντέλο διαμόρφωσης απόψεων των Friedkin και Johnsen. Χρησιμοποιούμε Graph Neural Networks - συγκεκριμένα ένα Graph Convolutional Network - για τη δημιουργία ενός αλγορίθμου ικανού να λειτουργεί με μεγάλα γραφήματα. Συγκρίνουμε τον αλγόριθμό μας με έναν άπληστο αλγόριθμο, τον GreedyExt, που έχει προταθεί στο παρελθόν. Τα αποτελέσματά μας δείχνουν ότι ο αλγόριθμός μας είναι πολύ πιο γρήγορος από τον GreedyExt και επιτυγχάνει παρόμοια απόδοση όσον αφορά την αποπόλωση. Αξιολογούμε τον αλγόριθμό μας τόσο σε συνθετικούς όσο και σε πραγματικούς γράφους.

Acknowledgement

I would like to express my gratitude to my thesis supervisor **Prof. Thrasyvoulos Spyropoulos** for his guidance, encouragement and understanding. His mentorship played a pivotal role in steering me in the right direction during challenging times. Moreover, I am indebted to him for inspiring me to delve into a subject that has now become my passion. Furthermore, I would like to express my appreciation to my family for their unwavering support throughout all these years. Lastly, and of utmost importance, I would like to thank all my friends for being present throughout this journey in every possible way. You transformed this challenging experience into a more positive and enjoyable one. This thesis would not have been possible without the contribution of all those mentioned above.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the contemporary digital landscape, dominated by platforms like Instagram, Facebook, and TikTok, users find themselves immersed in an ever-expanding sea of content [2]. Despite the sheer volume, the experience of users usually expands to limited content. The main reason behind this phenomenon lies in the recommendation algorithms employed by these platforms. Those algorithms capture user preferences and interactions and are designed to recommend personalized content. Combined with homophily - the tendency of people to interact with similar minded people - this fact leads to a significant decrease in the variety of posts that appear in users feed.

This narrowing of exposure leads to a broader issue of polarization, where users organize themselves into isolated, ideologically homogeneous communities, commonly referred to as echo chambers. Within these echo chambers, individuals' pre-existing beliefs are reinforced, often leading to extremism, fanaticism, and a diminished openness to alternative viewpoints. Addressing this challenge has become a central focus in social network analysis, with several studies proposing methods to reduce polarization by recommending friends from outside the echo chamber [3] or suggesting diverse content [4].

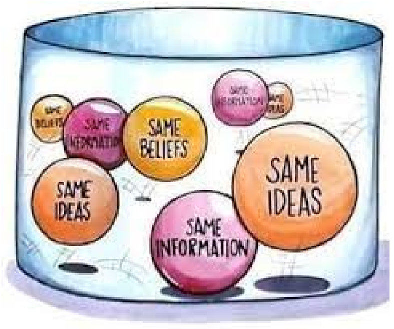


Figure 1.1: The problem of echo chambers

However, these approaches do not explicitly account for users' opinions or quantify the level of polarization in the network. They primarily focus on diversifying content recommendations without a deeper consideration of the underlying opinion dynamics.

In this thesis, we tackle polarization from a different perspective, utilizing an established opinion dynamics model: the Friedkin and Johnsen model. By assigning numerical opinions to each user, we can quantify polarization more precisely. Our objective is to identify a set of users whose opinion moderation would have the most significant impact on reducing overall network polarization. This moderation is not necessarily achieved through content or friend recommendations. Instead, we take an abstract approach, assuming we have the means, through various interventions (such as education campaigns [5]), to convince certain individuals to adopt more neutral or moderate opinions on contentious issues. Given the reality of limited resources, the set of users that we choose has to be carefully selected.

This optimization problem extends beyond social media networks, applying to real-world communities, such as immigrant populations or minority groups, where we aim to influence opinions on topics like vaccination. It is common for such communities to exhibit hesitancy towards important topics, particularly those related to public health, such as vaccination [6]. This reluctance can stem from a lack of understanding, cultural beliefs, or distrust in authorities. Successfully engaging these groups is critical not only for public health initiatives but also for fostering their broader integration into society. Addressing these concerns through targeted education and awareness campaigns can be key to promoting inclusivity and encouraging informed decision-making.

The ability to make informed, strategic decisions about which individuals to engage with is critical in maximizing the impact of any effort to reduce polarization within these networks. However, choosing the right set of individuals is a challenging task, especially in large networks like the modern social networks. In this thesis, we model networks as graphs and we adopt a machine learning approach that relies on Graph Neural Networks - neural network architectures designed to work with graphs - to help us identify a good set of users to be targeted. We propose an algorithm that performs better than state-of-the-art heuristics and, scales to large graphs.

1.1 Definitions

Social Networks are those web-based services that allow individuals to (1) construct a public or semi-public profile within a bounded system, (2) articulate a list of other users with whom they share a connection, and (3) view and traverse their list of connections

Figure 1.2: Famous Social Networks

and those made by others within the system [7].

These networks can represent a wide range of social interactions, from friendships and collaborations to information exchanges and influence dynamics. The study of social networks helps us understand how groups of people interact, form communities, and share resources or ideas. By analyzing the patterns and strength of connections within a network, researchers can gain insights into phenomena like opinion formation, the spread of information, and the role of influential individuals in shaping collective behavior.

Formally, we model a social network as an undirected graph $G(V, E)$ where V is the set of nodes (users) and $E \subseteq V \times V$ is the set of edges between them. An edge between user i and user j can denote for example that users i and j are friends in Facebook. Often each edge is associated with a weight w_{ij} to capture how strong or weak a connection between users might be.

An **Echo Chamber** [8] is a social environment, often seen in online spaces, where individuals are exposed primarily to opinions and information that reinforce their existing beliefs. Within these spaces, opposing viewpoints are either absent or actively filtered out, creating a closed loop of similar ideas. This can lead to a distortion of reality, as people become more entrenched in their perspectives without being challenged by alternative viewpoints. Echo chambers are closely related to the concept of filter bubbles, where algorithms on social media platforms or search engines personalize content, further limiting exposure to diverse information. Figure 1.3 shows an example of a social network with dual echo chamber. Specifically, this network comes from the greek political discussion on Twitter. We found this image at Political Lighthouse Datalab .

Echo chambers and filter bubbles can have significant negative impacts on both individuals and society as a whole. By limiting exposure to diverse perspectives, they can lead to a more polarized and fragmented public discourse. Individuals within these environments may become more resistant to new information or alternative viewpoints, reinforcing biases and making constructive debate difficult. Over time, this isolation from differing opinions can lead to misinformation and extremist views gaining traction, as there's less opportunity for critical evaluation or challenge. In a broader sense, echo

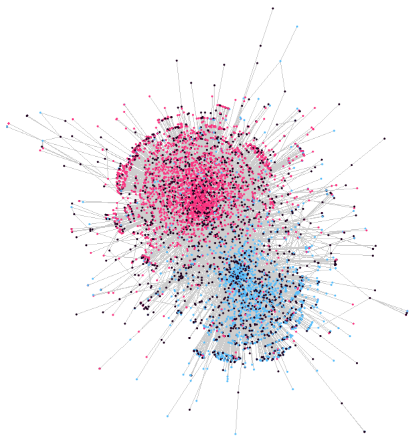


Figure 1.3: An example of a Dual Echo Chamber Network

chambers can undermine democratic processes, as informed decision-making relies on access to a wide range of ideas and perspectives.

Figure 1.4: Effect of Echo Chambers

An **Opinion Formation Model** is a theoretical framework used to understand how individuals within a social network form, change, and influence each other's opinions over time. These models typically simulate interactions between individuals, where opinions evolve based on factors such as peer influence, external information, or personal biases. By capturing the dynamics of opinion shifts, these models help us study how consensus, polarization, or fragmentation occurs in a society. They provide insights into real-world phenomena, such as how trends, political ideologies, or social norms spread through populations, and the impact of influential figures or echo chambers on public opinion.

One of the most common opinion models was introduced by DeGroot [9]. According to this model every individual has a numerical opinion $z_i \in \mathbb{R}$ that depends on the opinions of his connections (friends, coworkers, followers etc.). More precisely according to DeGroot:

$$z_i = \frac{\sum_{j \in N(i) \cup i} w_{ij} z_j}{\sum_{j \in N(i) \cup i} w_{ij}} \tag{1.1}$$

where $N(i)$ represents the set of users that user i is connected with (neighbors), and w_{ij} is the weight of each connection that encodes the influence that user j exerts on user i. In simple words, the weight captures the significance of each opinion in the averaging process. High value of w_{ij} means that the two users influence each others opinions. In a real scenario they could be relatives, or good friends and affect each others viewpoints due to their strong personal relationship. The averaging process usually converges to consensus, a state where all users share the exact same opinion as it was discussed in the original paper [9].

DeGroot's model has been studied extensively and it is one of the most famous models when it comes to opinion dynamics. However, the fact that it converges to consensus is usually not true in real social networks as stated in [10]. The model suggests that the users update their opinions through the averaging process of Equation 1.1, but it ignores the fact that in reality opinions are also affected by factors from outside the network. This of course, doesn't mean that the idea of averaging opinions is incorrect or irrational. Our intuition agrees with this process. For this reason in this work we will use a variation of this model which doesn't reach consensus in most cases. The model that we use is known as Friedkin and Johnsen (FK) [10], named after the researchers that developed it.

Friedkin and Johnsen in [10] introduce a family of flexible opinion models that account for exogenous factors. The exact model that we employee is the one that was used by the authors of [5]. According to this model each user maintains a persistent, internal opinion s_i. This opinion remains constant throughout the averaging process. The internal opinion tries to capture the fact that people's opinion not only depends on the opinions of their environment but it depends on internal strong held beliefs as well. Such beliefs come from the individuals education, background, origin and political orientations. So, according to this model, every user has two opinions: her internal opinion s_i that remains constant and her external opinion z_i that is updated through an averaging process, similar to DeGroot. z_i is essentially the expressed opinion of user i, and it is affected both by s_i and the expressed opinions of $N(i)$. The formal definition of the model is provided in Chapter 3.

1.2 Problem Description

Numerous techniques have been proposed to deal with the impacts of echo chambers [11] [12] [3]. Some of them include recommending new friendships (links) between users of opposing echo chambers, while others suggest tweaking recommendation algorithms to serve diverse content to individuals. In this thesis we will deal with the dual echo chamber - two well structured and connected communities of the network that are barely connected with each other. Each chamber represents a set of users that share common opinions and the information flow from one community to the other is naturally restricted due to the small amount of links between the two.

This is a combinatorial optimisation problem and a naive algorithm would fail to work even with small graphs, because of the high complexity. In fact the problem is NP-hard. The proof can be found in [5], but in summary it comes from a reduction from the *m-SubsetSum* problem.

The formal definition of the problem that we are solving is given in Chapter 4, but we provide some high level description here. We assume that we have a graph in hands that requires depolarisation. Furthermore, we are given an integer $K << |V|$. K is the number of nodes in the graph that we can fix their external opinion z to zero. K has the role of a budget. So given this graph and the budget, which K users should we pick and set their opinion to zero, in order to minmise the polarisation in the network? This question describes the problem of interest. We discussed earlier some possible means to convince users to adopt a neutral opinion, like educational campaigns or online targeting with specific articles [5]. But, we don't particularly deal with the "how to convince someone to change their opinion". Assuming that this can be done, we seek for a good choice of K users, in order to minimise the overall polarisation in the network, based on a measure that we discuss in Chapter 3. We will refer to this process of setting the opinion of user $z_i = 0$ as "targeting node i", "neutralizing" or "moderating". All these terms are used interchangeably throughout this text.

This problem was first introduced in [5] and its name is *ModerateExpressed*, because the expressed opinions of some users are moderated. In the same paper they proposed a greedy algorithm to solve the problem. This algorithm is named *GreedyExt*. *GreedyExt* is an iterative algorithm that makes K iterations over the set of nodes, and at each iteration it greedily picks the node that minimises polarisation. Our approach doesn't deviate much from *GreedyExt*. The latter showed great results in terms of de-polarisation, meaning that the solution set (the K nodes) that the algorithm picks, lead to lower polarisation than the baseline algorithms that it was compared to. Unfortunately, *GreedyExt* has one downside. It doesn't scale to larger graphs. *GreedyExt* and *ModerateExpressed* are defined formally in Chapter 4.

1.3 Solution Description

Our approach doesn't deviate much from the algorithm that we just discussed. In fact our solution is also a greedy algorithm that iteratively picks the node that it thinks will yield the best results (minimise polarisation at the new equilibrium), when its opinion is set to zero. The difference comes from the way that the nodes are evaluated and selected. While *GreedyExt* had to examine the resulting polarisation after setting **every** node to zero - a process that requires computing the steady state vector $\mathbf{z}$ - and select the best one, our algorithm employees a Graph Neural Network based architecture that predicts the polarisation of the graph, after setting each node to zero, and selects the best one. The result is an algorithm that makes equivalent choices to the choices of *GreedyExt* but the time required is dramatically less. This allows our algorithm to scale to large graphs, which otherwise was highly inefficient. The main benefit of our approach is that it ourperforms *GreedyExt* in terms of time, while it achieves almost identical results in terms of final polarisation.

1.4 Contributions

The main contribution of this thesis is an algorithm that solves the *ModerateExpressed* problem with similar perfomance as *GreedyExt*, and it scales to large graphs as well. Specifically:

- (Offline Algorithm) We train a GNN-based architecture that learns the importance of nodes in different graph topologies in terms of affecting the network's polarization, based on mostly "local" characteristics, hence not requiring full knowledge of the graph.
- (Online Algorithm) We use the trained GNN to devise an iterative algorithm to pick the set of K nodes to "moderate". Our algorithm *does not* require the complexity of analytically solving the graph, as required by *GreedyExt* [5], which evaluates the importance of each node by simulating the scenario in which this node is targeted and calculating the final polarisation.

- Using a range of synthetic and real datasets, we demonstrate that our approach successfully finds a set of K nodes whose de-polarization performance approximates that of the algorithm of [5], but does in significantly less time.

1.5 Thesis Outline

In Chapter 2 we present other works in the context of opinion de-polarisation. Chapter 3 provides the necessary background to formally define the problem and the solution that we suggest. Chapter 4 is dedicated to the formal definition of the problem that we are dealing with in this thesis. In the same chapter we present the algorithm *GreedyExt*. Next, in Chapter 5 we present (a) the methods that we used to generate synthetic graphs that simulate polarised networks and (b) the real networks that we used. Chapter 6 presents the proposed solution and discusses some limitations while Chapter 7 presents the experimental results.

Chapter 2

Related Work

Evolution of Opinions in Social Networks

A significant body of research has focused on studying the evolution of opinions in social networks. Friedkin and Johnsen opinion model [10] is perhaps one of the most acceptable models in the literature. According to this model every user has two opinions: an internal and an external. The internal opinion depends on the background, education, origin and other factors that play an important role into shaping someones personality. It is consider to be fixed. The external opinion is a weighted average between the opinions of someones friends. This model is intuitively rational and has also been validated with real data. It is an extension of another famous model: DeGroot [9]. We use Friedkin and Johnsen model for this thesis because of its realistic nature: Consensus is usually not reached, which is what we observe in modern socail network settings. Another famous opinion model is the Voter Model. The Voter Model is a discrete opinion model where the opinions are updated in rounds and individuals adopt the opinion of one of their connections with some probability [13]. Finally, in the Bounded Confidence Model [14] interactions modify the opinions of the nodes only when they are within a confidence interval $\epsilon \in [0, 1]$ from each other. If they are, when u interacts with v, u moves closer to v's opinion.

Quantifying Polarization in Social Networks

[15] quantifies the polarisation of social networks based on the structure of the underlying graph. [5] adopts an opinion model, where users are tagged with numerical opinions in range $[-1, 1]$, and introduces a measure named polarisation index that successfully identifies polarised networks.

[16] performs a comparison between polarized and non-polarized networks and proposes a metric designed to measure the degree of polarization between two communi-

ties.

In [11] they introduced the concept of disagreement in social networks and the trade-off between disagreement and polarisation. They used the Friedkin and Johnsen opinion dynamics model, just like we do. They aim for a social network that minimises both disagreement and polarisation. Their work is closely related to [5], with the main difference being that they account for disagreement as well. Another paper that uses the Friedkin and Johnsen opinion dynamics model is [17]. In this work they modeled the opinion evolution as a game between users. Their goal was to measure the cost of maintaining personal opinions. The gave this cost a name: Price of Anarchy.

De-polarisation of Social Networks

In addition to quantifying polarization, many works have focused on designing interventions aimed at depolarizing opinions in social networks. One common approach involves the use of content-based recommendations that expose users to diverse or opposing viewpoints in an effort to reduce ideological extremity. In [12] they model the Newsfeed and Wall of every user as a Markov Chain. They suggested adding curated recommendations to the newsfeeds in order to show users more diverse content. The optimal recommendation rate that maximises the average diversity accord- ing to their model, can be found by solving a system.

Another approach centers on friend recommendations, where users are encouraged to connect with individuals who hold different opinions. In [4] they proposed a framework to asses the effects of friend-recommendation systems on the echo chamber and polarisation. They found that friend recommendation systems can strengthen echo chambers under certain conditions, while their effect might be negligible depending on the structure of the underlying network. In the context of friends recommendations, one very interesting work is [3]. They use Graph Convolutional Networks [18] along with NLP methods to construct representations of users and the echo chambers, which they later use to recommend relevant and diverse friendships.

The most closely related work to this thesis is [5], where the authors use the Friedkin and Johnsen opinion model to address the problem of minimizing overall polarization in a network. Specifically, they aim to identify a set of K users whose opinions, when moderated, lead to a reduction in polarization. This thesis is motivated and builds significantly on their approach, tackling the same core problem but with a method that is designed to scale more effectively to larger graphs.

Chapter 3

Background

In this chapter we provide the necessary background that will be used in later sections.

3.1 Friedkin and Johnsen Opinion Model

Here we will present formally the opinion formation model that we briefly discussed in the introduction and we will quickly show the mathematics that lie behind the calculation of the equilibrium [10].

3.1.1 Opinion Model

We are given an undirected weighted graph $G(V, E)$ where V is the set of nodes and E is the set of edges. Each node is assigned an internal opinion s_i, which is constant, and an external opinion z_i. Both the internal and external opinions take values in the range $[-1, 1]$. We consider 0 to be the neutral opinion, while -1 and 1 are the most extreme opinions that an individual can have regarding a topic. Every edge between nodes i and j is associated with a weight w_{ij} The external opinion z_i is updated according to the following rule:

$$z_i = \frac{s_i + \sum_{j \in N(i)} w_{ij} z_j}{1 + \sum_{j \in N(i)} w_{ij}} \tag{3.1}$$

After some rounds of updating, the system reaches equilibrium where the opinion vector **z** remains fixed. The expressed opinion z_i of node i represents a compromise between the persistent value of s_i and the expressed opinions of others to whom i is connected [17]. It has been shown that if every person i updates her opinion based on Equation 3.1, then the expressed opinions converge to a unique opinion vector $\mathbf{z}$, where each component z_i is the opinion of user i in the equilibrium [5].

Consensus is usually not reached which makes the model more realistic and ideal for our work. The reason why we don't want a model that reaches consensus is because in such case, if we set and fix the opinion $z_i = 0$, sooner or later all opinions would converge to zero. An opinion model that leads to consensus makes our problem trivial. Adding to that as we briefly mentioned earlier, consensus is usually not reached in real social networks [10]. We are interested into minimising polarization at **equilibrium**. The steady state vector can be computed by running the evolution process untill convergence, or by solving a system of linear equations. We analyze this method of computing the steady state in the following section.

3.1.2 Equilibrium - Laplacian Matrix

This model was used in [17] where they tried to quantify the cost of the lack of consensus hence the title "How bad is forming your own opinion". They viewed the averaging process as the trajectory of best-response dynamics in a one-shot, complete information game played by the nodes in V, where i's strategy is a choice of opinion z_i. In essence, updating z_i as in Equation 3.1 is the same as choosing z_i to minimize the following quantity:

$$c_i = (z_i - s_i)^2 + \sum_{j \in N(i)} w_{ij}(z_i - z_j)^2$$

The proof of this statement follows in the next page.

Proof. In order to minimize c_i we take it's derivative with respect to z_i and set it to zero.

$$
\begin{aligned}
&\frac{dc_i}{dz_i} = 0 \Longleftrightarrow \\
&2(z_i - s_i) + \sum_{j \in N(i)} w_{ij} 2(z_i - z_j) = 0 \Longleftrightarrow \\
&z_i - s_i + z_i \sum_{j \in N(i)} w_{ij} - \sum_{j \in N(i)} w_{ij} z_j = 0 \Longleftrightarrow \\
&z_i(1 + z_i \sum_{j \in N(i)} w_{ij}) = s_i + \sum_{j \in N(i)} w_{ij} z_j \Longleftrightarrow \\
&z_i = \frac{s_i + \sum_{j \in N(i)} w_{ij} z_j}{1 + z_i \sum_{j \in N(i)} w_{ij}}
\end{aligned}
$$

□

Having said that we can conclude that at equilibrium $c_i^{'} = 0 \; \forall i$. So an alternative way to computing the steady state vector z is solving the corresponding system of equations that hold at equilibrium. This observation will lead to a compact formula for finding z as they propose in [5].

First we need to define the Laplacian Matrix $\mathbf{L}$ of the graph:

$$
\mathbf{L}_{ij} = \begin{cases} \sum_{k \in N(i)} w_{ik}, & \text{if } i = j \\ -w_{ij}, & \text{if } i \neq j \end{cases}
$$

At equilibrium $\forall i$:

$$
\begin{aligned}
&c_i^{'} = 0 \Longleftrightarrow && (3.2) \\
&(z_i - s_i) + \sum_{j \in N(i)} w_{ij}(z_i - z_j) = 0 \Longleftrightarrow && (3.3) \\
&z_i + \sum_{j \in N(i)} w_{ij}(z_i - z_j) = s_i \Longleftrightarrow && (3.4) \\
&z_i + z_i \sum_{j \in N(i)} w_{ij} - \sum_{j \in N(i)} w_{ij} z_j = s_i \Longleftrightarrow && (3.5) \\
&z_i(1 + \sum_{j \in N(i)} w_{ij}) - \sum_{j \in N(i)} w_{ij} z_j = s_i && (3.6)
\end{aligned}
$$

These equations can be written using the weighted Laplacian Matrix as:

$$(\mathbf{L}+\mathbf{I})\mathbf{z}=\mathbf{s} \tag{3.7}$$

The Laplacian Matrix is a positive semi-definite matrix which means that $\mathbf{x^T L x} \geq 0$ $\forall x$. Based on this $\mathbf{L}+\mathbf{I}$ is positive definite.

Proof. $\mathbf{x^T(L+I)x = x^T L x + x^T I x = x^T L x + x^T x = x^T L x + ||x||^2 > 0}$ □

In this case all eignevalues of $\mathbf{L}+\mathbf{I}$ are positive which means that the matrix is invertible. So returning back to (3.7):

$$\mathbf{z}=(\mathbf{L}+\mathbf{I})^{-\mathbf{1}}\mathbf{s} \tag{3.8}$$

Thus, we can use (3.8) to compute the steady state vector $\mathbf{z}$ instead of simulating and running the averaging process.

Note: The same is true for unweighted graph where all weights are equal to one.

3.2 Polarisation Index

Our goal is to minimise polarisation. Before that we need a metric to quantify the polarisation of a network. We will use the metric that was proposed at [5] and was proved capable of capturing the polarisation. The metrics name is *Polarization Index* and we use the notation π.

$$\pi=\frac{||\mathbf{z}||_2^2}{|V|} \tag{3.9}$$

The polarization index is simply the square of the L2 norm of the opinion vector $\mathbf{z}$. In order for the metric to be independent from the size of the graph we divide by the number of the nodes. For example if all users have extreme opinions 1 and -1 then the polarization is going to be large whereas if all users have neutral opinion $z=0$ the polarization index $\pi=0$.

3.3 Graph Neural Networks

Graph Neural Networks (GNNs) are a class of neural networks designed to work directly with graph-structured data. Unlike traditional neural networks that operate on

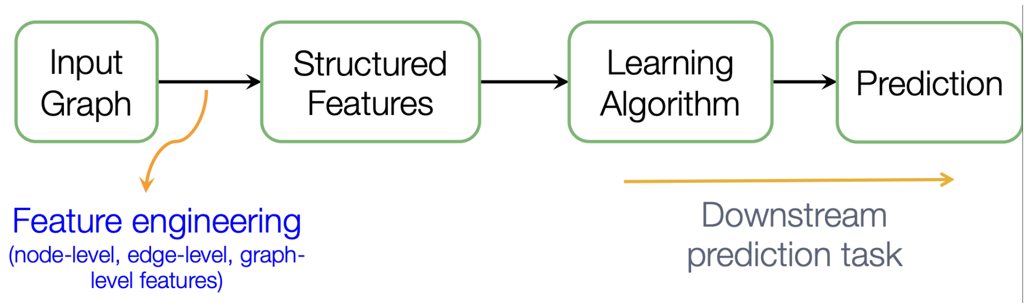


Figure 3.1: Feature Engineering Pipeline [1]

fixed structures like grids or sequences, GNNs can capture complex relationships between nodes (entities) and edges (connections) in a graph. They work by iteratively aggregating and transforming node features based on the graph's structure, allowing them to capture both local and global patterns in the network. GNNs are particularly powerful for tasks like node classification, link prediction, and graph-level classification, and they're widely used in fields like social networks, recommendation systems, biology, and physics.

3.3.1 Node Embeddings

Before the evolution of Graph Respresentation Learning and Graph Neural Networks, in order to apply machine learning with graph data, the researchers had to go over the time consuming process of feature engineering. This process usually requires expertise in the task of interest and specific manipulation depending on the downstream task. So every time we had to deal with a problem with graphs, we had to go through this process and extract useful features for this specific task manually.

This manual extraction of useful features was dropped and replaced with the evolution of **Graph Representation Learning**. Graph Representation Learning enables the automatic task-independent feature learning. One of the most famous methods that has been used is *node2vec* [19]. Simply put, *node2vec* assigns one vector, which we call embedding, to each node. In other words, it embeds each node into a d-dimensional space. The goal is to encode nodes in a way such that similar nodes are close in the embedding space. In *node2vec* similarity is defined through random walks. They first ran a lot of random walks starting from each node. If node v visits node u frequently it means that these two nodes are somehow closely related either through immediate links or through higher order neighborhood. In any case, the dot product of the embeddings $e_u^T e_v$ should be proportional to the probability of visiting node v in a random walk that starts from node u. For example if u and v are neighbors then the probability of visiting u starting

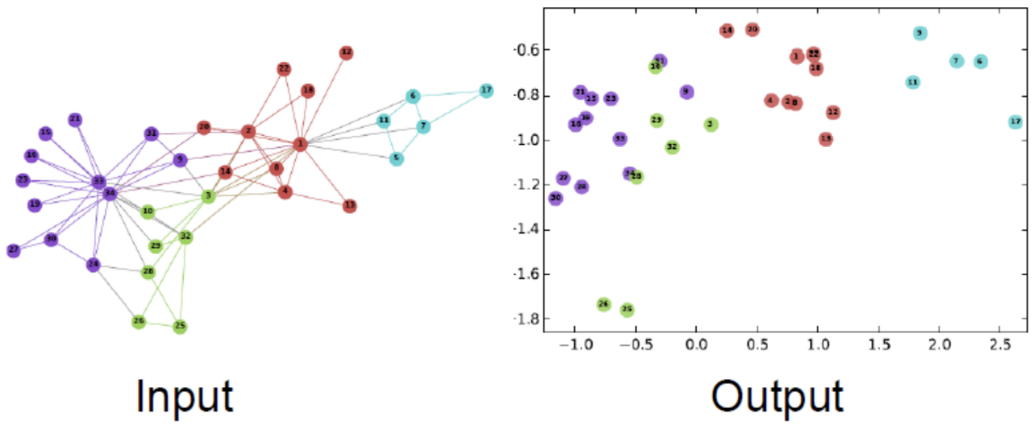


Figure 3.2: Node Embeddings

from v is high. So, the two nodes should have similar embeddings, which means their dot product value should be high which happens when their embeddings are close in the embedding space. That's the core idea. Notice that the learning process is task independent. We can use those embeddings for downstream tasks. For example, we could train a classifier for node label prediction. Closely related nodes should have the same label, and this is an easy task for the classifier, if we provide adequate node embeddings.

Although methods like node2vec and DeepWalk are successful in many applications, they come with limitations. Some of them are:

1. We need one embedding for each node which brings $O(|V|)$ complexity.
2. We can only work with nodes that we have seen during training and thus we have learned an embedding. This way we cannot generalise into new unseen graphs without creating the embeddings from scratch.
3. They are solely based on random walks and don't incorporate features attached to the nodes, which in many cases carries important information for the task. For example in the process of solving the *ModerateExpressed* problem we would like the embeddings to capture information about the opinions of the nodes.

Thankfully Graph Neural Networks are capable of constructing embeddings that capture structural information and leverage node features at the same time.

3.3.2 Graph Convolutional Networks

The first and most basic architecture of Graph Neural Networks is the Graph Convolutional Network [18]. This architecture is based on an efficient variant of convolutional neural networks which operates directly on graphs. One natural question is why do we even need special architectures to work with graphs? Why don't we simply use a dense neural network and feed it with the adjacency matrix of a graph, augmented with the node features? This approach initially seems fine but it has a lot of issues. The most important are:

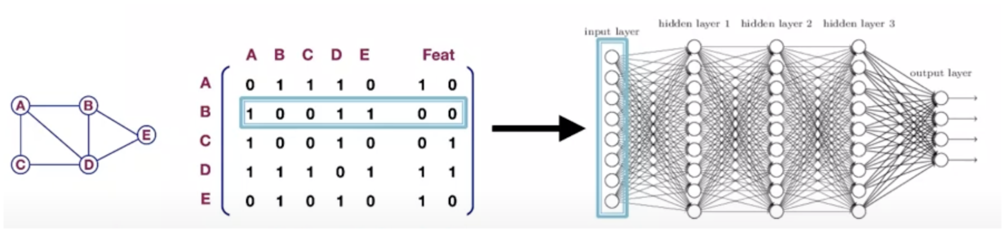


Figure 3.3: Feed Adjacency Matrix to Fully Connected NN

1. The input of the neural net is $O(|V|)$ because each row of the adjacency matrix has $|V|$ entries plus the node features. The number of training examples that we have is $|V|$ (in a node task like node classification), so the number of the trainable parameters is multiple times the number of training examples, which leads to unstable training and overfitting. To provide an example, suppose we have a graph with 10 nodes and for simplicity nodes have no features. In this case we need a neural net with 5 neurons at the input layer. This brings us to a total number of parameters X. Now, if we decide to add another 5 nodes to the graph, then we will need 10 input neurons, which automatically increases the number of parameters. So, the minimum complexity of our model has a lower bound that depends on the size of the input graph. For a graph with $|V|$ nodes, we can't have a model with less than a minimum number of parameters. We begin with a minimum capacity that depends on the number of the nodes i.e. the number of the training examples i.e. the size of the dataset. Usually in machine learning we adjust the complexity of the model, based on the size of our dataset. For example with small datasets we avoid very complex models due to their high capacity that leads to overfitting.

2. Such approach is not applicable to graphs of different sizes because the dimensionality of the input depends on the size of the graph. If we change the architecture of the network we have to train it from skratch.

3. It is sensitive to node ordering. The same graph can have multiple adjacency matrices if we simply reorder the nodes. But the information is always the same.

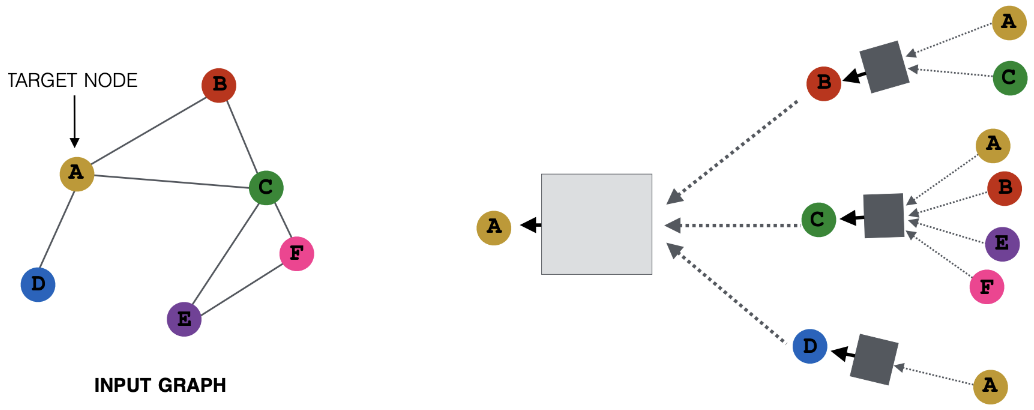


Figure 3.4: Computation Graph for Node A

> Naturally graphs don't have ordering, which is one challenge that GNNs deal with. So if we train a neural net on a graph and then simply permute the adjacency matrix, the network will yield different output regardless the fact that the input is actually the same. To further explain this point, let's take a look at Figure 3.3 and suppose we give as input to the graph node E and we get an output. Now imagine if we swap the names of nodes A and B (only the names). We have the exact same graph, the exact same information but the adjacency matrix is different. Focusing on node E, the first two columns would be swapped. Now, if we feed the new node E to the neural net we will get a different output because we provided a different input, but the information is actually the same. This behavior is not desirable.

Briefly, GCNs are able to collect information from the neighborhood of a node. Each node has a feature vector. In our task the feature vector of each node i contains the opinions of the node (s_i, z_i) so we have 2-dimensional feature vectors for each node. When constructing the embedding for one node, his neighbors send a message with their features. The messages are aggregated and transformed and the information is kept into a new vector. An embedding can encode information that comes from 1-hop neighbors or more, but usually we avoid collecting information from many hops away because we run the danger of collecting information pretty much from every node in the graph. In this case all the nodes would collect the same information and end up having similar embeddings, and thus be indistinguishable. The following picture summarizes the functionality of such networks:

The depicted architecture is a 2 layer GNN that carries information from 2-hop neighbors. In this image we can see how the information that constructs the embedding for node A is collected. Inside the boxes a 2 step process takes place. First, the incoming vectors are aggregated with an order invariant aggregation function such as averaging. Then a non-linear transformation is applied and the resulting vector is passed to the next layer. The following formula shows the math behind the embedding for node v for the layer $k + 1$:

$$h_v^0 = x_v \text{ this is simply the initial features of node v}$$
$$h_v^{k+1} = \sigma(W_k \sum_{u \in N(v)} \frac{h_u^k}{|N(v)|} + B_k h_v^k)$$

Where σ is any non-linear function such as ReLU. So basically at each layer we take an average of the embeddings of the neighbors and apply a transformation to construct the embedding of the next layer. This way the embedding summarizes the neighborhood of each node. Then we can use the final embeddings for downstream tasks such as classification and regression. All we have to do is attach a head at the end and feed it with the embedding as input. What's interesting about GNNs is that we can train them in an end to end fashion. The embeddings are trained for our specific task. Once we learn the parameters W_k and B_k for each layer k, then we can use the network with any graph at test time. Moreover the network can work with any graph size which makes it a great tool to work with graph data.

Chapter 4

Problem Statement and Proposed Solution

Here we give the formal definition of the problem *ModerateExpressed*. We provide the formal definition for the algorithm *GreedyExt* [5] and finally we present the algorithm that we propose, the GNN Algorithm.

4.1 ModerateExpressed

As we have already mentioned this problem was introduced in [5]. The formal definition is:

ModerateExpressed

Given a graph $G = (V, E)$, a vector of internal opinions $\mathbf{s}$, the resulting vector of expressed opinions $\mathbf{z}$, and an integer K, identify a set T of K nodes such that fixing the expressed opinions of the nodes in T to 0, minimizes the polarization index π.

To explain a bit further, essentially what we have to do is:

1. Based on the initial Graph G and opinions $\mathbf{s}$ and $\mathbf{z}$ we pick K nodes and add them to the solution set T
2. Then we fix the external opinions $z_i = 0, \forall i \in T$.
3. Next, we run the opinion evolution process and get the final opinion vector $\mathbf{z}$ at the new steady state. Note that we actually don't have to run the simulation but we can rather compute the new steady state based on the equations that we provided in Section 3.1.

4. Finally we calculate the polarisation index with the new opinions.

The goal, is to construct a solution set T, where $|T| = K$, such that the polarisation in the end is minimised. This is a discrete optimization problem and it belongs to the class of NP-hard problems. An equivalent problem was also proposed in [5], where instead of setting the external opinions $z = 0$, they set the internal opinions $s = 0$ in order to minimise polarisation. Their results showed that moderating the external opinions was more effective, and for this reason we focus only on this variation of the problem.

4.2 GreedyExt

This algorithm was introduced in [5] and it is a greedy algorithm named *GreedyExt*.

GreedyExt

GreedyExt is an iterative algorithm which starts with an empty set T^0. At each step t the algorithm adds to the existing solution T^{t-1} the node v which when setting $z_v = 0$ causes the largest decrease $\pi^{t-1} - \pi^t$ in the objective function. Again, we fix the external opinion of the selected node to zero and calculate the new steady state vector **z**, like we described in Section 4.1. We have to do this at each timestep for every node and eventually select the one that minimises polarisation. For cleaner explanation we provide the pseudocode for *GreedyExt* in Algorithm 1.

Step 19 of the pseudocode of Algorithm 1 might seem confusing at first glance, for this reason we explain it in more detail:

Step 19: At each timestep, a node i is selected and added to the solution set, as we have already mentioned. The opinion of this node will be set and fixed to zero. This means that we shouldn't account for this node in future opinion evolution processes, because in such case it would deviate from zero and converge to some new opinion. In simple words, we want to set this node to zero, run the opinion evolution process to affect the opinions of the rest of the nodes once, and then we remove all edges connecting to node i. This way the node causes an effect in the opinions of the network and this effect "persists" throughout the rest of the timesteps. If we let the node participate in the evolution process in the following iterations, it will deviate from zero and it will reverse its effect to some extent. In such implementation, at the end of the algorithm only the node of the last iteration will have its external opinion equal to zero, which is not what we aim for.

A naive implementation of *GreedyExt* is computationally expensive. Specifically its

Algorithm 1 GreedyExt Algorithm

```
1:  Input: Graph $G = (V, E)$, Maximum timesteps $K$
2:  Initialize solution set $T \leftarrow \emptyset$
3:  $\pi_{min} \leftarrow +\infty$
4:  for $t = 1$ to $K$ do
5:      for $node$ in $V - T$ do
6:          $z_{node} \leftarrow 0$
7:          Compute the new steady-state vector $\mathbf{z_{new}}$
8:          $\pi \leftarrow \frac{||\mathbf{z_{new}}||_2^2}{|V|}$
9:          if $\pi < \pi_{min}$ then
10:             $\pi_{min} \leftarrow \pi$
11:             $v^* \leftarrow node$
12:         end if
13:         Reset the opinions to the original state (before the inner loop started executing)
14:     end for
15:     $z_{v^*} \leftarrow 0$
16:     Compute the new steady-state vector $\mathbf{z_{new}}$
17:     Update the graph with the new opinions $\mathbf{z_{new}}$
18:     Isolate $v^*$ by removing all links between $v^*$ and other nodes
19:     Add $v^*$ to the solution set $T$
20: end for
21: Output: Solution set $T$
```

time complexity is in $O(Kn^3)$, where K is the number of nodes whose opinion is set to zero and $n = |V|$. The complexity analysis is simple. For one node we have to check the polarisation at the new steady state. In order to compute the new steady state we need to multiply and $n \times n$ matrix with an $n \times 1$ based on formula 3.8. The complexity of its process is $O(n^2)$. But we have to repeat this process for every node, and we do it K times because of the K timesteps. So the overall complexity is $O(Kn^3)$

The authors of [1] leveraged some linear algebra properties and managed to bring the complexity down to $O(Kn^2)$. Nevertheless, in their paper they mention that *GreedyExt* works fine for small graphs but doesn't scale to larger graphs. Our method aims to overcome this scalability limitation, offering a good solution even for larger graphs.

We implemented both the naive and the optimized version of the algorithm *GreedyExt* in order to compare our results. The implementation details for the optimizes version are provided in the original paper [5].

4.3 The GNN Algorithm

The algorithm that we propose consists of two parts: (a) the "offline" and (b) the "online" parts. The offline part is concerned with the training of the GNN, while the online part deals with the problem of selecting the K nodes to be set to zero. We discuss the "offline" part in detail in Chapter 6.

In this section, we delve into the specifics of the online algorithm that we propose, which builds upon and aims to improve, in terms of time, the previously discussed *GreedyExt* algorithm. The core principle behind both algorithms is iterative: at each timestep, we aim to select the best node to add to the solution set based on its contribution to the reduction of polarization in the network. However, our approach introduces a key innovation that dramatically enhances efficiency and scalability.

The fundamental difference lies in how the "best" node is identified. In the original *GreedyExt* algorithm, this process involved a brute-force search where, at each step, the polarization of the network was computed after setting the opinion of each node to zero individually. This required recalculating polarization multiple times, leading to substantial computational overhead, especially for large graphs. In contrast, our algorithm replaces this brute-force search with a trained neural network model, which predicts the polarization reduction directly for each node. Figure 4.1 summarizes the pipeline for finding the best node:

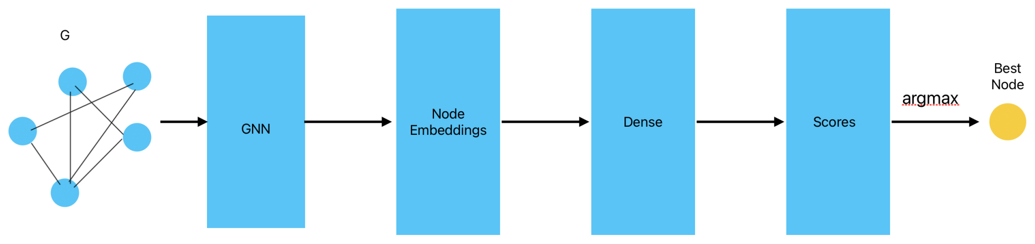


Figure 4.1: Selection Pipeline

In summary, by integrating machine learning into the node selection process, our algorithm aims to significantly improve upon the GreedyExt method, in terms of time required to construct a solution set, that is almost as good as the one that *GreedyExt* finds. It maintains the iterative structure, but replaces brute-force computation with a more scalable approach, reducing complexity, while preserving accuracy in the identification of key nodes. The proposed system is a Graph Neural Network followed by a regression head (what we denote as "Dense" in Figure 4.1). Usually a simple linear layer is enough to replace the Dense, meaning we don't actually need a Fully Connected

Neural Network architecture there, although we can have one, if we want. The Graph Neural Network already introduces non-linearities in the pipeline, and that's why a simple linear transformation in the end is usually enough. We discuss the GNN component of the pipeline in detail in Chapter 6. The pseudocode for the "online" part of the algorithm is provided in Algorithm 1.

Algorithm 2 GNN Algorithm

```
1: **Input:** Graph $G = (V, E)$, Maximum timesteps $K$, Pre-trained GNN model
2: Initialize solution set $T \leftarrow \emptyset$
3: **for** $t = 1$ to $K$ **do**
4:     $scores \leftarrow$ GNN.predict$(G)$
5:     Select node $v^* \leftarrow \arg\max_v scores(v)$
6:     Add $v^*$ to solution set $T$
7:     Update $G \leftarrow$ set opinion of $v^*$ to $0$ in $G$
8:     Compute the new equilibrium and isolate node $v^*$
9: **end for**
10: **Output:** Solution set $T$
```

Chapter 5

Datasets

In terms of data, our results are based both on synthetic and real networks. First, we present how we generated the synthetic graphs.

5.1 Synthetic Data

The main focus of this thesis is on the dual echo chamber, so we had to manually create graph structures that capture the properties of a dual echo chamber polarised network. We used different approaches.

5.1.1 Approach 1 for Synthetic Graph Generation

One approach is to create a set of n nodes and randomly assign opinions s to them in the range [-1, 1]. Next, we add the edges between the nodes. The probability of adding an edge is proportional to the distance between the opinions of the two nodes. Specifically the connection probability of node u and v is:

$$P(u, v) = b \cdot (1 - |s_u - s_v|), \text{ where } b \text{ is a base probability}$$

Notice that if the distance of the two opinions is greater than 1 then the probability becomes negative which programmatically means that the two nodes don't connect.

The reason why we choose a linear formula for the calculation of $P(u, v)$ instead of a higher order like quadratic is the following: If we used a quadratic formula then the probability of two nodes connecting goes to zero very quickly even for small distances between the opinions. As a consequence, we would end up getting a network where users with **very** similar opinions connect with each other, simply put. In other words, a higher order formula increases the homophily (the tendency of people interacting with similar minded individuals) in the network too much. In contrast, the linear formula introduces a moderate degree of homophily and avoids potential fragmentation. Moreover, we can always control the dergee of homophily through the base probability parameter b. To be more precise assume the following alternative quadratic formula and an example:

$$P(u, v) = b \cdot (1 - |s_u - s_v|)^2, \text{ where } b \text{ is a base probability}$$

Now assume $|s_u - s_v| = 0.2$. Then :

$$\textit{Linear: } (1 - |s_u - s_v|) = 0.8 \qquad \textit{Quadratic: } (1 - |s_u - s_v|)^2 = 0.64$$

According to this simple example we see that with the quadratic formula the probability of forming a connection decrease faster with respect to the distance of the opinions of two nodes.

With this method we get naturally two communities with opposing opinions. To make the data more realistic and slightly increase the polarisation in the network we extended the graph generation with the following strategies:

1. We decided to make sure that we have some extreme users in each community, in order to simulate a realistic network where extreme users are often part of. Moreover, introducing some extreme users increases the overall polarisation. For this reason, during the opinion assignment we assign opinion 1 or -1 with probability $P_{extreme}$, otherwise we sample uniformly an opinion in the range [-1, 1].
2. We introduced variance in the degree of nodes. Specifically, the base probability b of each node takes randomly one value from a pre-defined list of possible base probabilities. For small graphs up to 200 nodes the list of possible base probabilities is [0.1, 0.05]. We intentionally set the base probabilities to small values, because the number of possible edges for a node in a graph with n nodes is $(n-1)$. In a graph with 200 nodes and base probability $b = 0.1$ a node's degree will be relatively high. In any case, we can tune those parameters to fit our needs. In practice, we this methodology we get some nodes with higher and some with lower degree.

With the method that we have described, we get a network with two echo chambers where each chamber has high and low degree nodes and a fraction of the nodes has extreme opinions. We believe that this structure successfully simulates a realistic polarised network. The following figure shows an example graph that is created following this methodology:

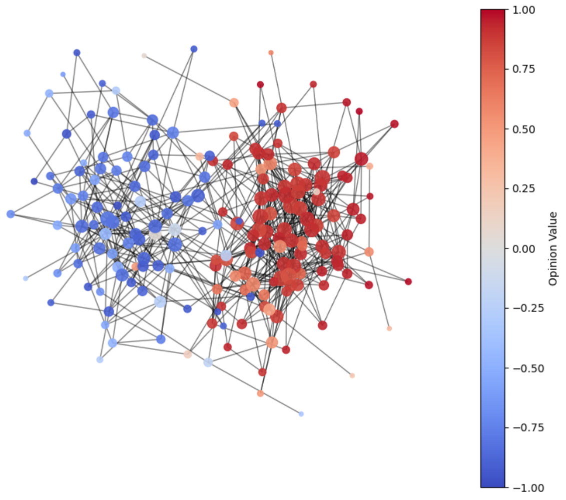


Figure 5.1: Manually generated dual echo chamber graph with Approach 1. The opinions refer to the steady state vector $\mathbf{z}$

This way we get two communities with opposing opinions and each community is sure to contain some extreme users. The polarisation of the graph in Figure 5.1 is $\pi = 0.128$

We now perform some further analysis on the graph that we generate. Our goal is to have a network where the steady-state opinions are not gathered around zero, to introduce some polarisation. We'd rather have more opinions closer to the edges of the opinion spectrum, in order to simulate a polarised network with two communities.

We plotted two histograms, one for the distribution of the degrees in the graph and one for the distribution of the opinions. The diagram in Figure 5.2 shows clearly that the network is polarised as most of the opinion density is gathered around the two extremes. Finally we plot the distribution of degrees to demonstrate that there is indeed a variety in the degrees of the nodes. As we can see in Figure 5.3 the network consists of mostly medium degree nodes but there are also some high and some low degree nodes. We believe this is also a characteristic of real networks.

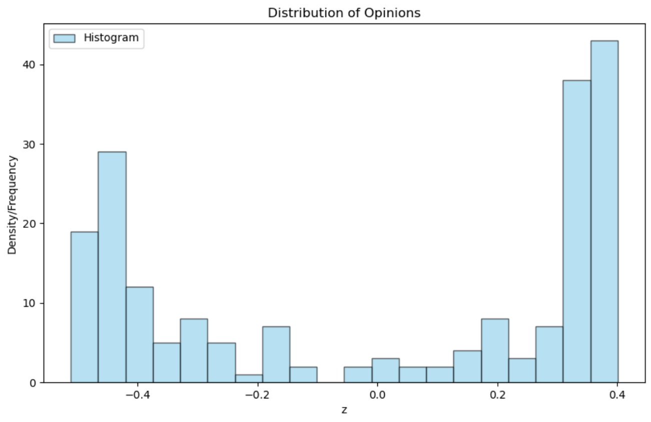


Figure 5.2: The distribution of opinions **z**

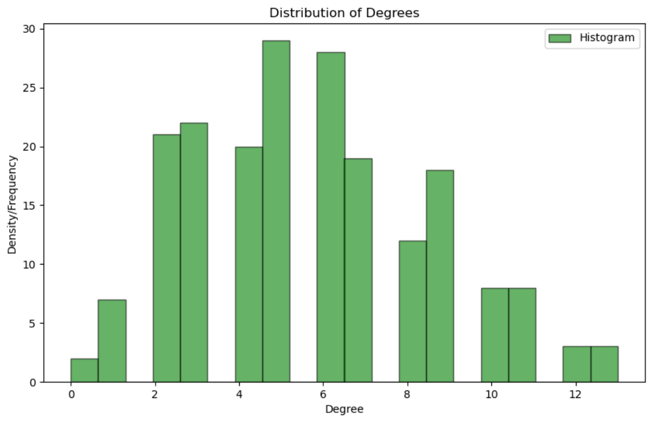


Figure 5.3: The distribution of degrees

5.1.2 Approach 2 for Synthetic Graph Generation

The second approach follows a completely different methodology. We focus individually on each community and after we create both communities, we add few edges between them. Moreover, the structure of each community is not random. We don't just use a simple model like Erdős–Rényi because in such models all nodes are similar. This is because in models like Erdős–Rényi the probability of each edge is the same and thus the nodes are differentiated only through their opinions. In a real scenario, even inside a community nodes are different. Some nodes are more famous and influential while other nodes are less famous. In fact the degrees in real social networks follows a power law distribution based on Mark Newman [20]. In our previous approach the extreme nodes had a high chance of being connected with each other (because their opinion distance is small since they are all extreme). In this approach we want to have extreme nodes again, but not necessarily connected with each other. So we decided to create each community by interconnecting smaller, sub-communities. Each sub-community of course has some extreme and famous nodes. Then the sub-communities are connected with few edges. At the end we connect the two chambers - the two large communities with opposing oppinions. The resulting graph is shown in the following figure:

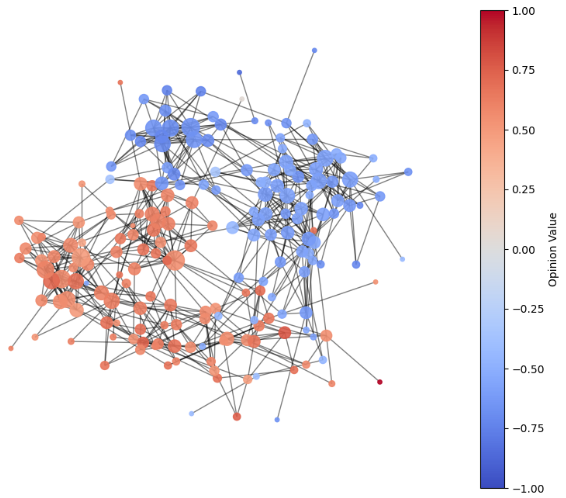
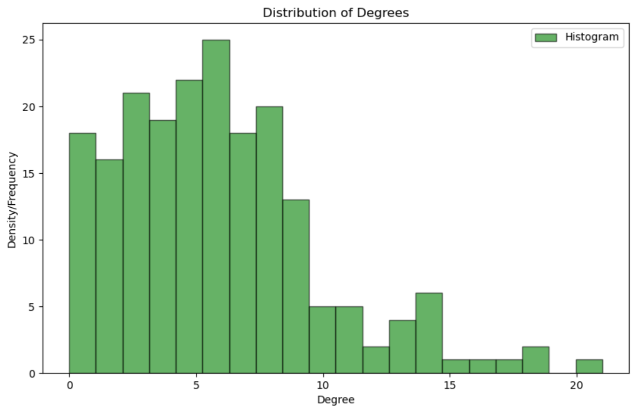


Figure 5.4: Graph generated with Approach 2

The polarization index for this graph is $\pi = 0.093$. This network has two communities but they are not separated as much as in the first case. Moreover we can see the subcommunities within each chamber. The size of the nodes is proportional to their degree.

The following diagrams in Figure 5.5 show the degree and the opinion distribution of this graph. Again we can see that the opinions are polarized and we get variance in the degrees of the nodes. However, de-polarisation is harder in this case because intuitively we have to depolarize each sub-community separately. For example if we target an influential extreme node that is part of a subcommunity, his neutralisation won't have significant effect into other sub-communities due to the weak connectivity.

The two approaches that we have presented so far, for generating synthetic graphs, are two simple approaches. There are many alternatives that one can come up and experiment with. Nevertheless, in practice we got satisfying results from those graphs. Moreover, the algorithms that we present are also tested on real datasets, which we present next.



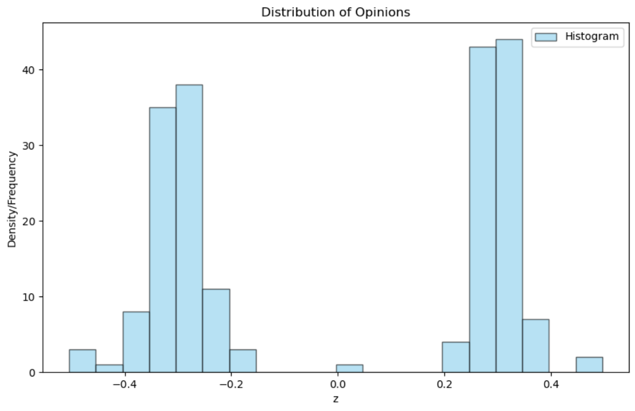


Figure 5.5: Distributions of Degree and Opinions of Graph generated with Approach 2

5.2 Real Data

Political Books

This dataset consists of political books that were sold from Amazon. The edges between political books represent frequent co-purchasing. The books are labeled as Conservative, Liberal and Neutral. The opinion mapping that we use is the following: Conservative: 1, Neutral: 0, Liberal: -1. The reason why we choose these 3 values is because we have no means to quantify numerically for each node what their exact opinion is. One alternative is to randomly sample opinions in the negative side of the spectrum of Liberal nodes, and in the positive side of the spectrum for the Conservative nodes. Finally for the neutral nodes we could sample opinions around zero. There is really no big difference. Of course, the resulting polarisation of the network and the final selected nodes by the algorithms would be different as the whether they are selected or not depends both on their opinion and graph structure. But we are examining whether the algorithm that we propose makes equivalent choices to the *GreedyExt* algorithm, so the starting point of the network doesn't really make any difference. In both cases the *GreedyExt* algorithm will make its greedy choices, and our algorithm should present similar behavior. For our experiments we decided to stick with the +1,0, -1 mapping to align with the equivalent experiments in 7. The polarisation is $\pi = 0.107$. It consists of 105 nodes and 441 edges. The relevant graph is shown in Figures 5.6.

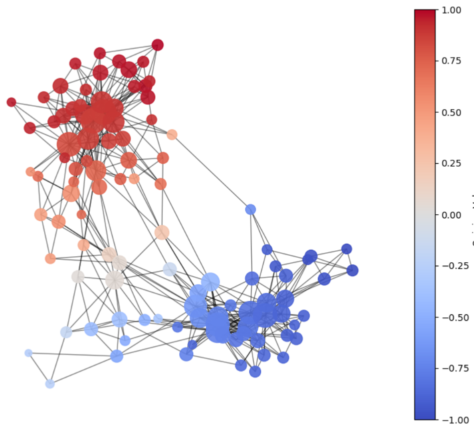


Figure 5.6: Political Books Network

Karate Club

One of them is the famous Karate Club with 34 members. It consists of two equal-sized communities of friends around two rival instructors. The polarisation index for this graph is $\pi = 0.89$. Figure 5.7 shows the corresponding graph. Every node has a label with one of two instructors. We assign $s = +1$ to nodes of one instructor and $s = -1$ to nodes of the other instructor, for the same reasons as we explained before.

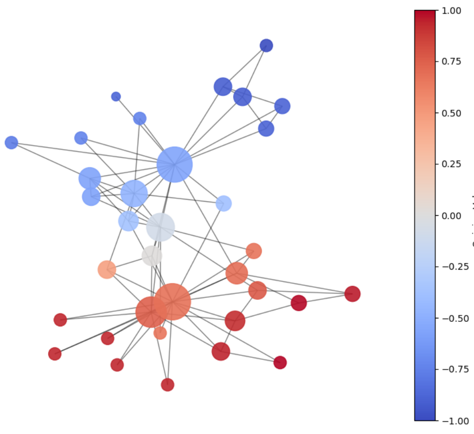


Figure 5.7: Karate Club Network

LiveJournal

LiveJournal is a free on-line blogging community where users declare friendship to each other. LiveJournal also allows users form a group which other members can then join. Such user-defined groups are considered as ground-truth communities. This dataset was introduced in [21] and is made public from the authors (SNAP Datasets). It consist of multiple communities. Because in this thesis we are interested in the dual echo chamber, we had to perform some data pre-processing. We took the 5000 best communities according to the original paper [21], where they define what makes a community "good". Then we sorted them based on their length and constructed the graph between the two largest communities. The number of nodes in each community is almost the same, and as we can see in Figure 5.8 this graph has a dual echo chamber structure.

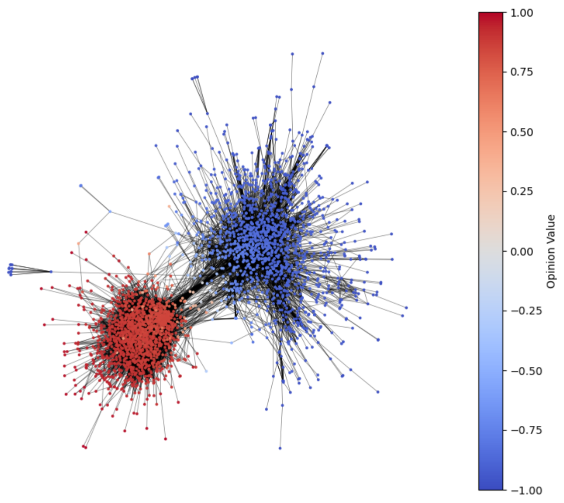


Figure 5.8: LiveJournal Network

Table 5.1 shows some statistics about the datasets that we use. The last column π shows the initial polarisation index of the network, before we start solving the problem.

Dataset	Nodes	Edges	π
Books	105	441	0.108
Karate	34	78	0.089
LiveJournal	2766	24138	0.179

Table 5.1: Statistics of the Real Networks

Chapter 6

Training the GNN

Earlier we mentioned, the algorithm that we propose can be broken into two main parts: the “offline” part, where we train the GNN, and the “online” part where we use the trained GNN to solve the *ModerateExpressed* problem. We presented the “online” part in Algorithm 1, in Chapter 4. We shift our focus now to the ”offline“ part of the algorithm.

Essentially what we aim to do is to train a GNN to predict the gain (the decrease in the polarisation) after setting a node to zero. We refer to this as the **“gain regression task”**

6.1 Create the Training Dataset

In the previous section we discussed about how the graphs that we use are generated. However, having a graph attached with opinions is not enough to train a model. Our goal is to train a model that generates useful embeddings which we then use to determine the effectiveness of each node, in terms of de-polarisation. Thus, we have to train our model in a supervised setting, and for this reason we need to construct a dataset with ground truth values. The steps for creating the dataset are the following:

1. We first create multiple synthetic graphs with the methods that we introduced in the previous section. We calculate the steady state external opinion vector $\mathbf{z}$.
2. Then for every user i we set her opinion $z_i = 0$ and compute the new equilibrium. We define the quantity *gain* as the decrease of the polarisation between the two steady states:
$$gain = \pi_{old} - \pi_{new}$$
We repeat this process for every node in every graph and store the gain as the

target values of our dataset. This way we have a dataset that contains information about the gain that we receive, when we set the opinion of any node to zero. For example if we set $z_i = 0$ we might get a gain of 0.3. If instead we set $z_j = 0$ we might get a gain of 0.5. In this case node j setting $z_j = 0$ is more effective, in terms of de-polarisation. Note that the gain is not always a positive quantity.

Overall, we create 200 graphs for our training dataset with 200 nodes per graph on average.

6.2 Training and Architecture

In this section, we focus on the central component of our pipeline: the Graph Neural Network (GNN). The specific architecture we employed is the Graph Convolutional Network (GCN), which has proven to be a strong choice for tasks involving graph-structured data. There are multiple other architectural choices that one can use, such as the Graph Attention Network [22], but for this thesis we stick with the GCN architecture. The embeddings generated by the GCN have a dimensionality of 16, which we found to be sufficient for capturing the structural and relational information necessary for our task. This dimensionality strikes a balance between model complexity and the expressiveness of the embeddings, allowing us to effectively encode the local neighborhoods of each node without introducing unnecessary overhead.

One of the most challenging steps towards developing our solution was to figure out what network architecture to use. By network architecture we refer to the exact values of the hyper-parameters including the number of layers, the number of hidden units, the learning rate etc.

6.2.1 Number of Layers and Oversmoothing

It turns out that the correct **number of layers** to use depends on the structure of the graph. The number of layers determines “how far away the information for the construction of an embedding is coming from”. For example if we use a 2 layer architecture, the embedding of a node is constructed by aggregating information from 1-hop and 2-hop neighbors. To explain exactly what we mean when we say that the correct number of layers depends on the structure of the graph, first we need to introduce the concept of **over-smoothing**. Oversmoothing is a challenge faced by the GNN’s and it is directly connected with the number of layers. Unlinke normal Deep Neural Network, more layers doesn't mean better performance on average, in the context of GNNs. In general, the more layers we add, the more distant information an embedding can encode. But, by adding more layers we risk aggregating information from the entire graph, which could lead to a situation where the embeddings of different nodes become indistinguishable from one another, because their construction is based on common information. This phenomenon, often referred to as over-smoothing, can degrade the quality of the embeddings and hinder the GCN's ability to learn useful node representations. Figure 6.1 shows how quickly the number of shared nodes (nodes that are used into the computation of the embedding for all nodes of interest) increases as we increase the number of layers.

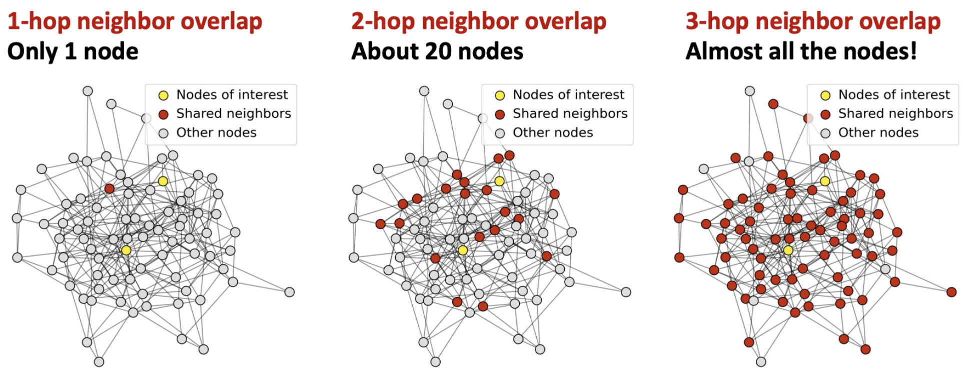


Figure 6.1: Neighborhood overlap

A shallow architecture is often recommended because, in many cases, gathering information from just a few hops away is sufficient to capture the relevant graph structure. Therefore, we must have in mind this tradeoff when choosing the number of layers: if we choose a shallow architecture we might fail to learn what we aim for, but if we go deeper, we might fail to learn at all. In order to make an informative choice, we performed some further analysis on the task that we are solving.

First of all, the proposed system is a Graph Neural Network followed by a regression head. The task that we are solving is a node regression task. Specifically for each node we try to predict what the gain is going to be if we set the external opinion of a node to zero (gain regression task). By gain, we always refer to the decrease of the polarisation after we set a node to zero. According to the Friedkin and Johnsen opinion model that we use, a change in the opinion of a node will affect the opinions of other nodes because of the averaging process. Perhaps a change in the opinion of a node could affect the opinions of every node in the network. Based on this, our model is expected to perform poorly in the node regression task. The reason is because our model only aggregates information from l-hops away, where l is the number of layers that we use. Of course we choose a value for l that doesn't introduce oversmoothing, meaning that by default the embeddings can't encode information from the whole network. So in a scenario where setting the external opinion of a node to zero, leads to changes in the opinions of every other node, theoretically a GNN shouldn't be able to predict the gain precisely, because it only observes a subset of nodes.

However, if we had a graph structure where the opinions of nodes further than l-hops didn't change significantly, then a GNN that only observes the nodes up to l-hops away should perform relatively well on the gain regression task. That's because based on the assumption that the opinions of nodes further than l-hops away don't change significantly, their impact on the gain is negligible. In such case, a reasonable number of layers for the GNN would be l. For example a sparse graph could be a good fit here. Because of its sparsity we expect mild changes in the opinions of distant nodes. This is why we support that the correct number of layers depends on the graph structure. Because we have to aggregate from the network as much information as possible, and avoid over-smoothing at the same time.

Now, we repeat here, based on this intuition, our model shouldn't be extremely good at the gain regression task because the actual gain depends on the opinions of the whole network, while the GNN only has access to information from l hops away. A natural question is, **"how well can the model perform?"** We investigate this question next.

6.2.2 How well can the model perform

In ordered to calculate an upper bound to the performance of our model we conducted some analysis on the gain. We performed some series of experiments to measure the fraction of the gain that comes from the change in the opinions of the l-hop neighbors, and how it compares to the actual total gain. In order to explain further we need to introduce some notation, which we then use in a simple example for better understanding.

Notation

Assume we have n nodes and we target node i. The following array represents the vector $\mathbf{z}$

$$\begin{array}{|c|c|c|c|c|c|}\hline z_1 & z_2 & ... & z_i & ... & z_n \\ \hline\end{array}$$

After we set node i to zero, the opinions in the new steady state vector will be different.

The following vector is the new steady state vector, denoted as $\mathbf{z}^{'}$:

$z_1^{'}$	$z_2^{'}$	...	$z_i^{'} = 0$	...	$z_n^{'}$

The total gain is computed as (for simplicity we don't divide the polarisation index by n here):

$$\begin{aligned} gain &= |\mathbf{z}|^2 - |\mathbf{z}^{'}|^2 && (6.1)\\ &= z_1^2 + z_2^2 + ... + z_i^2 + ... + z_n^2 - z_1^{'2} - z_2^{'2} - ... - z_i^{'2} - ... - z_n^{'2} && (6.2) \end{aligned}$$

This is the actual gain from setting $z_i = 0$. It takes into account the change in the opinion of all nodes.

Let $N_l(i)$ denote the set of l-hop neighbors of node i. Those are the nodes that can be reached from a path with $length = l$, starting from node i. We construct an artificial opinion vector which we call $\mathbf{z_{l-hop}}$. Let $\mathbf{z_{l-hop}}$ be the vector of opinions where:

$$\mathbf{z_{l-hop}} = \begin{cases} z_j^{'}, & \text{if } j \in \bigcup_{m=0}^{l} N_m(i) \\ z_j, & \text{otherwise} \end{cases}$$

In simple words, $\mathbf{z_{l-hop}}$ is the vector that consists of the new opinions (from $\mathbf{z}^{'}$) for nodes that are up to l-hops away from node i, and the old opinions (from $\mathbf{z}$) for nodes that are more than l-hops away from node i

Based on this, let $gain_{l-hop}$ be:

$$\begin{aligned} gain_{l-hop} &= |\mathbf{z}|^2 - |\mathbf{z_{l-hop}}|^2 \\ &= z_1^2 + z_2^2 + ... + z_n^2 - z_{l-hop_1}^2 - z_{l-hop_2}^2 - ... - z_{l-hop_n}^2 \end{aligned}$$

Finally let $GainError$ be:

$$GainError = |gain - gain_{l-hop}|$$

Our end goal is to examine how the $GainError$ behaves as we increase l. The following simple example demonstrates how we calculate the $GainError$ in a simple scenario.

Example

Assume we have n nodes and we target node i. Moreover assume that $N(i) = \{2, n\}$, i.e. node i's 1-hop neighbors are nodes 2 and n. We want to compute $GainError$ for $l = 1$.

The following vector is $\mathbf{z_{1-hop}}$:

z_1	$z_2^{'}$	...	$z_i^{'} = 0$	...	$z_n^{'}$

Notice that for non 1-hop neighbors of node i the opinion in $\mathbf{z_{l-hop}}$ is the same as in the original vector $\mathbf{z}$.

Then,

$$\begin{aligned} gain_{1-hop} &= |\mathbf{z}|^2 - |\mathbf{z_{l-hop}}|^2 \\ &= z_1^2 + z_2^2 + ... + z_i^2 + ... + z_n^2 - z_{l-hop_1}^2 - z_{l-hop_2}^2 - ... - z_{l-hop_i}^2 - ... - z_{l-hop_n}^2 \\ &= z_1^2 + z_2^2 + ... + z_i^2 + ... + z_n^2 - z_1^2 - z_2^{'2} - ... - z_i^{'2} - .... - z_n^{'2} \\ &= z_2^2 + z_i^2 + z_n^2 - z_2^{'2} - z_i^{'2} - z_n^{'2} \end{aligned}$$

Then we can easily calculate the $GainError$ according to its definition.

Now that we have presented the necessary notation we can explain the idea. Our initial intuition says that the opinions from nodes that are far away from the target node i, do not change significantly. If that's true, then we can get a good approximation of the total gain by considering the $gain_{l-hop}$ for some l. In such case, a reasonable choice for the number of layers for the GNN would be l. In simple words, suppose we only consider the changes in the opinions of neighbors up to $l - hops$ away. If the corresponding $gain_{l-hop}$ is close to the actual gain then we can choose the number of layers for the GNN to be l and theoretically the GNN should be able to perform well on the regression task. Essentially, this solves our previous concern regarding the GNNs limitation due to its inability to observe the whole network. If $gain_{l-hop}$ is close to the actual gain, then the GNN doesn't need to know necessarily what's happening further than $l - hops$ away, in order to make good predictions in terms of the total gain.

In essence, $gain_{l-hop}$ is the gain if we assume that nodes that are further than $l - hops$

are not affected significantly. That's why the corresponding entries for those nodes are the same in $\mathbf{z}$ and $\mathbf{z_{l-hop}}$. In order to see how $GainError$ behaves as we increase l we performed the following experiments:

Experiments

Assume we have a graph G and we want to see how the $GainError$ behaves for $l = 2$

1. We set one node to zero, compute the new steady state vector $\mathbf{z_{'}}$ and calculate the gain
2. Then we calculate the $gain_{2-hop}$
3. We store the $GainError$ for this node
4. We reset the opinions of the nodes to the original state
5. We repeat the process for all nodes in the graph and at the end we calculate the average $GainError$

We conducted this experiment $\forall l \in \{0, 1, 2, 3, 4, 5, 6\}$. When $l = 0$ we are simply considering only the change in the opinion of the target node itself. We refer to the average $GainError$ as the Mean Absolute Error.

Especially for the synthetic data, the average $GainError$ that we report is computed as an average over multiple graphs.

The scatter plot in Figure 6.2 shows the results over multiple graphs generated with Approach 1 and Approach 2 that we disccused in Chapter 5. In this plot we see the different values of l on the x-axis and the Mean Absolute Error on the y-axis. To provide an example, the dot for x=3 is the average error, when we calculate the $gain_{3hop}$ and compare it to the actual gain. So if we assume that we can calculate the gain by using only the change in the opinions of neighbors up to 3-hops away, and consider the change in the opinions of the others negligible, then we will on average have an error of approximately 0.007. This small value is misleading. In fact, $GainError = 0.007$ is pretty large in this context. The average real gain (which we try to approximate through the $gain_{l-hop}$), for the graphs generated with Approach 1 and Approach 2, when setting a node to zero is 0.014. This means that if we have an error of 0.007 we are $\frac{0.007}{0.014} = 0.5$ - approximately 50% off.

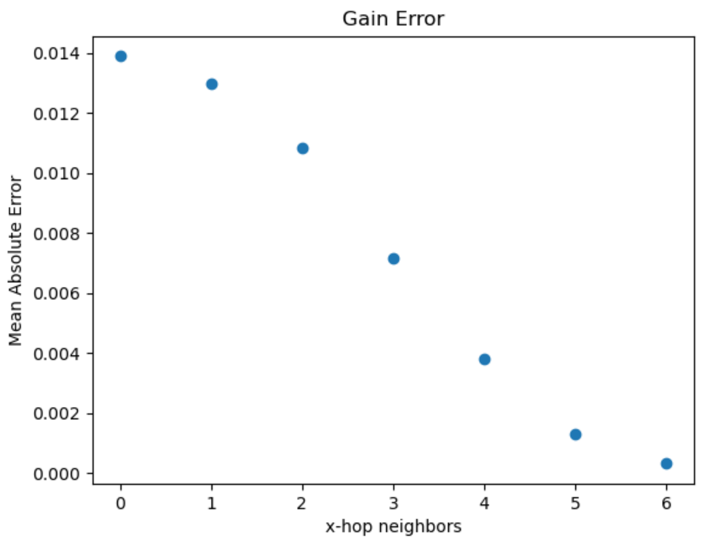


Figure 6.2: Mean Absolute Error on the gain when we consider only the change in the opinions of up to x-hop neighbors for 40 graphs generated with Approach 1 and Approach 2

We created such scatter plots for the Books and the Karate dataset as well. Figure 6.3 refers to the Books dataset. As we see in the figure, for this dataset if we consider up to the 4-hop neighbors, the $GainError$ goes to zero, which means that if we only stick with nodes up to 4-hops away we can get accurate prediction of the gain. Interestingly, if we limited ourselves up to 2-hop neighbors we could have an error close to 0. Based on this, theoretically a GNN with 2 layers should perform well on the regression task.

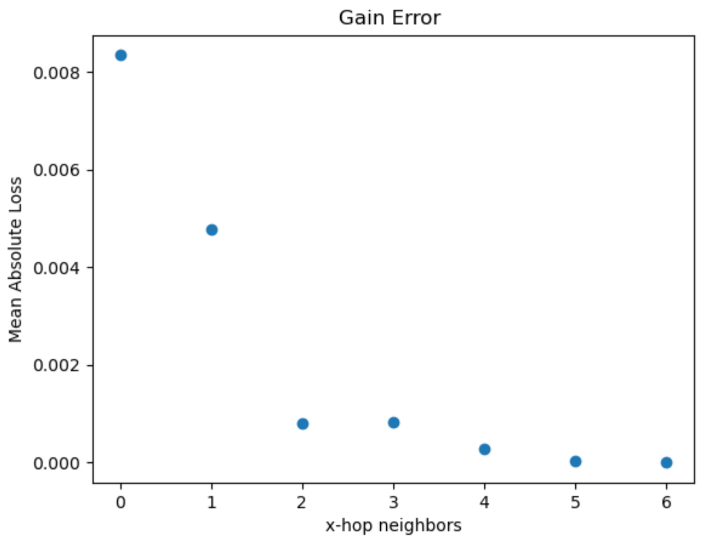


Figure 6.3: Mean Absolute Error on the gain for the books dataset

Finally, in Figure 6.4 we show the scatter plot for the Karate dataset.

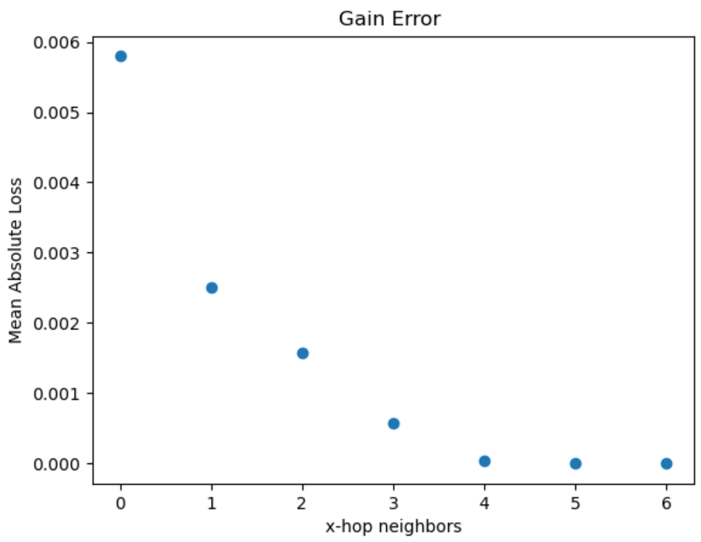


Figure 6.4: Mean Absolute Error on the gain for the karate dataset

Now we clarify what we should and shouldn't "take" from this discussion. Let's start with what we should not "take". The scatter plots that we provide don't necessarily mean that for example "if you use a GNN with $l = 2$ layers, you will end up getting an error x". GNN's are neural network architectures and we don't know exactly what each neural network eventually learns. Moreover they are function approximators, meaning that naturally they will have some loss. Finally, what the neural network will learn depends significantly on the data that we train it on. Poor data will lead to poor performance. Now, in a perfect world, where we would have the perfect data and where the neural network would learn exactly the underlying function, we could maybe say that if we use l layers we will not become better than what those scatter plots suggest. But again, we can't be 100% sure about that because what the neural network learns is not so clear.
The way that we like to view those scatter plots is as a rough estimate of how "blind" the neural net is for different values of l. We use the term "blind" because the neural network tries to estimate the gain, which as we saw in Equation 6.1, depends on z_i for every node i, but only has access to specific nodes and the structure of its local neighborhood. This of course is true for all machine learning tasks. For example in a classification task where we try to decide if a person is a man or a woman based on the height and weight. Sure, the height and weight are useful features but there are also other factors that determine the gender. But the height and weight turns out to be enough to create a good classifier. A great property of machine learning is that we don't need 100% of the information relative to a task, in order to train a model that performs well on this task. Of course the more useful features we have the better, but if we don't have some we can still make it work.

Back to our problem now, we didn't rely only on those scatterplots to decide our architecture. We experimented with different values and we created different models which we compare later. However, according to the scatter plots of the real datasets in the Figures 6.3, 6.4, a GCN with $l = 2$ could theoretically perform well. Interestingly as we will see later this intuition is confirmed. One of the models that we created has 2 layers and it works great with the real datasets.

One final thought. As we explained earlier, due to the limited information that we provide to the neural net, we don't expect to achieve the perfect regression. However, this doesn't mean that we should drop this approach and conclude that it can't be used for the underlying task of selecting K nodes to minimise polarisation. Sure, we might have significant loss on the regression task because for example for one node instead of $gain = 0.5$ we predict $gain = 0.3$ and for another node instead of $gain = 1.5$ we predict $gain = 1$. This is not great regression. However, based on those "bad" predictions we would still select the correct node, that is the one with true gain = 1.5 and predicted gain = 1. What we're trying to say here is that the fact that we most likely won't be perfect at regression, doesn't mean that we will be bad when selecting the K nodes to add to the solution set, for the same reason why a human doesn't need to know the exact height of two persons in order to tell who is taller.

6.2.3 Models

As we mentioned in previous Section, our models were trained on 200 graphs that were generated with Approach 1 and Approach 2. Each graph consists of 200 nodes. In Table 6.1 we show different settings of our models.

The 4th column "Development Loss" refers to the L1 Loss on a Development set that consists of 20 graphs. We use this set for hyper-parameter tunning and to capture overfitting. If the loss on the training set decreases and the loss on the development set starts to increase, then we can conclude that the model overfits the training set. In Table 6.1 we show the lowest development loss that each model achieved, meaning the loss of the best epoch throughout training. We also provide the loss per epoch for different settings in Figures 6.5, 6.6 and 6.7

This is the loss for the regression task and we clearly see that more complex models perform better. In fact, the best model in terms of gain regression on those graphs is GCN1.4 with 128 dimensional hidden layers and 4 layers depth. Moreover notice how

Model	Layers Num	Hidden Units	Development Loss
GCN1.1	1	128	0.40
GCN1.2	2	128	0.29
GCN1.3	3	128	0.24
GCN1.4	4	128	0.19
GCN2.1	1	16	0.45
GCN2.2	2	16	0.38
GCN2.3	3	16	0.35
GCN2.4	4	16	0.31
GCN3.1	2	256	0.26
GCN3.2	3	256	0.20

Table 6.1: Models

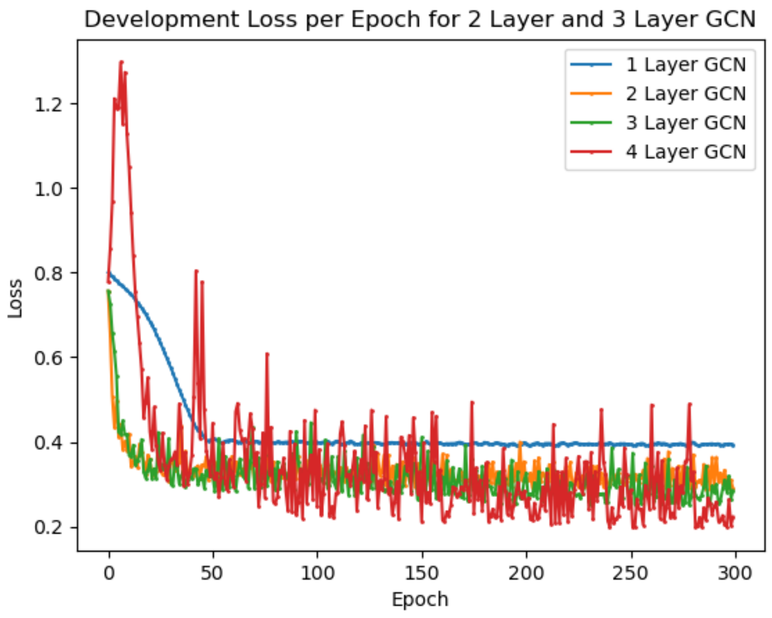


Figure 6.5: Development Loss per epoch for GCN1.1 - GCN1.2 - GCN1.3 - GCN1.4

the training process becomes less stable as the complexity of the models increases. The reason why we choose hidden layer dimensionalities to be 16, 128 and 256 is because we want to explore the scenarios with small dimensionality (16), medium (128) and a large (256).

In the gain regression task on the synthetic graphs, the best models are GCN1.4 and GCN3.2. But as we will see later the model GCN2.2, which is a far less complex model, performs great on the task of interest, which is selecting the K best nodes to minimise polarisation.

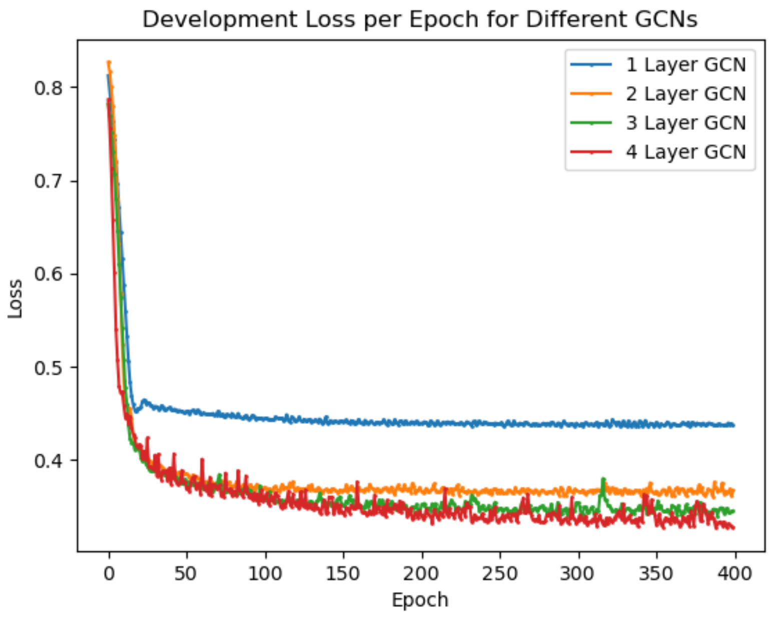


Figure 6.6: Development Loss per epoch for GCN2.1 - GCN2.2 - GCN2.3 - GCN2.4

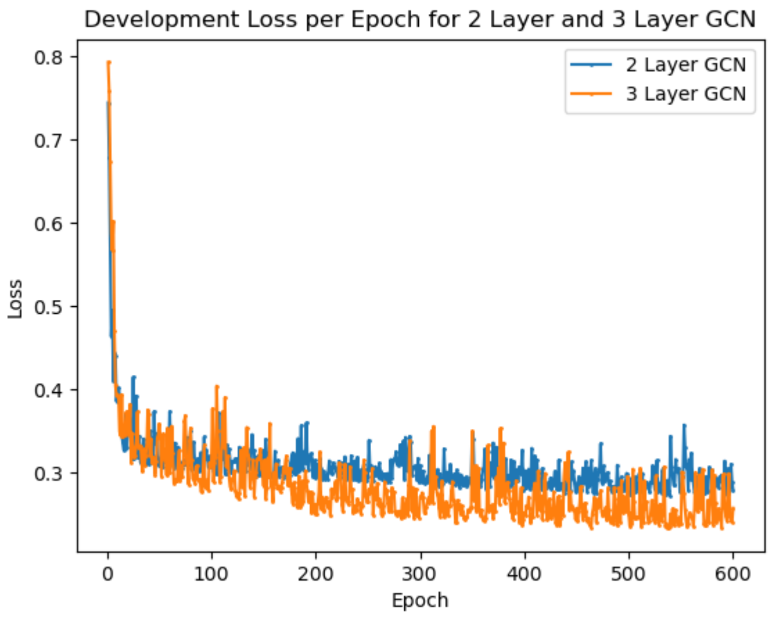


Figure 6.7: Development Loss per epoch for GCN3.1 - GCN3.2

Chapter 7

Experimental Results

In this section we present the results of our algorithm on a variety of graphs for different models. Furthermore we compare our algorithm with *GreedyExt* in terms of polarisation and scalability.

7.1 Compare to a Random Algorithm

The first step is to prove that our problem is not trivial and that our algorithm actually does something useful. For this purpose we compare our algorithm to a random algorithm.
The random algorithm is an iterative algorithm which at each timestep samples a node with equal probability and sets it to zero.
In order to compare with the random algorithm we test on 6 different graphs. Three of them are generated with Approach 1 and the remaining three are generated with Approach 2. The results are summarized in Table 7.1 and in Figure

Graph	GainPercentage Random %	GainPercentage GNN %
1.1	25	49
1.2	23	48
1.3	26	49
2.1	29	52
2.2	26	43
2.3	30	56
Average	26	51

Table 7.1: GainPercentage for Random Algorithm and GNN Algorithm on 6 Synthetic Graphs

In Table 7.1 see the gain percentage for each graph after setting $K = 5$ nodes to zero.

The gain percentage is simply the percentage of decrease in the polarisation after setting K nodes to zero. The graphs that we used consist of 200 nodes. We see that the GNN Algorithm is twice as good as the random algorithm. For this experiment we used the model GCN1.3 (Table 6.1). The percentage that we show on the table is the percentage of decrease in the polarisation and it is calculated as:

$$GainPercentage = \frac{InitialPolarisation - FinalPolarisation}{InitialPolarisation}$$

In Figure 7.1 we see the raw final polarisation for each graph for the two algorithms.

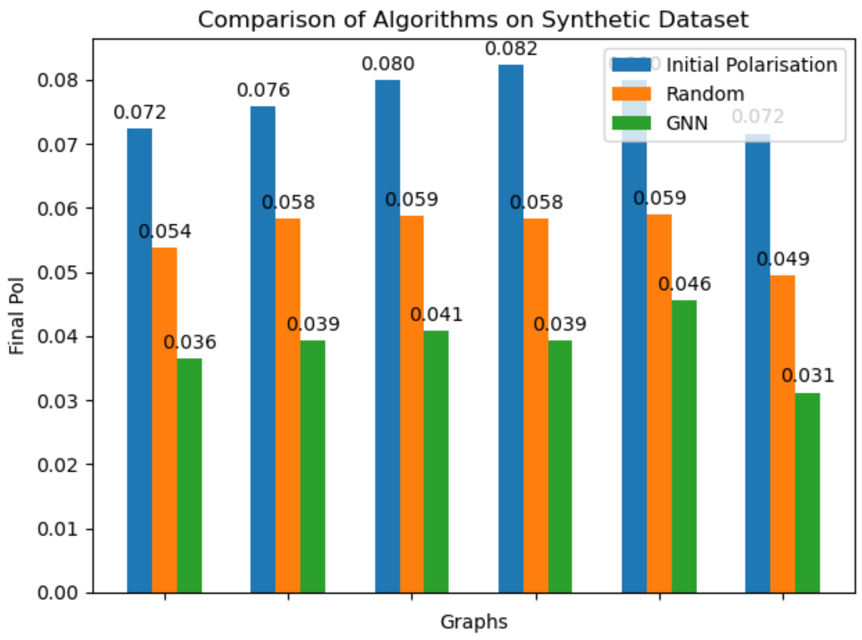


Figure 7.1: Raw polarisation after setting K nodes to zero for Random Algorithm and GNN Algorithm

From those results we conclude that our algorithm consistently outperforms the random algorithm. This result suggests that the problem is not trivial and that our neural network architecture actually learns something useful regarding the underlying task of de-polarisation. Now we introduce a new metric to compare the two algorithms. We call it $GainRatio$ and it is defined as follows:

$$GainRatio = \frac{AverageGainGNN}{AverageGainRandom} = \frac{51}{26} = 1.96 \approx 2$$

This metric simply means that the gain of the GNN algorithm is two times the gain of the random algorithm, on average. The $GainRatio$ depends on K. This is because if we choose very large K, we effectively set a large fraction of the network opinions to zero. In this case the set of nodes picked from the random algorithm will overlap significantly with the nodes picked from our algorithm and consequently, the $GainRatio$ will get closer to one. This observation is taken into account in the experiments that we conduct and that's why we bound K to be smaller than 10-20% of the total nodes. In order to prove this statement we perform an experiment that shows the polarisation on a graph as K increases.

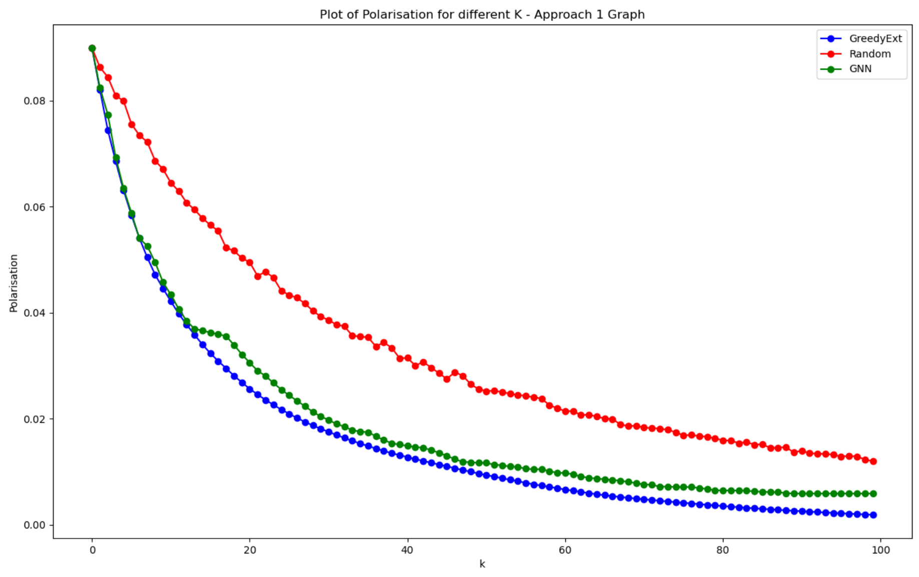
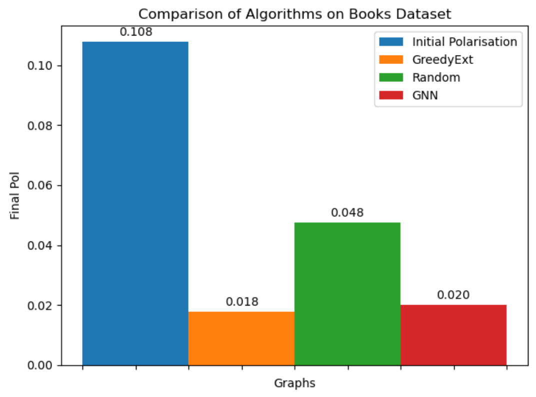
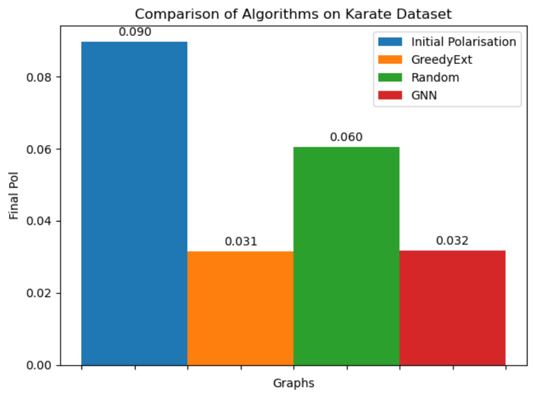


Figure 7.2: Polarisation vs K

Figure 7.2 shows what happens to the polarisation for a graph generated with Approach 1 as K increases. For clarity, we explain exactly what we see on the plot. Each bullet represents a fresh start of the algorithm where we set k nodes to zero, for the corresponding k on the x-axis. For example the blue bullet that corresponds to k=10 translates to: "Start the algorithm GreedyExt with the **original** graph, set k=10 nodes to zero and report the polarisation". Every bullet on the plots is a fresh start of the algorithm with the original graph. This is a very informative figure. First of all, we can see our GNN Algorithm following almost perfectly the *GreedyExt*. Moreover we can see that as K increases, the gap between the Random Algorithm and the other two algorithm is becoming smaller. This proves our previous statement that the $GainRatio$ depends on K, and for relatively large K it will approach 1. Smaller $GainRatio$ means that the advantage of our algorithm over the random algorithm is smaller and if it reaches 1, the random algorithm is exactly as good as the GNN Algorithm. This is because for large K, most of the nodes



(a) Books



(b) Karate

Figure 7.3: GCN1.4 vs Greedy on Books and Karate Dataset

are set to zero and the network is de-polarised no matter what. We can summarize this phenomenon in the following sentence:

If your budget is large enough (K), you don't need to make a sophisticated choice. You can simply buy everything.

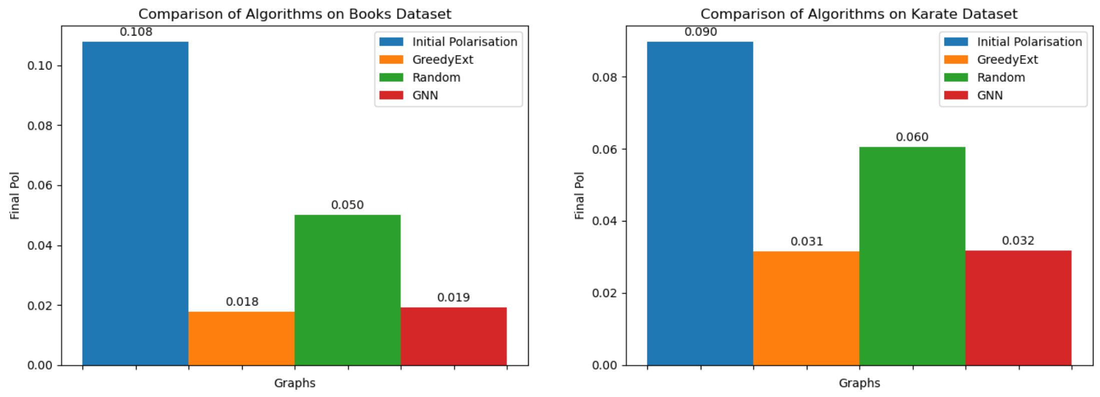
7.2 Model Selection and First Comparisons with the GreedyExt Algorithm

In this section we start comparing our algorithm with the *GreedyExt* algorithm. We will use some models from Table 6.1 and compare them to the *GreedyExt* algorithm on real networks. It is reasonable to choose models that performed relatively well on the regression task.

We begin with the GCN1.4 model which is a complex model with four 128-dimensional hidden layers and achieved the lowest development loss in Table 6.1. In Figure 7.3a we see that the final polarisation of the GNN Algorithm is very close to the one that *GreedyExt* achieves on the Books dataset. The same is true for the Karate Dataset in Figure 7.3b.

Next we investigate how a simple model performs. For this reason we perform the same experiments but we now use the model GCN2.2, which has two 16-dimensional layers and didn't perform very well on the gain regression task. The corresponding figures are Figure 7.4

Interestingly, the results are very similar. This agrees with what we briefly discussed



(a) Books (b) Karate

Figure 7.4: GCN2.2 vs Greedy on Books and Karate Dataset

earlier: The fact that a model is not good at regression, meaning that it has relatively higher loss compared to another, doesn't mean that it is going to be deterministically bad at solving the *GreedyExt* problem.

In summary, in this section we confirmed our previous claim that even though some models might not be perfect in terms of gain regression, they are still good enough to differentiate the "good" nodes. By good we mean nodes that minimise the polarisation. We also performed some first comparisons with the *GreedyExt* algorithm and saw that on two out of two real datasets our algorithm achieves similar polarisation to the *GreedyExt* algorithm. Finally, we developed a strong feeling that for this specific task we don't need perfect regression nor do we need very complex models, as the 16 dimensional two layer model is almost as good as the 128 dimensional four layer model. From now on we will continue with the simple model GCN2.2 to compare it with the *GreedyExt* algorithm and we will consider the more complex models again later.

7.3 GreedyExt vs. GNN Algorithm

7.3.1 Synthetic Data

The model that we use here is the GCN2.2, a relatively simple model with 2 layers and 16-dimensional hidden units. We use this model because as we saw earlier it performs almost as good as the more complex models. In fact its performs almost as good as the *GreedyExt* algorithm, and since we have a simple effective model we have no reason to employee more complex solutions. In Table 7.2 we show the performance of our model compared to the *GreedyExt* and Random algorithms for 6 synthetic graphs, generated with Approach1 and Approach 2. The graphs consist of 200 nodes and we set $K = 10$ which translates to 5% of the graph being nodes being set to zero.

Graph	GainPercentage GreedyExt %	GainPercentage GNN %
1.1	53	52
1.2	46	42
1.3	53	51
2.1	55	50
2.2	55	50
2.3	52	48
Average	52	49

Table 7.2: GainPercentage for GreedyExt Algorithm and GNN Algorithm on 6 Synthetic Graphs

The gain ratio of these algorithms is calcualted as:

$$GainRatio = \frac{AverageGainGreedyExt}{AverageGainGNN} = \frac{52}{49} = 1.06$$

As we can see from the $GainRatio$, the GNN algorithm is nearly as good as the *GreedyExt* on synthetic data as well.

For easier visualisation we provide a bar-chart with the final polarisation on each graph for the algorithms that we have discussed, in Figure 7.5.

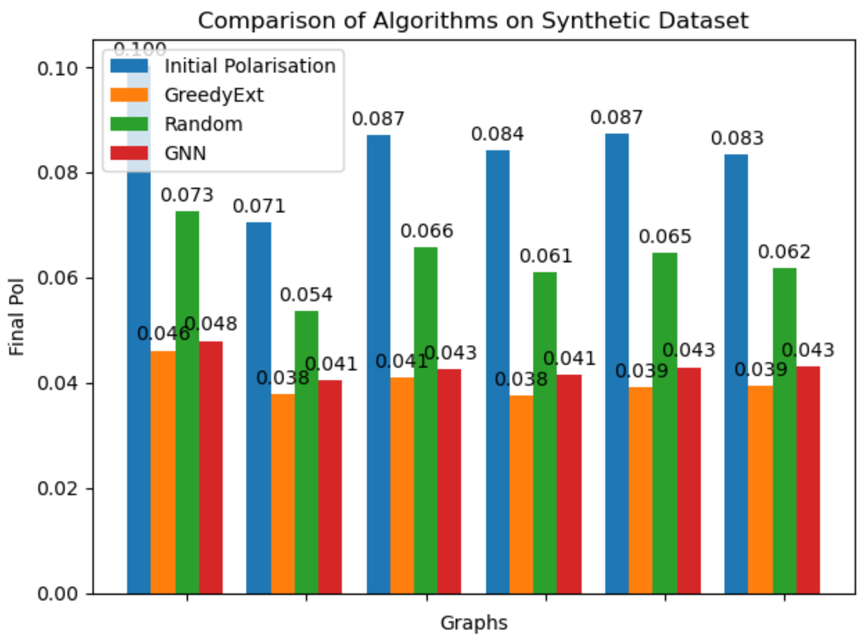


Figure 7.5: GNN vs GreedyExt on 6 synthetic graphs

The gain of *GreedyExt* is 1.06 times the gain of the GNN Algorithm. We expected our algorithm to be slightly worse because neural networks are trying to approximate the gain, but there is always some loss. Especially in some graph structures there might be multiple nodes that offer almost the same gain. In such scenarios the neural net, because of its approximation nature, could get confused and assign slightly higher score to a node that is not the best, but it is close to the best. Moreover, as we have already mentioned in previous sections our Graph Neural Networks are not the best gain regressors, but they are good enough for this task. *GreedyExt* on the other hand always makes the optimal choice for each timestep. This is the reasons why *GreedyExt* is slightly better than our GNN Algorithm, in terms of polarisation.

7.3.2 Real Data

We have already included a figure that shows the performance of our algorithm on the real datasets. The figure that we refer to is Figure 7.4. The only dataset that we have not presented yet is the LiveJournal which is the largest dataset that we have with ground truth communities. The corresponding bar-chart is shown in Figure 7.6. Here we set $K = 250$ which is approximately 10% of the dataset. Again, we observe that even for this large dataset, the simple model GCN2.2 performs almost as good as the *GreedyExt* Algorithm.

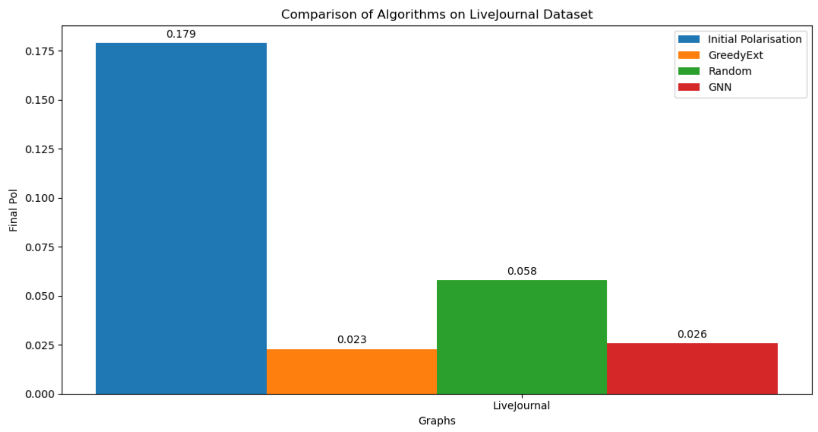


Figure 7.6: GCN2.2 vs Greedy on the LiveJournal dataset for $K = 250$

For completeness and easier comparison we provide a collective figure (Figure 7.7) about the 4 real datasets for $K = 10$. The sequence of the datasets is: Books (left), Karate (middle), LiveJournal (right).

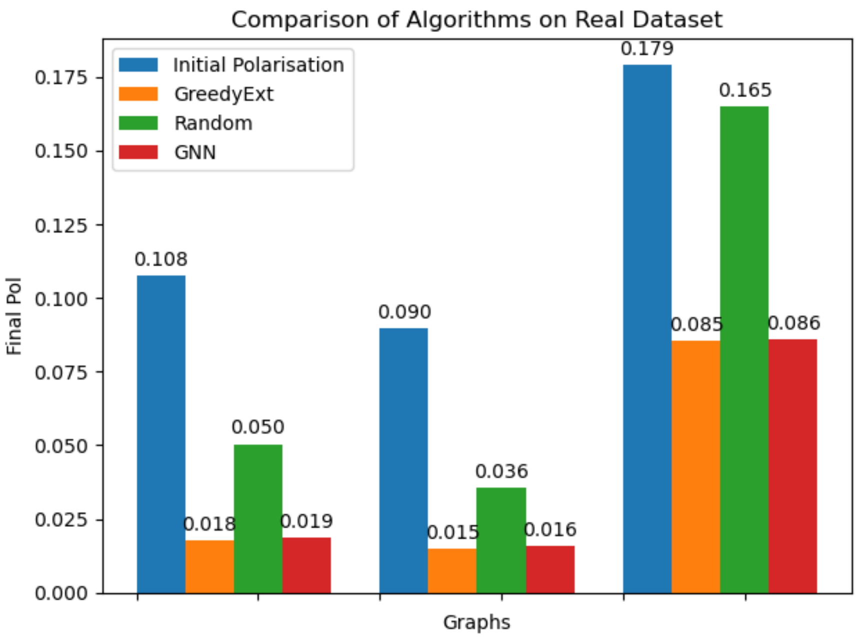


Figure 7.7: GCN2.2 vs Greedy vs Random on the real datasets for $K = 10$.

Finally we provide a table with the gain percentage (the percentage of decrease in the polarisation) of each algorithm on each real dataset: Table 7.3

Graph	GainPrcntg GreedyExt %	GainPrcntg GNN %	GainPrcntg Random %
Books	83	82	56
Karate	83	82	61
LiveJournal	52	52	7
Average	72.5	72	41

Table 7.3: GainPercentage for GreedyExt, GNN and Random Algorithm on the real datasets

The $GainRatio$ for the GNN Algorithm vs the *GreedyExt* Algorithm is:

$$GainRatio = \frac{AverageGainGreedyExt}{AverageGainGNN} = \frac{72.5}{72} \approx 1$$

This number is not the exact ratio because we have performed some rounding in the GainPercentages, but the real ratio would have been very close (it was 1.06 for the synthetic data). As a conclusion, we see that in both the real and synthetic data our algorithm follows closely the performance of *GreedyExt* in terms of polarisation.

7.4 Polarisation for Different K

In this section we provide some plots that show the relationship between the polarisation and K, and how they differ among the three algorithms that we have discussed so far. We set K to be approximately equal to 10% of the number of nodes in the graph.

In Figure 7.8 with the Political Books Graph we see how closely our algorithm follows *GreedyExt*. Furthermore it is a visualisation of our previous discussion regarding large K and Random Algorithm's performance. As K increases, the gap between the polarisation of the Random and the GNN algorithm becomes smaller.

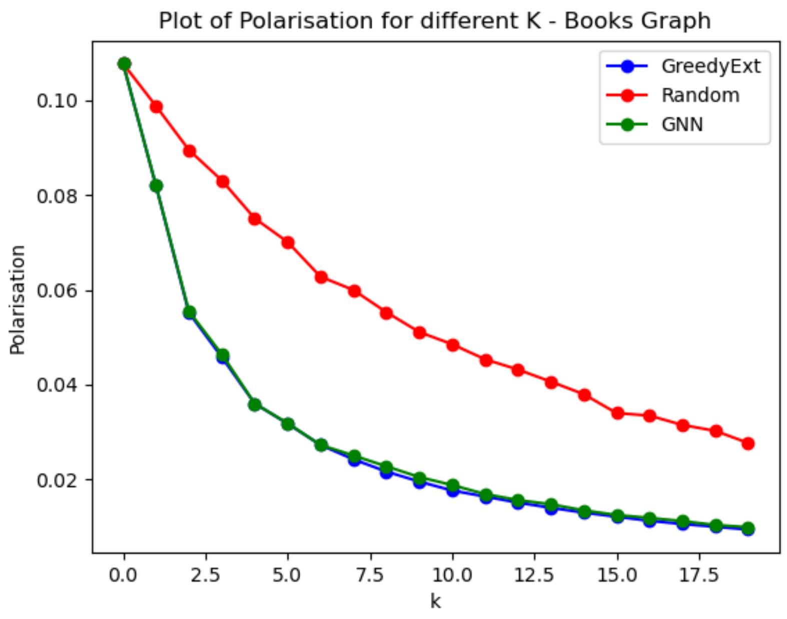


Figure 7.8: Polarisation vs. K for the Books dataset

Figure 7.9 yields an interesting result. If we look closely we will notice that the GNN Algorithm is better than GreedyExt for k=2. This result proves the sub-optimality of GreedyExt. There exists a set with 2 nodes that when we set them to zero, the final polarisation is lower than what the GreedyExt suggests. *GreedyExt* is a greedy algorithm. It is not far-sighted and at each timestep picks the node that yields the largest gain, without accounting for potentially even larger gains in the future - what we call long term rewards. Of course, the GNN Algorithm is better than *GreedyExt* for k=2 on this specific dataset by mistake. The GNN was trained to pick the best node greedily at each timestep. This is the same behavior as the *GreedyExt* algorithm. But as we mentioned earlier, the approximate nature of neural networks leads to "mistakes". Nevertheless, in this case the "local" mistake was actually beneficial for the underlying task.

In Figure 7.10 we show the corresponding plot for the LiveJournal dataset and $K = 70$.

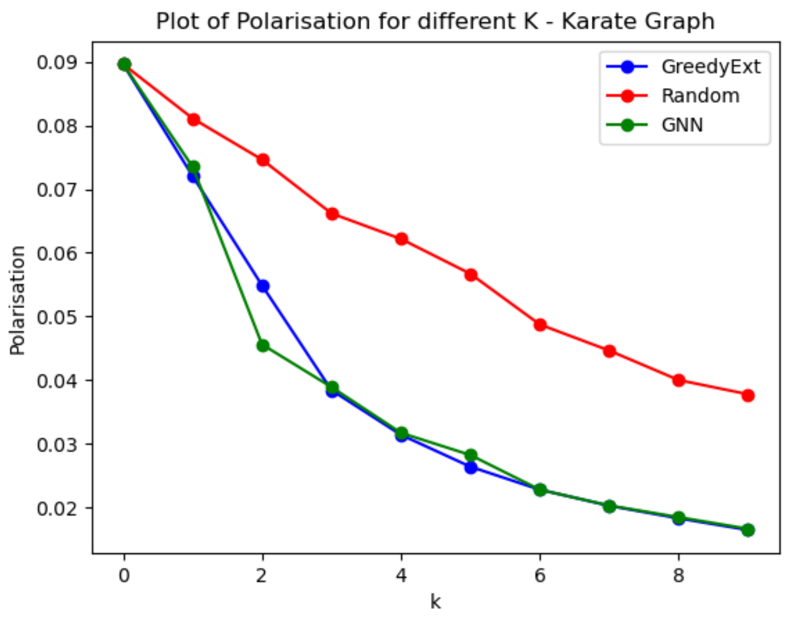


Figure 7.9: Polarisation vs. K for the Karate dataset

We limited K to this small value because of the long time required by the Random Algorithm to complete. Specifically, in order to get accurate results for the random algorithm, for each k we have to run it multiple times and compute the average of the final polarisation. However, again we observe that the GNN Algorithm follows closely the *GreedyExt*, even for this realtively large dataset.

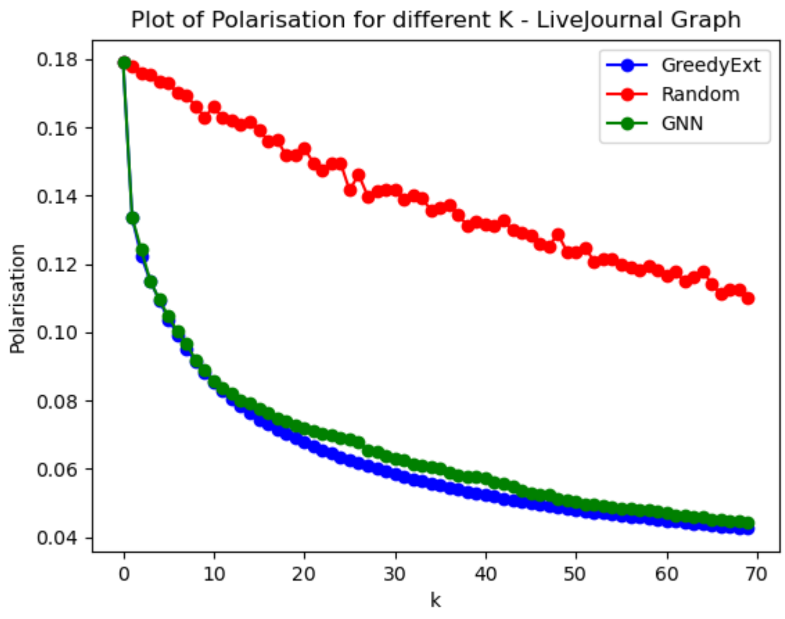


Figure 7.10: Polarisation vs. K for the LiveJournal Dataset $K = 70$

Finally, the Figures 7.11 and 7.12 shows the results of the algorithms on custom polarised graphs with 200 nodes. The graphs were generated with the Approach 1 and 2 that we discussed earlier. We confirm that the GNN Algorithm slightly deviates from the *GreedyExt*. The main reason why we think this happens realtes back to the Gain Error in Figure 6.2. As we have already mentioned multiple times, our architecture cannot become extremely good in the gain regression task, due to the limited information that it has. If we focus on the Gain Error figures for the real datasets (Figures 6.3, 6.4) we will notice that 2-hop information is enough to bring the gain error close to zero. This means that, our model could work really well with those datasets, as it does. In contrast, the gain error of the synthetic graphs (Figure 6.2) is relatively high for $l = 2$. Based on this, we can justify some deviation when it comes to the synthetic graphs.

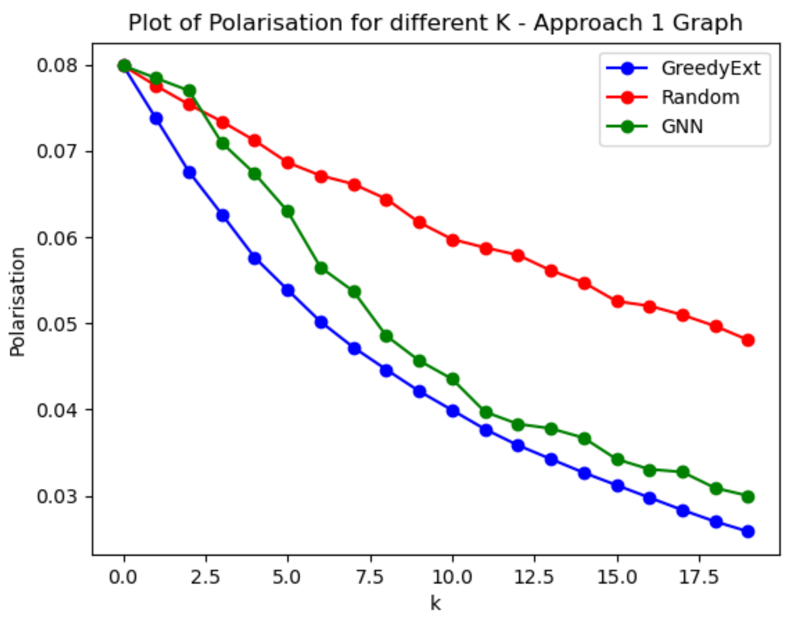


Figure 7.11: Polarisation vs K for Synthetic Graph generated with Approach 1

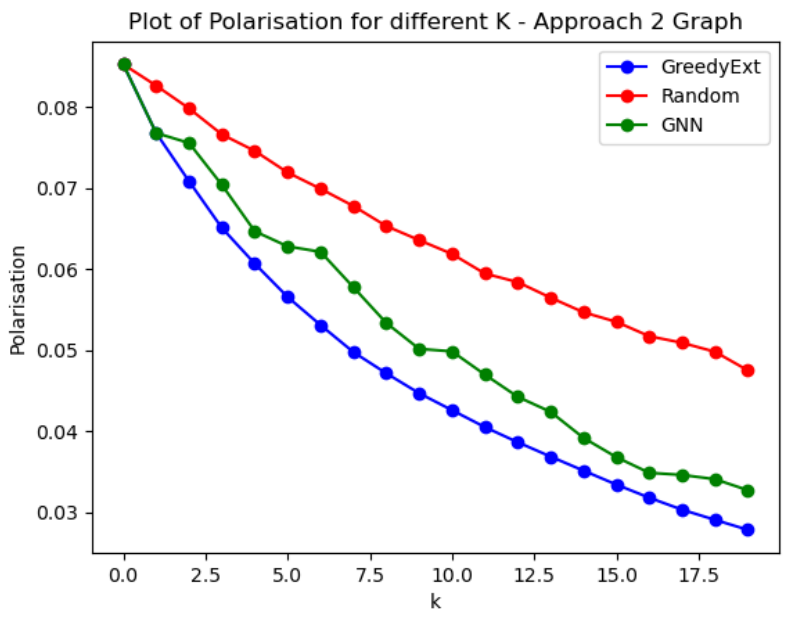


Figure 7.12: Polarisation vs K for Synthetic Graph generated with Approach 2

To further support our previous statement we bring our focus back to the model GCN1.4 that is a Graph Convolutional Network with 4 layers and 128 dimensional hidden units. Moreover this model was the best in terms of developoment loss on the regression task. Based on what we've said so far this model should work better with the synthetic graphs because:

1. It was the best at the gain regression task
2. It has 4 layers, meaning that it captures information from up to 4-hops away, and based on the gain error for the synthetic graphs (Figure 6.2) it should have an advantage

Our intuition is confirmed according to Figure 7.13

As we can see there, the 4 layer architecture has an advantage on the synthetic graphs compared to the simpler 2 layer model. In the next section however, we will compare the two models on a different dimension: the time they require to find the solution.

7.5 GNN Algorithm vs GreedyExt in terms of time

So far, we have demonstrated that our algorithm performs similarly to the *GreedyExt* algorithm when it comes to reducing polarization. In this section, however, we shift our

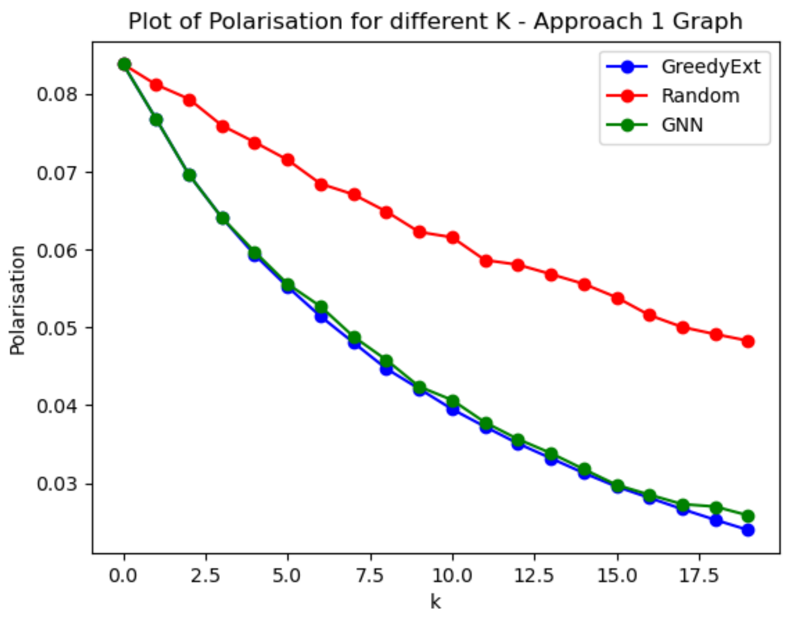


Figure 7.13: Polarisation vs K for Synthetic Graph generated with Approach 1 for the GCN1.4 model

focus to show that our approach significantly outperforms *GreedyExt* in terms of execution time and scalability. Specifically, we provide a plot that illustrates the performance of both algorithms on graphs of varying sizes.

As seen in Figure 7.14, the GNN-based algorithm demonstrates a clear advantage over *GreedyExt* in terms of runtime. For larger graphs, the runtime of *GreedyExt* increases dramatically, making it highly impractical for handling large datasets. In contrast, the time required by our GNN Algorithm scales much more efficiently, showing only minimal increases as graph size grows. This behavior aligns with expectations, since during the test phase, the GNN Algorithm only needs to perform a single forward pass through the network, followed by an argmax operation over the nodes to identify the best one.

The advantage of our algorithm in terms of time is quantified with the metric $TimeRatio$. For a graph with $|V| = 1200$ the time improvement is $TimeRatio = \frac{SecondsForGreedyExt}{SecondsForGNN} = \frac{15}{1.5} = 10$. For a graph with $|V| = 1500$ the time improvement is $TimeRatio = \frac{SecondsForGreedyExt}{SecondsForGNN} = \frac{25}{2.5} = 10$.

Overall, our algorithm achieves:

1. $GainRatio = \frac{1}{1.06} = 0.94$. This means that the decrease in the polarisation from GNN is 94% of the decrease that the GNN achieves.

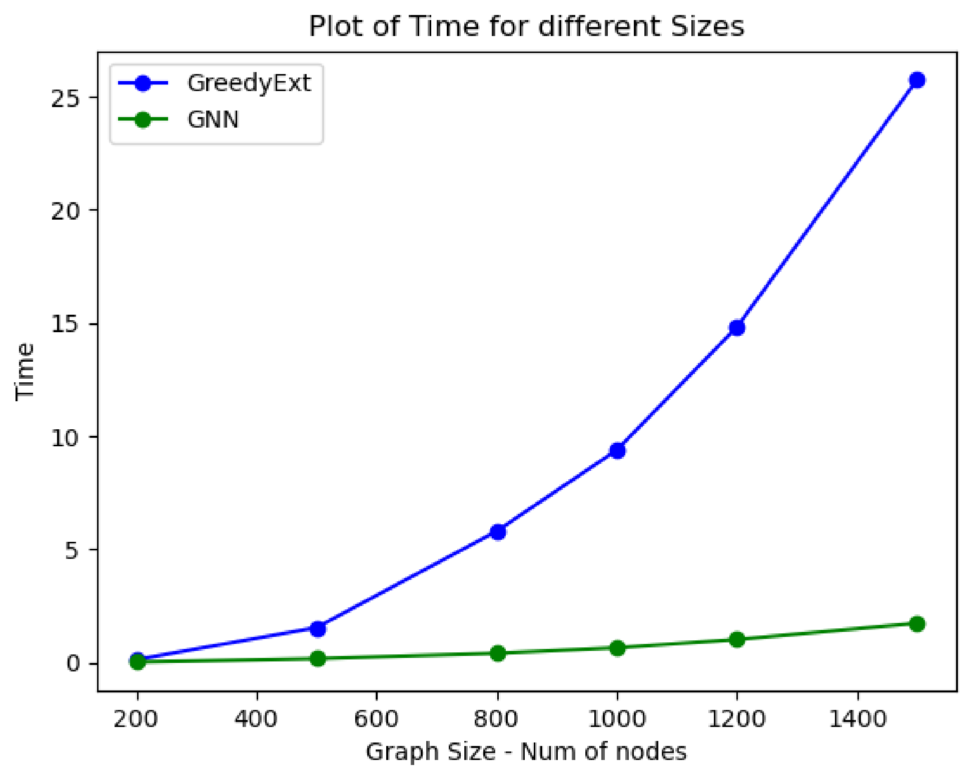


Figure 7.14: Time (seconds) to complete vs Graph Size $|V|$

2. $TimeRatio = 10$, which means that it does the job 10 times faster than the *GreedyExt*.

This is a significant improvement in terms of time, without losing much of the performance in the polarisation decrease.

Moreover, the PyTorch Geometric library that we use for our GNN implementation benefits from highly efficient sparse matrix operations, which further boosts performance. This is particularly important when dealing with large graphs, where computational efficiency becomes critical. It's worth noting that while the current experiments were run on CPU, even greater performance improvements could be expected if we leveraged GPU acceleration. The reason for this is that, the computation for each embedding depends only on the embeddings of other nodes from the previous layers. This means that the computation of the embeddings for a specific layer can be parallelized.

As the plot illustrates, our GNN-based approach maintains a significant advantage in terms of scalability, providing a more practical solution for larger graphs compared to *GreedyExt*. Even without using GPUs, the GNN Algorithm already outperforms the traditional method, making it far more suitable for large-scale problems.

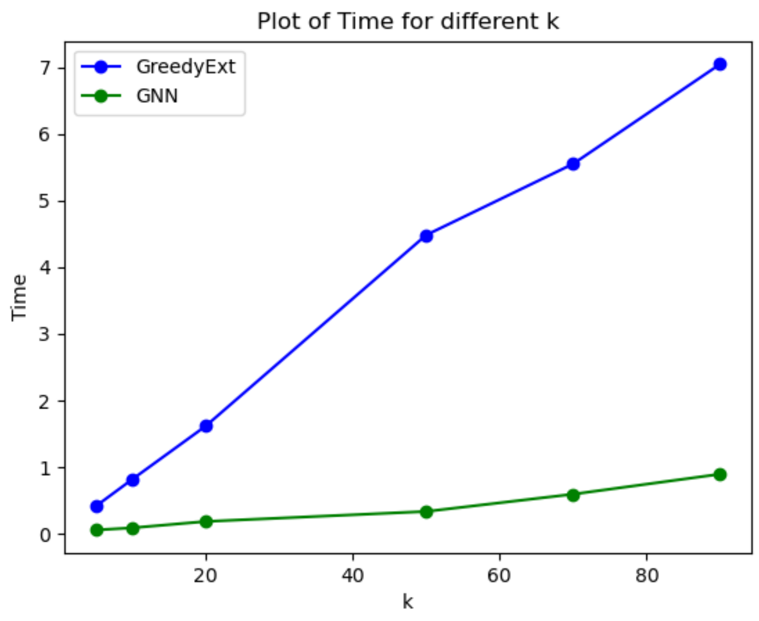


Figure 7.15: Time (seconds) to complete vs K

We also explore how the required time behaves with respect to k. This time we sample a graph with 500 nodes and store how long it takes for each algorithm to solve the problem for various k. The results are demonstrated in Figure 7.15. Again, the GNN Algorithm outperforms *GreedyExt* in terms of time as k increases.

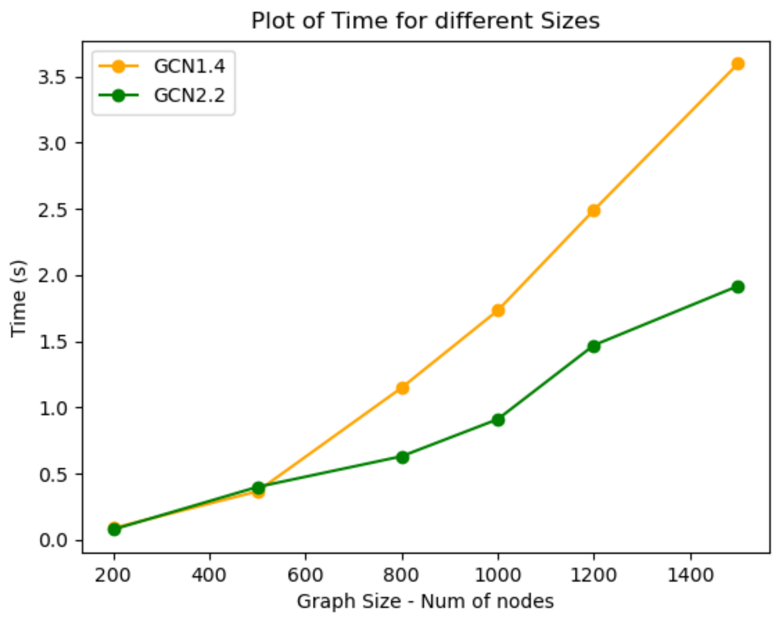


Figure 7.16: Time (seconds) to complete vs K for GCN1.4 vs GCN2.2

Finally we compare now the GCN1.4 with the GCN2.2 in terms of time, in Figure 7.16

The time ratio in this case is:

$$TimeRatio = \frac{SecondsForGCN2.2}{SecondsForGCN1.4} = \frac{3.6}{1.8} = 2$$

This means that the 2 layer model is $2\times$ faster than the 4 layer model. According to what we've discussed so far we can make the following conclusion regarding the two models: The GCN1.4 performs better in terms of regression. It is a way more complex model and for this purpose it performs slightly better with the synthetic data. However, when it comes to real graphs, the way simpler model GCN2.2 achieves similar gains. Moreover, in terms of time, the GCN2.2 model is two times faster than the GCN1.4. Overall GCN2.2 doesn't lose significant accuracy while it maintains a clear advantage regarding the time required against both the *GreedyExt* and the GCN1.4.4

Chapter 8

Conclusions

8.1 Conclusions

In this thesis, we proposed an alternative approach to address the *ModerateExpressed* problem, originally introduced in [5]. One of the key limitations of their solution is its lack of scalability, as their algorithm becomes increasingly slow, when applied to larger graphs. Our method leverages Graph Neural Networks (GNNs), specifically a 2-layer Graph Convolutional Network (GCN), to generate node embeddings. Instead of using brute force evaluation, we apply a simple argmax function to identify the node with the highest score. In terms of polarization, our approach achieves results comparable to those of the original algorithm. However, the main advantage of our method is its ability to scale efficiently to larger graphs due to the GNN architecture.

Additionally, while the datasets used in [5] were limited to fixed opinions of $+1$ and -1, our work allows for opinions that span the entire range of $[-1, 1]$. We evaluated our model on both synthetic and real-world data, demonstrating its robustness in various scenarios. Furthermore, our GCN performed well on both weighted and unweighted graphs. We also presented an example illustrating the suboptimality of the original greedy approach.

8.2 Future Prospects of Our Work

Looking ahead, an exciting direction for future research would be to explore the combination of Graph Neural Networks and reinforcement learning for solving NP-hard discrete optimization problems, like the one discussed here. While some preliminary work exists in this area, it remains relatively underdeveloped and holds significant potential. Lastly, the algorithm presented in [5] is optimized based on specific linear algebra properties related to the opinion dynamics model they assume (the Friedkin and Johnsen model). While we also employed this model, we are confident that our GNN-based approach is model-agnostic. This means it can be trained on different opinion dynamics models, as long as the evolution of opinions depends on the structure and features of each node's local neighborhood.

References

[1] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[2] E. Bakshy, S. Messing, and L. A. Adamic, "Exposure to ideologically diverse news and opinion on facebook," *Science*, vol. 348, no. 6239, pp. 1130–1132, 2015.

[3] A. Tommasel, J. M. Rodriguez, and D. Godoy, "I want to break free! recommending friends from outside the echo chamber," in *Proceedings of the 15th ACM Conference on Recommender Systems*, 2021, pp. 23–33.

[4] F. Cinus, M. Minici, C. Monti, and F. Bonchi, "The effect of people recommenders on echo chambers and polarization," in *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 16, 2022, pp. 90–101.

[5] A. Matakos, E. Terzi, and P. Tsaparas, "Measuring and moderating opinion polarization in social networks," *Data Mining and Knowledge Discovery*, vol. 31, pp. 1480–1505, 2017.

[6] D. Daniels, A. Imdad, T. Buscemi-Kimmins, D. Vitale, U. Rani, E. Darabaner, A. Shaw, and J. Shaw, "Vaccine hesitancy in the refugee, immigrant, and migrant population in the united states: A systematic review and meta-analysis," *Human vaccines & immunotherapeutics*, vol. 18, no. 6, p. 2131168, 2022.

[7] D. M. Boyd and N. B. Ellison, "Social network sites: Definition, history, and scholarship," *Journal of computer-mediated Communication*, vol. 13, no. 1, pp. 210–230, 2007.

[8] K. H. Jamieson, *Echo chamber: Rush Limbaugh and the conservative media establishment*. Oxford University Press, 2008.

[9] M. H. DeGroot, "Reaching a consensus," *Journal of the American Statistical association*, vol. 69, no. 345, pp. 118–121, 1974.

[10] N. E. Friedkin and E. C. Johnsen, "Social influence and opinions," *Journal of mathematical sociology*, vol. 15, no. 3-4, pp. 193–206, 1990.

[11] C. Musco, C. Musco, and C. E. Tsourakakis, "Minimizing polarization and disagreement in social networks," in *Proceedings of the 2018 world wide web conference*, 2018, pp. 369–378.

[12] A. Vendeville, A. Giovanidis, E. Papanastasiou, and B. Guedj, "Opening up echo chambers via optimal content recommendation," in *International Conference on Complex Networks and Their Applications*. Springer, 2022, pp. 74–85.

[13] A. Czaplicka, C. Charalambous, R. Toral, and M. San Miguel, "Biased-voter model: How persuasive a small group can be?" *Chaos, Solitons Fractals*, vol. 161, p. 112363, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0960077922005732

[14] G. Deffuant, D. Neau, F. Amblard, and G. Weisbuch, "Mixing beliefs among interacting agents," *Advances in Complex Systems*, vol. 3, no. 01n04, pp. 87–98, 2000.

[15] K. Garimella, G. D. F. Morales, A. Gionis, and M. Mathioudakis, "Quantifying controversy on social media," *ACM Transactions on Social Computing*, vol. 1, no. 1, pp. 1–27, 2018.

[16] P. Guerra, W. Meira Jr, C. Cardie, and R. Kleinberg, "A measure of polarization on social media networks based on community boundaries," in *Proceedings of the international AAAI conference on web and social media*, vol. 7, no. 1, 2013, pp. 215–224.

[17] D. Bindel, J. Kleinberg, and S. Oren, "How bad is forming your own opinion?" *Games and Economic Behavior*, vol. 92, pp. 248–265, 2015.

[18] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl

[19] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[20] M. E. Newman, *Networks: an introduction*. Oxford university press, 2010.

[21] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," in *Proceedings of the ACM SIGKDD workshop on mining data semantics*, 2012, pp. 1–8.

[22] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph Attention Networks,” *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ