Epidemic study based on Community Detection algorithms

Rares-Cristian Neagu - 001136056
Supervised by
Dr. Harry Triantafyllidis

A dissertation submitted in partial fulfilment of the University of Greenwich undergraduate degree programme

Bachelor of Science (BSc)
University of Greenwich - School of Computing and Mathematical Science
April 2023

Abstract

This report presents how the combination of community detection algorithms and viral spread models is a practical method of studying the spread of an infectious disease within a social network. The Louvain algorithm is a great choice for detecting communities for this application, partitioning the network based on the relationships between its members. To simulate the spread of a virus within the network, the SEIR (Susceptible-Exposed-Infected-Recovered) model as applied on the corresponding graph, this model recreates the spread of a virus by dividing the nodes into four states and allowing the nodes to change states following some predefined rules. Applying the community detection method and the viral spread simulation on a hypothetical population found that the network topology has an important role in the spread of an infectious disease. Our study revealed that some communities have a higher preponderance to spread the virus or are at a higher risk of infection. This report presents the effects of the most popular preventive measures, including vaccination campaigns and quarantine orders, how effective they could be in stopping the spread of the virus, and possible methods of interpreting the simulation results in order to identify the communities that require the most attention. The results emphasize the importance of introducing community detection in epidemiology studies and how important the network configuration is in determining the best response measures.

Acknowledgements

I would like to express my gratitude to all those who have contributed to the successful completion of this report. First and foremost I would like to thank my academic supervisor for their support and encouragement throughout the research process. I would also like to acknowledge the support of my family, friends and everybody and everybody else that helped me find the strength and motivation to complete this important chapter of my life.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

In the past two years, the world has confronted a real viral threat represented by the Covid-19 virus. The effects of the virus, both social and economic, are felt by communities worldwide. Urban areas, that are by their nature big and dense represent epicentres for viral infections (Ningrum, Chotib, and Subroto 2022). Having a method to study the communities and the method they interact is paramount in creating suitable and effective protective measures. Community detection represents a method of partitioning a network into multiple sub-networks to help understand the function they serve inside the network and how the relationships between the members are formed and what they represent. Since every network can be represented as a community and vice-versa, community detection is a valuable tool in sociology, computer science, biology, and any other discipline that can represent a system as a network (Fortunato 2010). Studying the communities these viral agents act on is important to understand the impact a measure might have on containing the spread and the effects felt by the members of the community. This project aims to demonstrate how community detection could be used in analyzing the spread of a virus inside a network by using the conclusions extracted from the topology of the network. This method should reveal how communities are affected in case of an epidemic, who are the most vulnerable members, and what measures can be implemented in order to minimize the virus' impact on the community. The mission of demonstrating that some communities are more affected by an epidemic event and that some communities play a larger role in spreading it inside the network is an important task in creating a sustainable and appropriate response measure. Given the complex and complicated structure of human social networks, the analysis of these networks is a daunting task. There are different methods of simulating a spread inside a network, usually, they are categorized as susceptible, infected, recovered, referred to as the SIR model, or susceptible, infected, susceptible, referred to as the SIS model, or a combination of the two models; depending on the virus' method of spreading and the immune system's reaction to it. (Vargas-De-León 2011). The most appropriate model for this project would be the SIER model, defining a susceptible, infected, exposed, and recovered method of describing the virus method of operating inside a network. This model takes into account the exposed nodes as well as the infected and recovered ones. Considering the exposed nodes separately from the susceptible nodes fits with a community-oriented project like this one. Quantifying the number of exposed nodes is beneficial in understanding how the virus travels from one community to another and how the virus' action can be mitigated such that the exposed nodes are more and more protected, therefore blocking the malicious activity of the virus.

Chapter 2

Literature Review

As it was mentioned above, community detection is a critical tool in analyzing networks, regardless of the context. Despite the number of applications this problem has, solutions are still lacklustre and present only small aspects of the community. Large networks with a greater number of members and relationships are hard to examine and the computational power and time necessary to determine the communities are not yet generally available. Detecting the communities inside a larger network is necessary for analyzing the spread of a virus. Having a clear understanding of how these communities interact with each other and what their role is inside the network is the first step in studying the spread of a virus. The relationships between nodes and clusters of nodes ultimately decide how the virus spreads inside the network and how it can be mitigated. In recent years different methods of counteracting the spread of Covid-19 were implemented, including mask mandates, restricting gatherings and events, and closing schools and workplaces (OECD 2020). Changes inside a graph's structure could represent all these methods. These changes will reflect a different spread pattern, allowing the authorities to conclude new methods of preventing spread in a virtual space without causing harm in the real world. Understanding the means of transmission, a virus uses to infect a new node are as important for this task as the detection of communities. The problem is that every virus has a specific method of infecting a host. The changes made to the topology of the network must reflect the specific method the virus spreads. This project will be focused not on a specific virus, but on a method of changing the parameters of the viral spread to simulate the spread of a multitude of viruses. This project aims to be a tool used in analyzing spread patterns and methods of prevention. Using a virtual space to implement preventive measures will allow authorities to see how said measures affect the spread and how effective they could be if they would be implemented in a community. In the end, the conclusions extracted from this study should help understand how communities interact and how preventive measures affect said communities.

2.1 Graph Theory and Community detection

To study the network and to better see the communities emerging inside the network, the data is usually structured in graphs where every vertex represents a member of the network, and the edges represent the relationship between two nodes (Fortunato 2010). Conditioned by the expected result or the intention of the study, the graphs used in the literature are weighted or unweighted. A weighted graph adds another layer of complexity while increasing the accuracy of the relationships allowing prediction based on the relationship between the nodes. To analyze the spread of a virus, the weights of the graph will not add to the accuracy of the model in highlighting the transmission routes and the impact the disease might have on the communities, since the transmission of the virus does not depend on the type of relationship between two nodes, and any contact between two individuals can lead to a new infection. The project is divided into two parts, one part will be a community detection algorithm that would represent the basis for the second part, the viral

spread simulation. Having a clear understanding of how the communities interact, what nodes are a greater risk for the said community is a start in determining how said community could be protected from a viral agent. A multitude of algorithms has been proposed for solving community detection, all with different approaches, but with similar results. Some of the methods include greedy algorithms, flow-based algorithms, degree-based algorithms, etc. (Hollocou, Bonald, and Lelarge 2018). One of the metrics used in detecting the community inside a network is modularity. Modularity is measuring how well a subset of the graph is connected compared to the rest of the nodes in the graph. The values of modularity range from -1 to 1, where a higher value indicates that nodes are more strongly connected to each other than the rest of the graph, and a negative value represents a less connected community than what is expected in a random graph. A popular algorithm that uses modularity as a metric to find partitions of the graph that represent meaningful communities is the Louvain algorithm.(Blondel et al. 2008) This method works by assigning each node to its community and then iteratively assigning a node to a new community monitoring the change in modularity. The nodes are merged into communities until no further improvement is possible. This algorithm has been successfully used in various applications, including the analysis of online social networks to group users with similar interests and behaviours in order to improve the quality of service they are provided(Pujol, Erramilli, and Rodriguez 2009). In the training of Graph Convoluted Networks, the algorithm was used to label data with community information, which can be used to improve the performance of the network. The Louvain algorithm has proven that it can be an effective tool in partitioning graphs with a large number of nodes and it consequently become a widely used tool in the field of network analysis.

2.2 Viral Spread

For the second part, the viral spread simulation, another model will be trained to simulate the methods of infection as accurately as possible. This model will be trained using information collected from a multitude of government agencies and NGOs around the world during the Covid-19 pandemic (Chalseo 2023). This data contains spatial and temporal data that reflects the number of infections in each area in a specific timeframe. This would help the accuracy of the simulation by recreating a natural method of spreading. Using real-life data and not just a series of variables that define the behaviour of the virus makes the simulation closer to life and the conclusions drawn are more effective in understanding how efficient the preventive measures are. Using the knowledge accumulated in the learning phase, it would simulate how the virus might act in a given network. The simulation would be determined based on the characteristic usually observed in a virus, the probability to be exposed, infected and recovered from the disease. Besides the characteristics of the virus, it is important to consider the most popular preventive measures and the reaction the population will have when they are implemented. For example, some agents might not respect the preventive measures implemented. Changing the characteristics of the virus, as well as the population response to the response procedures offers the possibility of studying an epidemic episode in more detail. To reflect the reality as closely as possible, the virus spread would be simulated a large number of times on different networks and the conclusions will be drawn based on the results of this scenario simulation. This method of reproducing the viral spread allows for a more accurate representation of the possible transmission patterns and the overall dynamics of the virus inside a network. By the end, the user should have a series of statistical data representing the infection; some important metrics might be the average time a node was infected, the average time the virus was present in the network, the most vulnerable community, etc. The tool would also generate a visual aid based on the multitude of simulation run in the previous steps to better understand

the implications of the numerical data.

2.3 Evaluation

Evaluating a community detection algorithm means, in essence, analyzing a well-defined network, with a known community structure and recovering the communities. While there could be found a multitude of such networks in different open-source libraries, their number is not infinite. A well-known method of generating new, but different networks is the Lancichinetti Fortunato Radicci benchmark (Lancichinetti and Fortunato 2009). This method is one of the most used methods in assessing community detection algorithms' efficiency. It generates a new network with predictable communities, based on predefined parameters. While this method generates a synthetic network, the connections between the nodes are a good representation of real-life communities, making it a better choice compared to other methods such as the one proposed by Girvan and Newman(Girvan and Newman 2002). This method could be used in testing an algorithm for both a large and small number of edges and nodes, revealing the limitations of the algorithm, its strengths, and weaknesses. Having a clear understanding of the observational limitations of the model exposes the level at which the algorithm can function properly and what are the most appropriate type of networks to be used in order to obtain the maximum efficiency of this system.

Another simple method of gauging the effectiveness of this solution is by comparing the performance of this model with other published solutions. Using too much time or power to solve a problem can be a factor in deciding if it is worth considering this project as a viable solution to the proposed problem, therefore this model must be evaluated in terms of efficiency. Using the already publish solutions as a benchmark, it is possible to test how well this project uses its resources compared to different and similar methods. Using the same thought process, the model could be tested on the same networks as other proposed solutions and evaluate how well it performed in finding the communities. Using the same communities will reveal how exactly this project behaves both in performance and accuracy. For the viral spread algorithm, the best method of evaluation is in this case the comparison with other similar algorithms. Since the scope of this project does not allow a more board evaluation technique, other methods of testing will not reflect if the algorithm does in fact output an acceptable result, or if it respects the collected data during the pandemic.

2.4 Implementation

The implementation of this project is divided into two stages. The first stage is developing the community detection algorithm, the results of which will be used in the viral spread simulation. Using the NetworkX python library to handle the graph operations, such as dividing the graph into sub-graphs, accessing the number of edges or nodes in the graph, its adjacency matrix etc, the implementation of the Louvain method is reduced to following the aggregation and modularity gain steps described in the seminal paper. The second stage is implementing the (Susceptible-Exposed-Infected-Model) such that it can simulate a viral spread on a given network. One important aspect of this model is the multitude of simulation parameters that allow for a better understanding of the virus and the effects of the network structure in the context of an epidemic episode. The model will implement different methods of recreating popular preventive measures and present the results in appropriate metrics, presenting data about the virus activity inside the whole network and a more comprehensive analysis of each detected community. The users should be able to customise the properties of the virus and the population response to the protective procedures, in order to gain insight

into possible transmission routes and counteractive measures.

Chapter 3

Analysis

The Coronavirus pandemic that started in early 2020 had devastating effects, both economic and social, making it apparent that the whole world missed the necessary tools and methods to successfully combat such a threat. This report will explore how studying communities that emerged in a network can be an important step in developing protection measures for those individuals that are at a greater risk. The social nature of humans determines them to live their lives as part of communities of people that share the same interests, geographic location, or behaviours. Detecting these groups and how their members interact with each other and members of other communities can help policymakers in creating more effective preventive measures that are tailored to the needs of said communities.

Community detection is a method of analyzing the members of a network and categorizing them into groups that are based on their similarities. This analysis helps in understanding the role of each individual in their respective community and how their position influences the spread of information, or in this case a virus. One key aspect of community detection in the context of a viral spread is detecting the nodes that have the power to spread the disease further into the network as it allows public health officials to take tailored measures in order to stop the spread. Protective measures such as mask mandates, or vaccination campaigns are general measures that aid in the fight against the viral agent. However, these measures have an impact on individuals, the effects of which are harder to detect and deal with effectively, having the potential to create another health crisis. Therefore the protective measures implemented to protect a community must respect the specific needs of the members of each community, minimizing the negative impact the measure might have on the group.

Understanding not only how the virus acts inside the network, but how the communities the virus acts on interact with each other and how their members behave is an important factor in optimizing the response measures. The Louvain method is a well-known and widely used algorithm in network analysis. This algorithm is modularity-based, meaning its purpose is to maximize the modularity of the graph by assigning nodes to different communities until the metric cannot be improved further. The Louvain method is known for the speed it can detect the communities inside a network and how easily scalable is, begin successfully used in networks of impressive size. Its simplicity and effectiveness made it the preferred choice in analyzing networks in multiple scientific fields such as biology, chemistry, and sociology.

The Louvain algorithm consists of two main steps that are repeated until no further improvements can be made in maximizing the modularity. The first step is the agglomeration of nodes, the algorithm assigns each node to its own community and iteratively adds nodes into communities, calculating the new modularity score of this configuration. At each iteration, a node is merged with one of its neighbours into a single community and chooses the partition that presents the largest increase in modularity. In the second step, the algorithm produces a new graph where the nodes are the communities discovered in the first step. The first step is repeated using the generated graph in the last step; this process is repeated until no the configuration cannot produce any more improvements in modularity. The null model provides the baseline

for the comparison with the real network. This model preserves structural properties of the network, such as the degree distribution, but randomizes the connections between nodes. This allows us to test if the observed community structure is significantly different from what the null model shows. A higher modularity score in the real network than in the null model, resulting that the community structure of the network is not a result of randomness.

The modularity of a partition of a graph is a value between -1 and 1, the closer the value is to 1 the more links are inside a community compared with the links between communities. The formula used to calculate the modularity is

$$Q = \frac{1}{2m} \sum_{i,j} [A_{ij} - \frac{k_i k_j}{2m}] \delta(c_i, c_j),$$

where m is the total number of edges in the graph, $A_{i,j}$ is the weight of the edge between i and j, k_i is the sum of weights of edges attached to vertex i, c_i is the community of vertex i and the δ-function $\delta(u, v)$ is the Kronecker delta function defined as $\delta(u, v) = 1$ if $u = v$, or 0 if $u \neq v$. Even if the equation was defined for weighted graphs, the modularity of an unweighted partition is calculated similarly. The only significant changes are that A_{ij} is 1 if an edge between node i and j exists, and k_i becomes the degree of node i.

The gain in modularity, noted as ΔQ, obtained by partitioning the graph is calculated using the following equation:

$$\Delta Q = \left[\frac{\sum_{in} + k_{i,in}}{2m} - \left(\frac{\sum_{tot} + k_i}{2m} \right) \right] - \left[\frac{\sum_i}{2m} - \left(\frac{\sum_{tot}}{2m} \right)^2 - \left(\frac{k_i}{2m} \right)^2 \right],$$

where $\sum_{in}$ is the sum of weights of the links inside the community C, $\sum_{tot}$ is the sum of the weights incident to the nodes inside the community C, k_i represents the sum of the weights of the links incident to the node i, $k_{i,in}$ is the sum of the weights of the links from the vertex i to the node assigned to the community C and m is the sum of all the weights inside the network (Blondel et al. 2008).

It is important to note that this application does not use weighted edges or any node attributes. In this case, instead of using the weight of an edge, the application makes use the value present in the adjacency matrix.

The viral spread simulation component of this application utilizes the SEIR (Susceptible, Exposed, Infected, and Recovered) model to simulate the spread of the virus within the network. This categorization of nodes enables the users to understand not only how quickly the virus spreads, but also the number of nodes exposed to the virus. Focusing on the number of exposed individuals, the simulation provides even more accurate representations of the risks posed to the community and allows for the development of strategies that target not only the infected but also what exposed individuals. In this model the nodes are initially categorized as susceptible, meaning they can contact the virus and are at risk of developing the disease. When a susceptible node comes in contact with an infected individual the node becomes exposed, and after a period of time determined by a simulation parameter, the exposed nodes become infected, and, infected nodes recover and become immune to the virus.

Each time step in the simulation represents a day, and for each day the states of the nodes are updated based on their previous states and the relationships with other nodes. The rate at which the nodes change state, the time it takes for a node to recover from the illness and the success of quarantining infectious nodes and vaccination campaigns are parameters that can be modified by the users. These simulation parameters should provide for recreating multiple possible scenarios. This categorization of nodes into four distinct categories and the multitude of simulation parameters that can be modified allows the researchers

to better understand the transmission routes and the more important nodes in the context of spreading it further, identifying therefore the most effective response mechanisms and the most vulnerable members of the network. (Mwalili et al. 2020)

Chapter 4

Requirements Specification

This chapter describes the requirements for the Community Detection and SIER Viral Simulation software, which is designed to be a tool for studying infectious diseases in a social network by evaluating different preventive measures to mitigate the impact of the disease. The software uses graph theory to identify the communities present in the network and simulate the spread of a viral disease using A SIER epidemic model.

4.1 Scope

The purpose of this software is to provide a platform that will model existing communities representing the population and the impact a viral agent might have on the network. This platform will enable researchers, healthcare providers, and policymakers to test different interventions and consider their effects in a virtual space and tailoring them to the needs of the communities, decreasing the negative effects they might have on the population. The software will provide outputs that can help in the creation and implementation of public health policies, resource allocation, and community outreach efforts.

The software is composed of two main parts:

1. Community detection algorithms
2. SIER epidemic model

4.1.1 Community detection algorithms

The community detection algorithms will provide users with valuable insights about the network structure and its subgroups, eventually leading to better decisions regarding the interventions developed for combating the virus. The software will provide the following features:

- The ability to input a graph-like data structure representing a social network where the nodes represent individuals and edges represent the relationships between them.
- The ability to apply community detection algorithms to the network in order to identify the subgroups based on various metrics such as modularity, clustering coefficient, or centrality measures.
- The ability to export the network and its subgroups as separate files in order to facilitate further analysis or sharing the results with other implicated parties.

4.1.2 SEIR epidemic model

The viral simulation part of this software is represented by an SEIR (Susceptible-Exposed-Infected-Recovered) model. This model separates the population into the aforementioned categories. The software will simulate

the spread of the virus based on predefined parameters such as the transmission rate, incubation period, and recovery rate. The software will provide the following features:

- The ability to input a network representing the population the virus will act on and the communities detected in the previous step.
- The ability to simulate the spread of the virus according to the SEIR endemic model.
- the ability to visualize the spread of the virus over time, and yield relevant statistical data about the simulation.
- the ability to export the simulation results as a file for further analysis.

4.2 Functional Requirements

Functional Requirements represent the practical requirements of the software. In other words, is a collection of tasks and problems the software should solve in order to achieve the goal it was developed to accomplish. These tasks range from how data is managed and processed to how the output is formatted in order to represent the conclusions most efficiently.

The Functional Requirements of this tool are as follows:

1. Input Data: The software should allow users to input a graph representing the social network they wish to study
2. Community Detection: The software should incorporate detection algorithms suitable for the size and scope of the project
3. Visualisation: The software should output the results of the community analysis clearly and concisely, highlighting the underlying communities inside the network and the relations between them
4. Simulation Initialization: The software should allow the users to change the simulation parameters, such as infection rate, recovery rate and preventive measures variables, and the number of initially infected nodes to study a multitude of possible scenarios
5. Epidemic Model: The software should use the SIER epidemic model to simulate the spread of the virus taking into account the parameters defined by the users
6. Visualization of Epidemic Spread: The software should output the result of the simulations in a clear and concise method, highlighting both the impact the virus had on the network as well as how it affected the communities
7. Exportation: The software should provide the functionality to export the results to facilitate the study in other similar graph analysis tools

4.3 Non-Functional Requirements

Non-functional requirements represent a collection of software qualities and behaviours, focusing on the usability of the tool more than the actual methods of solving the problem at hand. These requirements

represent the use cases and boundaries of the tool, as well as other aspects such as the maintainability or scalability of the software.

The Non-Functional Requirements of this tool are as follows:

1. The software should provide a user the ability to import networks into the simulation and community detection components
2. The software should handle large datasets and provide the results of both the community detection and viral spread simulation components in a reasonable amount of time
3. The software should be reliable and produce consistent results
4. The software's architecture should allow for future improvements and additions

In summary, the Community Detection and SIER Viral Simulation tool will provide a powerful tool for analyzing the impact a viral agent has on a community and what measures are more suitable in order to reduce the number of infected nodes as well as reducing the negative effects of this measures on the members of the network. Overall this software will provide a valuable resource for the public health community, helping them prepare for and mitigate the impact of infectious diseases in communities around the world.

Chapter 5

Design

Designing software is a crucial aspect of developing it, ensuring it meets all the functional and non-functional requirements. In this case of community detection and epidemic modelling software, it is important to consider the design to ensure that is efficient, effective, and easy to use. The design needs to address how these two components interact with each other and how the user will be able to control each part of this system. The design needs to consider both the input and output data formats, how these data structures will be used by both the community detection and the viral spread simulation and how would the conclusions will be visualized in order to draw the right decisions about the preventive measures. Usability and user experience is an important considerations in the design of the software, The design needs to provide a clear and intuitive method that will allow the users to input the data, change the simulation parameters, run the simulation, and implement different protective measures. The design also needs to address how the software will provide information about the simulations, the eventual errors, or other notifications. In conclusion, the design of this tool needs to carefully consider the technical requirements, user interaction as well as efficiency and effectiveness of both systems.

5.1 Community Detection

Community detection is an important step in network analysis in the context of infectious disease transmission. Identifying groups of nodes that are more densely connected compared to the rest of the network can help in understanding disease transmission patterns and developing effective control strategies. However, the task of detecting communities is challenging and there is no one-size-fits-all method that works for all types of networks.

It is crucial to ensure that the community detection algorithm is reliable and consistent so that the conclusions extracted from the analysis reflect the real-world situation. The accuracy of community detection can affect the decisions made to protect the communities, as they may need different types of interventions to control the spread of a viral agent.

Using modularity as the main metric for creating partitions in the context of a viral spread simulation will lead to more concise and accurate results. Considering an edge to be a meeting between two nodes in a given day, the algorithm finds a collection of nodes that meet more often compared with the other nodes, regardless of the other attributes a node might have. Having groups of individuals group based on their relationships with the other nodes in the network reveals how important these connections can be in determining the possible effects of a virus inside a network, making the Louvain algorithm the perfect method for this application.

The application expects the users to import their graph in the *'.graphml'* format. This is one of the most used methods of storing a graph together with its attributes. The results of the algorithm, besides the dictionary representing the optimal partitioning found by the algorithm, contain two more different repre-

sentations of the graph. One is a *'.graphml'* file of the imported graph where each node has a new attribute called *'community'* representing the label of the community it is part of, and the other representation is a new graph where the nodes are the communities discovered and the edges represent the connection between the communities. The edges have a weight equal to the number of connections between the nodes of each community and nodes have a *'population'* attribute representing the number of nodes that are part of that particular community.

5.2 Viral Spread Simulation

The Viral Spread Simulation is an essential part of this project as it provides users with information about how how the virus spreads within the network, what impact it can have on the communities inside the network, and what preventive measures are appropriate in order to reduce the influence the virus has on the members of said communities. The simulation is designed to be a simple yet accurate model of the viral agent. The simplicity of the model is obtained by not taking into account other information about the nodes or the relationships between them, such as their age, previous medical conditions, overall health, etc. An edge between two nodes represents a possible meeting between those nodes on that particular day, a meeting that could lead to a new infection.

One of the key advantages of the Viral Spread Simulation is the ability to customize the simulation parameters that define the behaviour of the virus and network members. This feature is more useful for public health organizations and regulators than are tasked with analyzing a multitude of viruses, who can use this tool to study a wide variety of viruses by changing the parameters that define the characteristics of the virus. Furthermore, this tool offers the ability to test different public safety measures by providing a method of controlling the spread of the virus. For instance, they can simulate the effect of different social distancing measures, mask mandates, or vaccination campaigns by tweaking the parameters associated with each measure accordingly and studying the impact these measures have on the spread of the virus.

Using the SEIR mode to simulate the spread of infectious disease, public health officials can study the dynamics of a virus inside a network and take more informed decisions regarding the best route of action against a viral threat.

Chapter 6

Implementation

6.1 Community Detection

The Louvain algorithm is a widely used heuristic algorithm community detection method. It aims to maximize the modularity score of a network by merging communities iteratively until the greatest score is obtained. The simplicity and efficiency of this method make it a very scalable option, maintaining its accuracy on networks with large or small numbers of nodes and edges. To facilitate the implementation of the Louvain algorithm in this project, the method has been developed using a Python class named *'communityDetection'*. This class makes it easier to access and modify the components of the algorithm at any given time, facilitating the eventual modifications and additions to the algorithm. Another benefit of using classes is that the other components of this project can access the information stored inside the class in a clean and organized way, making the whole project easier to maintain, optimize and experiment with. The *'communityDetection'* class is composed of three methods.

1. *'get_modularity'*, calculates the modularity of a given partition of the network, measuring therefore the quality of the partition.

2. *'merge_communities*, attempts to merge the pairs of communities that present the highest increase in modularity. This is achieved by iterating over pair of nodes in the network and assigning them to the same community. If the merge leads to a higher modularity, then the new partition is recorded and the process continues. This method is the most important part of this implementation, as this is where the optimal partition is created.

3. *'louvain'*, this method is used to start the process of merging nodes into communities and drives the algorithm forward. This method keeps calling the *merge_communities* procedure until a local maxima is achieved and the modularity cannot be improved. This method is also the one that returns the final partition of nodes.

One of the biggest qualities of this algorithm is the speed at which it can find the communities inside a network, however in the first stages of development, the algorithm returns acceptable results only for small graphs with several nodes lower than 100. While it can be useful in analyzing some communities, maybe some parts of a smaller town, it cannot be an effective tool in the study of networks representing entire cities, graphs that have thousands or even hundreds of thousands of nodes, and impressively more edges.

Analysing the time complexity of this approach is not trivial to determine, the algorithm is iterating over nodes, edges and communities, making the analysis more complicated. To make some general observations:

1. The *'get_modularity'* function loops over each community in the partition data structure. Within each loop, the function calculates the number of edges and degrees of one community. Considering that these calculations have a complexity as high as $O(E)$ for the number of edges and $O(D)$ for the calculation

of the degrees, concluding that the overall time complexity of this function is $O(C \times (E + D))$, where C is the number of communities in the current partition.

2. The *'merge_communities'* is composed of two nested loops that iterate over all the pairs of neighbouring nodes in the graph. The cost of this operation is $O(E)$ where E is the number of nodes; For each iteration, the function copied the current partition and starts merging communities, calculating the new modularity. Using the complexity observed in the *'get_modularity'* function it denotes that the overall time complexity of this function is $O(E * C * (E + D))$ since the *'get_modularity'* function is called for each edge in the graph.

3. The *'louvain'* method calls the *'merge_communitites'* unill no improvements in modularity are observed. Since there are at most E merges possible, the maximum number of iterations in this function is equal to the number of edges in the graph. Therefore, the time complexity of this implementation is $O(E^2 \times C \times (E + D))$

With a time complexity of $O(E^2 \times C \times (E+D)$, it is expected for this implementation to take an extremely long time to find communities with a large number of nodes and edges. Since E^2 can be much larger than $C \times (E + D)$, the complexity might reach $O(E^3)$, making it impracticable for large and complex networks. Therefore some ways of improving the time complexity of this algorithm, are needed. One study finds that most of the time spent in the early stages of finding the communities is occupied by moving nodes into their neighbouring communities and in addition to that shortcoming, the algorithm also spends a considerable amount of time going over the edges of nodes that are part of the same communities. Traag 2015 These two factors combined are slowing down the algorithm considerably without returning any rewards.

6.2 Viral Spread Simulation

6.2.1 Simulation Mechanisms

The Viral Spread Simulation is implemented using a Python class. This method was preferred since all the simulation parameters and metric data structures are easily accessible at any point of the simulation, decreasing the complexity of the code and increasing its readability. The class defines several variables that dictate how the virus behaves in the network:

- the rate of exposure, named *prob_e*, dictates the probability of a node contacting the virus given contact with an infected node
- the rate of infection, named *prob_i*, dictates the probability of a node developing the disease and spreading it further into the network
- the rate of recovery, named *prob_r*, dictates the probability of a node recovering from the disease and becoming immune to it

These variables reflect how a real viral agent might act in the real world and can be changed based on real data collected during past or present infections. This allows the researchers to adjust the parameters in order to study possible variations of the virus or a different virus entirely. The simulation class also keeps track of when an individual changes states and the probability of the node changing to the next step is determined by the transition time.

The simulation starts with all nodes in the susceptible state, making them all vulnerable, besides one random node that is infected. Each iteration of the simulation represents a day and is composed of these steps:

1. Exposure Step: Each infected node's neighbours have a chance of contracting the virus, making them exposed. The exposure probability is increased with each day the infected node didn't recover, making them more infectious as time passes.

2. Infection Step: In this step, the exposed nodes develop the symptoms of the disease and become infectious. The probability of a node becoming infected decreases with each day it passes, being considered susceptible again after 3 iterations.

3. Recovery Step: In this step the infected nodes are set to a recovered state, making them immune to the virus and can no longer spread the virus. The recovery process can start after a determined number of steps have been passed.

Some control measures, such as quarantining the infected nodes or introducing vaccines to reduce the number of infected nodes do not have a clear equivalent in the simulation parameters. To solve this problem, the model implements two different methods that reflect the changes in the viral spread determined by these preventative measures.

First, the model keeps track of two additional states, vaccinated and quarantined nodes. These states dictate if the node can contact or spread the virus. More precisely, after a node is quarantined it is no longer able to infect other nodes and a vaccinated node will not change its state to infected, regardless of how many times it is exposed to the viurs. These new states are incorporated into the model and are simulating the vaccination campaigns and quarantine measures imposed by the Centers for Disease Control. Secondly, in each simulation step, two methods new steps have been introduced, the vaccination step and the quarantine step. In these steps, a node becomes vaccinated if it is susceptible to the virus or enters the quarantine state if it is already infected with the virus. However, even these additional measures have some limitations as there will always be some pushback from the population, and some individuals may ignore the quarantine they have been placed in or refuse to get the vaccine. To account for this behaviour, the model considers two parameters, *prob_v* and *prob_q*.

The *prob_v* simulation parameter represents how likely it is for a node to get the vaccine, this parameter can be modified to represent different scenarios and open the possibility to explore situations in which the population is more or less receptive to the vaccine. The *prob_q* parameter represents the likelihood that an individual will respect the quarantine orders and interrupt contact with other members of the population. Just like the vaccination parameter, *pro_q* can be modified to reflect stricter or more relaxed policies.

Including these additional states, parameters, and methods, the model can simulate a simulation that is closer to real-world situations, giving the researchers the possibility to tune both the virus' and population's behaviours.

6.2.2 Data Visualisation

The point of the simulation is to understand the virus transmission routes and to discover those in the most vulnerable position. One single simulation is not enough to get a clear understanding of how the viral agent behaves in the network and what is the real danger the communities are exposed to. In order to get the bigger picture a series of simulations must be run and compared to develop the most effective protective

actions. This tool offers a series of statistical data aimed to analyze the simulations and the effects of the measures implemented in the model. The statistical data offered by the simulation part of this project is:

1. Average Number of infections: This is the average of the total number of infections throughout all simulations
2. Average Infection Rate: This is the average proportion of the population that gets infected throughout the simulations
3. Average Time to Peak Infections: This is the average simulation steps required to reach the maximum number of infected nodes
4. Average Reproduction Number: This is the average number of infected nodes by a single individual
5. Average Duration of the epidemic: The average length of time it takes the epidemic to subside
6. Super-Spreaders: The nodes with the most recorded infections throughout the simulations

The above-mentioned pieces of data are descriptive of the simulation and are important in analyzing the behaviour of the virus, but they are not characterizing the impact the disease has on the communities, therefore the application provides data related specifically to community safety.

1. The average number of individuals in each state for each community:
2. The community that has the most infectious nodes
3. The average proportion of members of a community that become infected during the epidemic

The format of the data presented is as important as the processes it describes. The data has to be clear, concise, and easy to understand to facilitate the design of new and more effective measures and not waste the users' time and resources on understanding the data. Wherever possible, the data will be displayed in a visual format avoiding vague or unclear legends and labels.

While this application can be useful in simulating the spread of a virus, it may not be the best tool for the analysis of the data resulting from the simulations. Therefore, both the raw data recorded during the simulations and the statistical information extracted can be exported to facilitate further analysis of the simulation. Giving the researchers the possibility to use the information in order to extract even more specific information related to the virus's behaviour and the risk it poses to the population.

Chapter 7

Testing and Integration

Testing and Integration are important steps in the development of any software tool. These steps ensure that the functional and non-functional requirements are met and that, in this case, the tool reaches the highest quality possible within the constraints. In designing and implementing this software the components that form the final product were considered separately, each component having its requirements and problem to solve. Nonetheless, these components are in constant cooperation and are exchanging information throughout the life of the application. Integration is the step in which the two main components of the application are set up to exchange information and the software can accomplish the task it was developed to do from start to finish. This step requires additional testing, different from the unit testing. The testing in this development stage focuses on the behaviour of the algorithms together, if the final results are correct and what is the actual limitation of the application.

7.1 Community detection

Testing Methods

This tool is composed of two parts, both having different requirements and functions that when pieced together result in a series of conclusions about the network and the possible behaviour of a viral agent. When it comes to testing, the community detection component has more strict error margins and an erroneous result will compromise the results of the second part. The community detection algorithm must be reliable and consistent. Since communities are an abstract concept and a person can belong to multiple communities at once it is important to define the communities and what metrics are most relevant in studying the network's topology to find said communities. The decision made for this report was to use modularity, as this metric considers the number of edges inside a community compared with the rest of the graph. Moreover, the Louvain method uses heuristics to find the best partitioning of the graph. That means that the algorithm might not find the best partitioning possible, only the one that results in the maximum value of modularity, making it more sensible to the initial position and the complexity of the network. Taking this into account, determining if the algorithm can approach the best partitioning most of the time becomes a crucial step in the development of this application. To achieve this the algorithm will be tested on a series of synthetic graphs generated specifically to test the accuracy of this method. These graphs have been generated with a known number of nodes and communities, allowing us to observe how the algorithm behaves when it is used on graphs with a larger number of nodes or communities. The Lancichinetti-Fortunato-Radicchi benchmark, also known as LFR-Benchmark, is a method of generating benchmark graphs that can be used to test the accuracy of a community detection algorithm. These graphs are generated with a known community structure, where nodes are assigned a community and the edges between nodes are created according to a set of rules.(Lancichinetti, Fortunato, and Radicchi 2008) The benchmark allows us to change the graph parameters, such as the size of communities and the relationships between them vary, creating graphs with

different levels of size and complexity. The community detection algorithm is used to find the communities on these synthetic graphs and the results are compared with the already-known labels.

A problem with this method of validating the result of our algorithm is that the final labels returned by our algorithm might differ from those recorded in the benchmark graph while the nodes are divided into the correct partition. This is known as relabeling error. Evaluating the algorithm's performance to find the underlying communities in a graph is more important than making sure the labels correspond.

One way to measure the actual partitioning is to use metrics that consider the amount of data clustered together. A metric that performs such a test is the Normalized Mutual Information (NMI). This test measures the similarity between the ground truth labels and the ones returned by the algorithm while considering that the partitions might have different labels. It achieves this by measuring the similarities in the information contained in the label sets. Normalized Mutual Information can be obtained using the formula

$$NMI(A, B) = \frac{2MI(A; B)}{H(A) + H(B)}$$

, where $H(X)$ is the entropy, amount of information, of the set H and $MI(A; B)$ is the Mutual Information of the two sets, defined as:

$$MI(A; B) = H(A) - H(A|B),$$

where $H(A|B)$ is the conditional entropy of the sets A and B.

Another method repeatedly used to test community detection algorithms is the Purity Score. Purity maps each community obtained by the algorithm to the ground-truth one with the highest overlap, calculating the percentage of correctly formed communities. Purity is obtained with the following formula:

$$Purity(A, B) = \frac{1}{n} \sum_{k} max_r |B_k \cap A_r|,$$

where n is the number of nodes, $A = A_1, A_2, ..., A_n$ and $B = B_1, B_2, ..., B_n$ are the set of community labels of each node detected by the algorithm and the ground truth labels respectively.

The last testing method used to validate the results of our community detection algorithm is the Rand Index. Compared with the previous method, the Rand index calculates what nodes were not considered part of the same community and what nodes were. The Rand index is defined as follows:

$$Rand(A, B) = \frac{2(a + d)}{n(n - 1)},$$

where n in the number of nodes, $A = A_1, A_2, ..., A_c$ and $B = B_1, B_2, ..., B_c*$ are the set of community labels resulting from the algorithm and the ground truth labels respectively, a is the number of pairs of nodes that are in the same community in both A and B, and d is the number of pairs of nodes that are no in the same community. (Liu, Cheng, and Zhang 2019)

It is important to note that using only a single evaluation metric to determine the accuracy of a community detection method may not provide a complete understanding of its abilities. Careful consideration of all the metrics can contribute to a more thorough evaluation of the algorithm's performance, each taking into consideration different factors of the classification problem.

Test Results

Using the LFR benchmark method explained in the previous section a number of graphs were generated, each with a different number of edges and nodes that are being used to test the accuracy of our algorithm.

Having a different number of nodes helps examine how will the algorithm perform when the number of nodes and edges increases. The graphs used to test the performance of our algorithm are described in table 7.1. The number of communities is not proportional to the number of nodes, which allows the analysis of the behaviour of the algorithm much more accurately.

Test Graphs			
Graph Name	No. of nodes	No of edges	No. of communities
LFR-50	50	60	12
LFR-60	60	126	3
LFR-70	70	146	3
LFR-80	80	144	8
LFR-90	90	187	6
LFR-100	100	219	12

Table 7.1: Graphs used to test the algorithm.

Using the method mentioned in the *Test Methods* section of this chapter some ideas of how good is our application at finding the best partition of the graph and how close it is to the real answer can be formed. In order to gauge the effectiveness of our method, comparing the results of our method with the results returned by some of the community detection methods available in the NetworkX module will determine if this implementation achieves similar results compared to some of the most available methods. For this test, *'greedy modularity'* and *'label propagation'* methods were used to determine the performance of this project.

Overall, the Louvain method has reasonably good results on smaller graphs, but its performance deteriorates when the number of nodes and edges increases. In terms of NMI, Purty and Rand Index scores, the Louvain generally performed slightly better or similarly, compared with the other two methods for graphs up to 70 nodes When the number of nodes increased the accuracy of all methods is affected, but the lowest results are the ones recorded by this application. This indicates that in larger networks, this implementation of the Louvain algorithm struggles to find meaningful communities.

7.2 Viral Spread Simulation

Testing Methods

Simulating a viral spread is a complex process that implies a multitude of different steps. Determining if the simulation runs correctly and each step is executed accurately is not an easy task. It is challenging to determine if the simulation is operating as expected or if there are logical errors in the algorithm since each simulation outcome is unique. Therefore some method of validating the result of this application simulation component were needed.

One method used to determine if the simulation is running correctly is to monitor the changes in the outcome when the simulation parameters differ. This method allows us to see if the adjustments of simulation parameters are implemented correctly and if relevant differences are observed. However this method is tedious, and it does not reveal if the simulation works properly, only if the variations in simulation parameters are correctly accounted for. Another method of validation results is to sample some of the simulations and observe the number of nodes in each state at every point in time. This approach allows us to determine if

Test Results					
Algorithm	Graph	NMI	Purity	Rand	Detected Communities
Louvain	LFR-50	0.60	0.52	0.85	12
	LFR-60	0.26	0.71	0.62	20
	LFR-70	0.23	0.64	0.66	20
	LFR-80	0.51	0.61	0.85	22
	LFR-90	0.04	0.31	0.25	3
	LFR-100	0.23	0.37	0.36	7
Label Propagation	LFR-50	0.58	0.52	0.85	16
	LFR-60	0.05	0.71	0.41	4
	LFR-70	0.06	0.64	0.36	3
	LFR-80	0.53	0.61	0.86	21
	LFR-90	0.66	0.31	0.66	15
	LFR-100	0.32	0.37	0.6	10
Greedy Modularity	LFR-50	0.65	0.52	0.88	12
	LFR-60	0.11	0.71	0.60	20
	LFR-70	0.06	0.64	0.61	6
	LFR-80	0.44	0.61	0.84	9
	LFR-90	0.24	0.31	0.77	8
	LFR-100	0.27	0.37	0.76	9

Table 7.2: Results of one test

the nodes change state correctly, exposing any miscalculations at one of the steps. However, this is also a very tedious and time-consuming method of gauging the accuracy of the simulation.

Testing results

In the early stages of testing, it was important to focus on the results of only one simulation at a time, it is crucial to make sure that one simulation can show us a possible outcome of the disease and that more simulations compared against each other will lead to a better understanding of the scenario described by the simulation parameters. In the start, the model of a very infectious virus with a low probability of recovery was simulated. The simulation parameters were set up the *prob_e* and *prob_i*, the probability of exposure and infection respectively, to be equal to 0.8, while the recovery probability was set to 0.2. This will result in a longer lifespan of the virus in the community and a larger number of nodes that end up infected. In this scenario, it was expected to see the number of infected nodes rise uncontrollably until the majority of nodes have contracted the disease. As it can be observed in Figure 7.1, the virus not only infected the majority of nodes but also lasted considerably longer than the scenarios described in Figure 7.2, where the effects of introducing the vaccine as the main preventive measure are recorded. When the population does receive the vaccine it was recorded that the number of infected nodes drastically decreases, but when the second preventive measure was implemented, the number of infections does not decrease as much, as Figure 7.3 reveals. This may show what are the most efficient methods of protection against a virus.

Next, it will be advisable to see the effects of changing the exposure or infection parameters, ensuring that the model can simulate viruses with different infection mechanisms.

After these tests, it can be stated confidently that the simulation aspect of this project does meet all the requirements and that it can simulate the spread of a virus based on the parameters selected by the user.

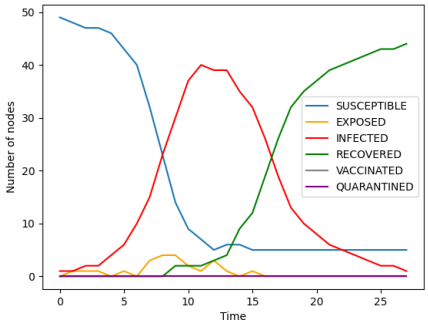

Figure 7.1: Viral spread on LFR-50 with no preventive measures

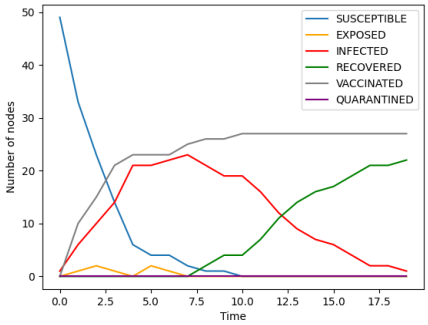

Figure 7.2: Viral spread on LFR-50 with vaccination as the main preventive measure

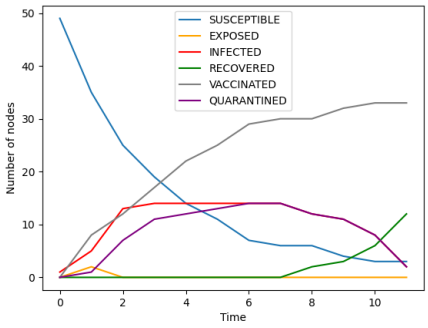

Figure 7.3: Viral spread on LFR-50 with vaccination and quarantine implemented

Chapter 8

Product Evaluation

8.1 Results Evaluation

Notice that even in the cases where the metrics were satisfactory, the number of communities discovered is too big or too little. This will affect the results of the viral spread simulation, lessening its ability to discover the best preventive practices. However, the community graph returned by the algorithm and the results of the simulation together can lead to the decision to treat the nodes of multiple communities as part of the same partition. Analysing the community graphs generated by the algorithm, Figure 8.1 reveals that multiple neighbouring communities are composed of a small number of nodes. Looking at the graph represented in Figure 8.2 it is possible to identify some nodes belonging to smaller communities have strong relationships with nodes from other smaller communities. One could assume that these smaller partitions can be aggregated into a bigger partition in the context of the viral spread, these conclusions should be considered and thoroughly analysed to end with the most optimised method of protecting the communities.

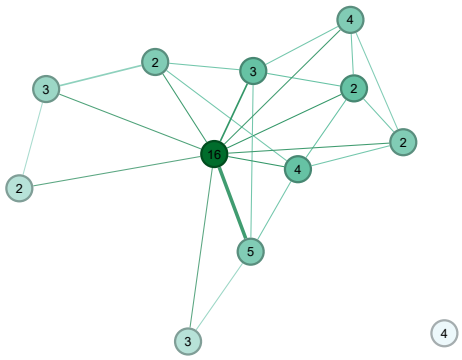

Figure 8.1: Community graph o LFR-50 where nodes are the communities detected and the edges represent the relationships between them, the labels represent the number of nodes in each community

In the *Testing* chapter of this report, we briefly explored the behaviour of the virus when the simulation parameters are changed and how would the response measures affect the spread of the virus. To better

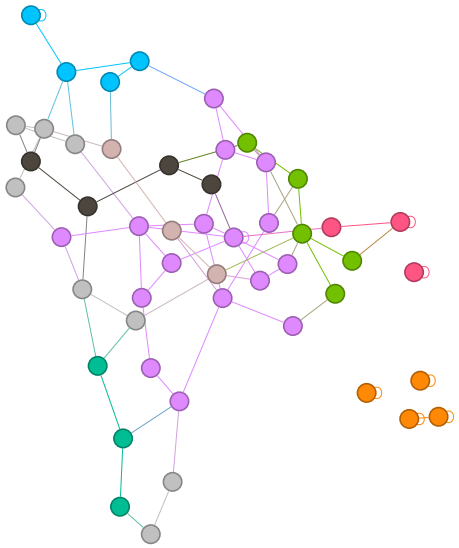

Figure 8.2: LFR-50 Graph where nodes are coloured based on the community they belong

understand the transmission routes and the actual danger the communities are in we need to run a series of similar simulations and compare the results. This way we can better predict the behaviour of the virus and create even better practices to stop the spread of this viral agent. Continuing to analyse the viral spread on the graph LFR-50 we'll run a number of 100 simulations for each scenario we want to test. In this report, we'll explore all four possible scenarios, no interventions, vaccination campaigns only, quarantine orders only and all measures implemented. In table 8.1 we can observe the appropriate metrics after 100 simulations on the graph LFR-50. We observe that every measure implemented is improving the population's response to the spread. Both Vaccination and Quarantining have a positive effect, reducing the number of infected nodes. However, in the case of this graph, quarantine was more effective in reducing the number of exposures and implicitly the number of infections. When both methods are combined, the improvements are not considerably better, noticing an increase in the average infection rate value, indicating that the virus moved more freely when all the measures were in place. We observed the same phenomenon in the individual tests, described in figures 7.1, 7.2,7.3

Calculating the average number of nodes infected in each community over the simulations determines the most affected communities. Using this method is useful when assessing the risk each community is exposed to. In figure 8.3 it is observed that one community is extremely affected by the virus. Community '44' suffered the most number of infections in the majority of scenarios, community 35 being a close second, denoting that these communities are the most vulnerable, in a real-life scenario these communities could be considered hot spots, forcing authorities to resort to the more crude method of containing the virus. On the other hand, it is noticeable that Communities 48, 49, 18, and 24 have very low infection rates and are not as affected as other communities. In all scenarios these communities had the lowest infection rates, indicating that the topology of the networks offers more protection to these communities compared with the rest of

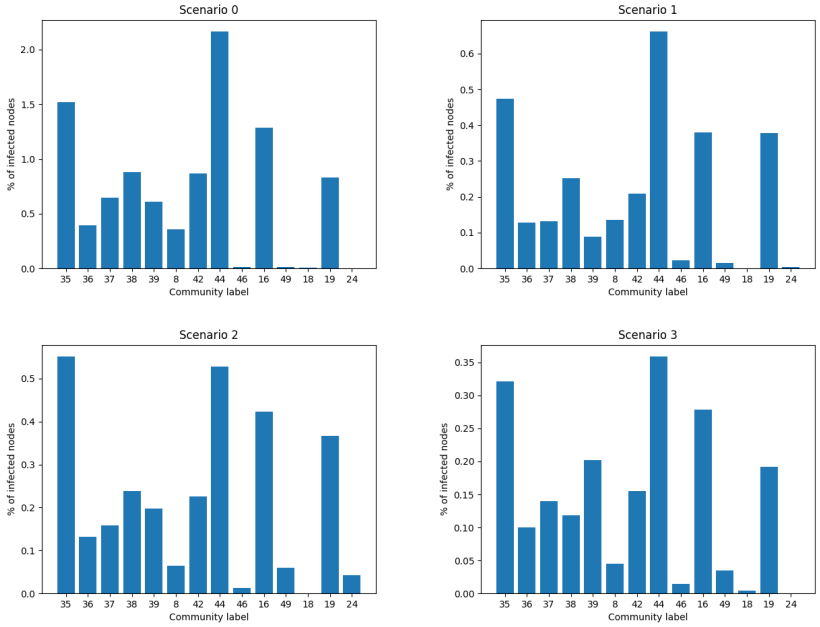

Figure 8.3: Average number of nodes infected in each community, for every scenario.

Test Results				
Scenario	Average Time	Average Inf. rate	Average R0	Super-Spreader
0 - No measures	36.31	4.38	1.73	49
1 - Vaccination	17.62	3.11	0.21	49
2 - Quarantine	14.94	0.57	0.22	49
3 - All Measures	13.44	0.69	0.09	49

Table 8.1: Simulation results on graph LFR-50

them and in the event of a real virus, these communities will not require the same amount of resources and attention, increasing the efficiency of the whole response protocol.

In these conditions, in the event of an epidemic episode, the communities that need the most attention and preparation are communities 35 and 44 as they showed the most worrying results in our simulations. Focusing on reducing the number of exposures in these communities will lead to an overall lower number of recorded infections throughout the network. For this specific network, the most efficient preventive measure is quarantining, preventing nodes to infect others while they have the disease proved to be more effective than making nodes immune to the virus.

8.2 Improvements

While this application satisfies the requirements and can be used to detect the communities of a given network and use that information to run a viral spread recreating possible real-life scenarios, it has some major shortcomings that impede this project to be used as efficiently as possible. First of all, the community detection component of this tool is too slow to be used on large data sets effectively, making it less useful in

the study of viral epidemics in large towns, cities or even counties. The accuracy can also be improved as the current method struggles to find relevant partitions in a large dataset. Having a more accurate community detection method will lead to simulation results of better quality. The SEIR model can also be improved by adjusting the functions that dictate the changes of state to be more precise to what is observed in the data collected during the actual viral spreads, this would lead to more accurate results and effective response measures. Adding more node attributes that are relevant to the viral spread will also increase the accuracy of the model; considering the age, medical condition and other such measures that influence the effect a virus has on an individual is important in determining the risk to which community is exposed. Considering these changes and additions to the application, the final product might be a more effective tool in fighting possible future epidemic episodes or even pandemics.

Chapter 9

Summary

In summary, this report follows the design and development of a viral spread simulation tool that uses a community detection algorithm as a means to improve the findings and development of new methods destined to protect the communities inside the social network the virus acts on. A well-known method of finding communities inside a graph is the Louvain method. This method finds the best partitioning of a graph by shuffling the nodes inside different communities and monitoring the changes in modularity, choosing the partition that yields the highest modularity value. Modularity is a metric used to measure the difference between the number of edges inside a community and the edges outside of the community. Forming communities based on the strength of the relationships between the nodes fit perfectly with the second component of this project the SEIR viral spread model. Using communities that might not be true to the ones formed by the members of the community, but that reflect the behaviours of the individuals and who are they most likely to interact with helps us to better understand how would a virus affect the community as a whole. To make sure the simulation will lead to helpful conclusions, the Susceptible-Exposed-Infected-Recovered model was implemented. Using this model, all the nodes' states are recorded and each change of state is dictated by a predefined function that replicates the possible behaviour of the virus. Using the communities discovered in the first step of this application, assumptions about how would the virus will affect not only the members but also the communities can be made and tested even further. It can now be consider how many nodes of the same group are infected, what communities are at the risk to spread it the most and how long was each community infected. Using these new findings, creating more proactive set of measures to reduce the number of exposed nodes or policies that would keep the number of infected nodes as low as possible, becomes a more informed process. The model also implements two of the most popular response measures in face of a viral threat, quarantining nodes that are infected and vaccinating nodes that are susceptible to the infection. Applying these measures to specific communities or simulating different population responses to these measures can help public health officials to better understand the implications of these measures and take into account multiple scenarios. Using the data generated by these simulations, authorities will have a better understanding of where are their resources most needed and what are the most efficient ways of protecting the communities.

Chapter 10

Conclusion

In this report, the possibility of using community detection as a means to improve the study of a virus infecting a social network was explored. Community detection is a method of identifying groups of nodes in a graph that share similar properties within a larger group of nodes. The Louvain method is a widely used and appreciated community detection algorithm. It partitions the network based on the strength of the connection between nodes, making it a great choice for our purpose. While the Louvain algorithm has its strengths, it is also important to mention its limitations, such as being sensitive to the size of the network and communities within. One of the most important strengths of this method is the usage of modularity, a metric around which is beneficial for us to create communities. This allows us to see how the users interact with each other in their daily activities, leading to more accurate findings about the impact a virus might have on a community. However, this implementation is too slow to be used on networks that represent large social networks. As test results have shown, the algorithm has satisfactory results for rather small networks, but its accuracy drastically drops for more complex networks. These two factors are the biggest drawbacks of this project, reducing its usability in biological research or social studies related to virus activity.

Another component discussed was the SEIR (Susceptible-Exposed-Infected-Recovered) model and how it is used to simulate a virus acting in a community. This model records the number of nodes that are susceptible to the virus, exposed to it, infected by it or completely recovered. Next, it was discussed how the findings of these simulations can help us understand the virus's behaviour in the network and more importantly how each community is affected by this virus.

Using the communities discovered in the first stage of our application, users can discover the groups of nodes that are in the most vulnerable position. Identifying the group of individuals that are at the most risk of infection is an important step in tailoring the preventive measures to the specific needs of the communities. It was observed how simulating the virus multiple times on the same network leads to important conclusions about the network's implicit protection against the virus. This tool can effectively be used to decide what groups of nodes have priority when it comes to protective procedures and spread control protocols.

In conclusion, the usage of community detection in combination with the SEIR model can provide a more comprehensive understanding of the virus spread patterns within a social network and its underlying communities. Identifying high-risk individuals and communities can be efficient in developing policies and restrictions, implementing mask mandates and initiating vaccination campaigns efficiently.

References

Blondel, Vincent D et al. (2008). "Fast unfolding of communities in large networks". In: *Journal of statistical mechanics: theory and experiment* 2008.10, P10008.

Chalseo, KCDC (Jan. 2023). *Case study: KCDC COVID-19 early response*. URL: https://www.kaggle.com/code/chalseo/case-study-kcdc-covid-19-early-response.

Fortunato, Santo (2010). "Community detection in graphs". In: *Physics Reports* 486.3, pp. 75–174. ISSN: 0370-1573. DOI: https://doi.org/10.1016/j.physrep.2009.11.002. URL: https://www.sciencedirect.com/science/article/pii/S0370157309002841.

Girvan, Michelle and Mark EJ Newman (2002). "Community structure in social and biological networks". In: *Proceedings of the national academy of sciences* 99.12, pp. 7821–7826.

Hollocou, Alexandre, Thomas Bonald, and Marc Lelarge (Mar. 2018). "Multiple Local Community Detection". In: *SIGMETRICS Perform. Eval. Rev.* 45.3, pp. 76–83. ISSN: 0163-5999. DOI: 10.1145/3199524.3199537. URL: https://doi.org/10.1145/3199524.3199537.

Lancichinetti, Andrea and Santo Fortunato (Aug. 2009). "Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities". In: *Physical review. E, Statistical, nonlinear, and soft matter physics* 80, p. 016118. DOI: 10.1103/PhysRevE.80.016118.

Lancichinetti, Andrea, Santo Fortunato, and Filippo Radicchi (Oct. 2008). "Benchmark graphs for testing community detection algorithms". In: *Physical Review E* 78.4. DOI: 10.1103/physreve.78.046110. URL: https://doi.org/10.1103%5C%2Fphysreve.78.046110.

Liu, Xin, Hui-Min Cheng, and Zhong-Yuan Zhang (2019). "Evaluation of Community Detection Methods". In: arXiv: 1807.01130 [cs.SI].

Mwalili, Samuel et al. (2020). "SEIR model for COVID-19 dynamics incorporating the environment and social distancing". In: *BMC Research Notes* 13.1, pp. 1–5.

Ningrum, Vanda, Chotib, and Athor Subroto (2022). "Urban Community Resilience Amidst the Spreading of Coronavirus Disease (COVID-19): A Rapid Scoping Review". In: *Sustainability* 14.17. ISSN: 2071-1050. DOI: 10.3390/su141710927. URL: https://www.mdpi.com/2071-1050/14/17/10927.

OECD (2020). "COVID-19 crisis response in ASEAN Member States". In: DOI: https://doi.org/https://doi.org/10.1787/02f828a2-en. URL: https://www.oecd-ilibrary.org/content/paper/02f828a2-en.

Pujol, Josep M., Vijay Erramilli, and Pablo Rodriguez (2009). *Divide and Conquer: Partitioning Online Social Networks*. arXiv: 0905.4918 [cs.NI].

Traag, V. A. (Sept. 2015). "Faster unfolding of communities: Speeding up the Louvain algorithm". In: *Physical Review E* 92.3. DOI: 10.1103/physreve.92.032801. URL: https://doi.org/10.1103%5C%2Fphysreve.92.032801.

Vargas-De-León, Cruz (2011). "On the global stability of SIS, SIR and SIRS epidemic models with standard incidence". In: *Chaos, Solitons & Fractals* 44.12, pp. 1106–1110. ISSN: 0960-0779. DOI: https://doi.org/10.1016/j.chaos.2011.09.002. URL: https://www.sciencedirect.com/science/article/pii/S0960077911001743.

Appendix A

Usage Information

Due to the word limitaion of this report not all simulation parameters were discussed. The report explores only the most important aspects of the simulation, ignoring several attributes that can aid in the study of a virus.

The complete list of all the simulation parameters:

1. G - networkX instance of the graph the virus will act upon
2. partition - a dictionary representing the nodes and their respective community
3. steps - the number of days the simulation will run even if there are still nodes in "EXPOSED" or "INFECTED" state
4. prob_e - probability of exposure given contact
5. prob_i - probability of getting the disease given exposure
6. prob_r - probability to recover from the disease
7. prob_v - probability that a node will accept the vaccine
8. prob_q - probability that a node will respect the quarantine
9. infection_duration - number of days a node has to be infected before it can recover
10. susceptible_again - control variable that allows nodes to become susceptible again after they recovered
11. nat_immunity - number of days a node stays in the recovered state if 'susceptible_again' is active

All these parameters are implemented as a method to tune the simulation in order to bring the simulation closer to life. However, is important to remember that this application represents a model and all the findings and conclusions need to be thoroughly assessed before they can be implemented in real solutions.

Both the simulation and the community detection algorithm can be accessed separately, creating a modulariy aspect that is intended to help the process of improving the overall algorithm. The file 'main.py' is used to drive all the components on a given network. Unfortunately all he save paths and names of files have been hard coded and the process of modifying them is tedious. This application is designed with for users that are accustomed to the python programming language and are comfortable in modifying the files directly.