Disease spreading of epidemics in networks

Author
Argyrios Dimou

Supervisors
Panagiotis Argyракис
Michalis Maragkakis

July 12, 2020
Acknowledgements

I would like to express my very great appreciation and to thank my supervisors Professor P. Argyrakis and Assistant Professor M. Maragkakis for their valuable and constructive suggestions during the planning and development of this research work. I would also like to acknowledge the support provided by the IT Center of the Aristotle University of Thessaloniki (AUTh) throughout the progress of this research work. Finally, I would like to thank my family and friends for their support throughout this project.
# Contents

## 1 Introduction
1.1 Network ......................................................... 9
   1.1.1 Number of Nodes and Degree ................................. 11
1.2 Network Models .................................................. 12
   1.2.1 Erdős Rényi ................................................ 12
   1.2.2 Small world ................................................. 13
   1.2.3 Barabási–Albert model ...................................... 14
1.3 Multilayer ....................................................... 16
1.4 Multiplex ......................................................... 17
   1.4.1 Without interlinks ......................................... 17
   1.4.2 With interlinks ............................................. 18
   1.4.3 Other types of networks ................................... 19
1.5 Epidemics ........................................................ 20
   1.5.1 Deterministic SIR ........................................... 20
   1.5.2 SIR on networks ............................................. 21
   1.5.3 SIRS ...................................................... 23

## 2 Modelling SIRS
2.1 SI .................................................................. 25
2.2 SIR ................................................................ 27
2.3 SIRS .............................................................. 28
2.4 Application to a larger Network ............................... 30
2.5 SIRS on a Watts Strogatz network ......................... 31
   2.5.1 Variation of parameter \( \tau \) ................................. 32
   2.5.2 Variation of parameter \( t_I \) ................................. 33
   2.5.3 Variation of parameter \( t_R \) ................................. 34
2.6 SIRS on a multiplex network of 2 Watts Strogatz networks ............................................. 35
   2.6.1 Variation of parameter \( t_{I_2} \) ....................... 36
   2.6.2 Variation of parameter \( t_{R_2} \) ....................... 39
   2.6.3 Variation of parameter \( \tau \) .............................. 41
   2.6.4 Variation of parameter \( \tau' \) ...................... 43

## 3 Community detection
3.1 Louvain Algorithm ............................................. 46
3.2 Modelling SIR on communities .............................. 47
3.3 1st model ....................................................... 47
### CONTENTS

3.3.1 Variation of parameter initial $R_0$ ........................................ 48  
3.3.2 Variation of parameter $P_{q_{rn}}$ ........................................ 53  
3.4 2nd model ............................................................... 57  
3.4.1 Variation of parameter initial $R_0$ ........................................ 57  
3.4.2 Variation of parameter $R_{q_{rn}}$ ........................................ 62  
3.5 3rd model ............................................................... 67  
3.5.1 $P_{q_{rn}} = 0.05$ ......................................................... 68  
3.5.2 $P_{q_{rn}} = 0.1$ ......................................................... 70  

Appendices ......................................................... 73  
A Source code of 2.6 .................................................. 75  
B Source code of 3.3 .................................................. 78  
C Source code of 3.4 .................................................. 81  
D Source code of 3.5 .................................................. 84
Abstract

In our thesis we study models which simulate infection spreading in social networks, where the nodes represent people and edges their social contacts. Our research is constructed of two parts, in each one of those we create models that fit to different types of infections and we study different topics. In the first part we assume that a node which has recovered from an infection, could be infected again, as after a certain amount of time it will be susceptible to the disease again. The topic that we study in this part is whether or not we could observe sustained oscillations of the number of infected people in a social network, i.e. if the infection remains in the network for a long period of time. In order to do that, initially we assume that only one node in the whole network is infected and observe how the infection spreads in a Watts Strogatz network. By adjusting the parameters of our model, we could sometimes observe sustained oscillations, and sometimes not. An interesting question is if we would observe sustained oscillations, by keeping constant the parameters of the disease spreading model in a single network in the case that we did not observe sustained oscillations and connect this network with another Watts Strogatz network. What should the parameters of the disease spreading model in the second network be? In the second part of our thesis, we assume that as a result of the nature of the studied infections, the recovered nodes have permanent immunity from the infection, thus the nodes can be infected only once. Considering that lockdown metres are a realistic possibility, we simulate those metres in our models. In order to create such a model, initially we divide the network to communities, i.e. smaller groups of nodes which interact mainly with other nodes inside the same community. By dividing the network to communities we can apply metres to any group separately and we can create models which use different criteria for the insertion of a community into quarantine. It is also important to point out that in these models are applied different methods to simulate lockdown metres. Our main concern will be how basic properties of the disease spreading model in a Watts Strogatz network like number the total different nodes that were infected during one simulation will be diversified by the changing parameters of our models such as the criterion that a community becomes quarantined. Finally, by observing the graphs of these properties produced by different models, we compare the models.
Chapter 1

Introduction

1.1 Network

In this thesis, the most fundamental structure where we apply all our models to, are networks. Networks are defined as graphs which represent a system of parts that interact in pairs. A network graph is symbolized $G(V, E)$ where $V$ are called vertices or nodes and symbolize the parts of the network and $E$ are called edges or links and symbolize the interaction between the nodes. The information that networks provide will be understood if we mention network examples, so in the next lines we will try to explore different network categories.

![Figure 1.1: Nodes and links of a network](image)

- **Technological networks**
  Networks like train or airplane routes, where the nodes are the geographical locations where a route ends, and the edges are the links between them.

- **Information networks**
  Examples of this category of networks are networks like the world wide web, where the nodes are the web pages and the edges are the highlighted text links that redirect the user from one web page to another. Another network example is a citation network which is very common in scientific articles. In this type of network a writer cites an article which he has taken in consideration during his research, so the reader can study the cited article too if he wants more details.

- **Biological Networks** Networks like neural networks where the nodes are neurons of the brain.
and the links between them are the synapses which transfer the information from one neuron to an interconnected one. Another example of biological networks are ecological networks, where the nodes are usually animals and one link between a pair of them means that one animal eats the other.

• Social Networks The networks that our research is focused on belong in this category. The nodes of these networks are people and the links between them symbolize any kind of relationship (friendship, family, collaboration) between them. Thus, one could understand that the link between a pair of people could mean a contact between them, which is the basic idea of the epidemic models.

We notice that the type of the system we want to describe and the category that the network belongs, define what the nodes and their interaction will be. However even though a simple network like figure 1.2 is a valid representation of the system, many details are not illustrated, like how strong is an interaction between the nodes, considering that in many networks such as a social network with edges which represent friendship, a pair of nodes could be close or less close friends.

![Random network](image)

Figure 1.2: Random network

In order to illustrate how strong is an interaction, a network could be weighted, i.e. a number could describe the strength of an edge. In a graph an edge with bigger strength could be represented with a thicker line or different colour (darker for example) than an edge with less strength (figure 1.3).
1.1. NETWORK

Figure 1.3: Random weighted network with $N = 15$ nodes, where weights are measured with double numbers from 0 to 1. If a weight is less than 0.5 the edge is represented with a dashed thin line, else it is represented with a thick black line.

In the above networks (figure 1.2, figure 1.3) if a node i is linked to node j means that j is also connected to node i. These kind of networks are called undirected and even though our model is based only in these types of networks, we have to state that there are also types of networks where the fact that a node i is linked to node j does not mean necessarily that node j is also linked to node i. An example of this category of networks is the world wide web, where despite that a link can transfer us from one internet page(i) to another(j), it is not certain that in the second page(j) will be another link that will transfer us again to the first one(i). These kind of networks are named directed.

1.1.1 Number of Nodes and Degree

Since we have decided that the networks in our algorithm will be undirected and unweighted we continue our thesis by defining quantities important to our calculations. First of all in order to observe which are the connected nodes we have to store a $N \times N$ matrix $a_{ij}$, the adjacency matrix, where N are the total number of nodes. In this matrix i and j will represent a pair of nodes and if a link between those nodes exists, $a_{ij}$ will be equal to 1, else will be 0.

$$a_{ij} = \begin{cases} 1 & \text{if node i is linked to node j} \\ 0 & \text{else} \end{cases}$$

Since the network is undirected if i is linked to j i.e. $a_{ij} = 1$, j will be connected to i and $a_{ji} = 1$ too. So, the total number of links $L$ of our network will be equal to:

$$L = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} a_{ij}$$

Another characteristic of a network is the degree $k$ of its nodes. According to our experience the nodes of the networks that were described in the previous section do not have the same number of connections. For example, in a social network of people which shows the friendship ties in a
community, not all people have the same number of friendships. The degree \( k \) of each node \( i \) is given in terms of the adjacency matrix as

\[
k_i = \sum_{j=1}^{N} a_{ij}
\]

If \( k_i \) of every node is known we can calculate the mean degree \( <k> \) a quantity that characterizes the network as a whole.

\[
<k> = \frac{1}{N} \sum_{i=1}^{N} k_i = \frac{2L}{N}
\]

### 1.2 Network Models

Properties like degree distribution, which define the model of the network affect its structure. In the next chapter we will examine different network models.

#### 1.2.1 Erdős Rényi

The simplest network model is the random graph. In this model the probability of a link being created between a pair of nodes is the same for all the possible pairs. This criterion is satisfied by setting fixed either the number of edges or the probability of an edge. In the first case the graph is symbolized as \( G(N, L) \) where \( N \) are the number of nodes, \( L \) the number of edges, which are chosen randomly between all possible \( \binom{N}{2} \) node pairs. Therefore, there is not just one network that satisfies this criteria.

Correspondingly another format of a random graph is \( G(N, p) \) where \( p \) is the fixed probability of an edge creation for every possible edge. This model is associated to Paul Erdős and Alfréd Rényi who described the model in a series of papers, and will be the one that we will use whenever we include an Erdős Rényi network in our research.

The degree distribution of this model is a binomial distribution, given by

\[
P(k) = \binom{N-1}{k} p^k (1-p)^{N-1-k}
\]
1.2. NETWORK MODELS

1.2.2 Small world

The Watts Strogatz small world model was introduced by Duncan J. Watts and Steven Strogatz and its purpose was to confirm that the small world effect, i.e. the existence of short paths between the nodes, is prevalent in any network kind. In order to describe this model, one starts with a 1D lattice with boundary conditions, where every node is connected with $k$ nodes near it. Then, some of the edges are rewired with probability $p$ to new random positions, resulting in change of the network.

- If the rewiring probability is 0, then no edges are rewired, thus the network remains as it is. (figure 1.5.a))

- If the rewiring probability lies between 0 and 1 then its value will determine the amount of edges that will change. (figure 1.5.b)

- If the rewiring probability is 1, then all the edges are rewired, so the network structure is the same as we would have if the model was the random graph. (figure 1.5.c)

It has been proven that even if we rewire a small percentage of the total edges of the network then in this model, any 2 nodes of the network are only few paths from each other, considering the shortest path.
1.2.3 Barabási–Albert model

One of the most fundamental and important algorithms of generating networks is the Barabási–Albert model. If we consider that there are many categories of networks that increase their nodes over time, such as citation, social networks or the Internet it would be important to explore the mechanism that a node which is being inserted in the network, creates edges with the nodes that were already in the network.

Albert-László Barabási and Réka Albert created a model which explains this mechanism by suggesting the following steps. Initially we have a network with N nodes. Afterwards the network will be created based on the ideas of growth and preferential attachment.

- **Growth.** Each time a node is added to the network, it will create m edges with m different nodes of the network. Thus it is important that the N initial nodes of the network at time \( t = 0 \), are more or equal to m, so that at \( t = 1 \) there are at least m available nodes.
• **Preferential attachment.** The m edges that each node will have, are not random. Nodes tend to prefer to connect to nodes that already have many edges, the hubs. For instance in a social network, new members of a social network tend to create friendships with people that already have many contacts, who are more possible to be approached than people with less contacts. The probability $p_i$ that a new node connects to a node $i$ that already existed in the network is equal to

$$p_i = \frac{k_i}{\sum_j k_j}$$

where $k_i$ is the degree of node $i$ and $\sum_j k_j$ is a sum to the degree of every node that is not connected to the new node.

Figure 1.6: Network produced with the Barabási–Albert model. Nodes $N = 20$ with a) $m = 1$, b) $m = 2$, c) $m = 3$
1.3 Multilayer

Even though the theory of single networks simulates so many systems is not enough to describe complex systems, since most of the times in nature we don’t observe single networks in isolation, but networks that interact with each other. The systems we study, are usually divided into subsystems, which we call layers and then we observe the way these heterogeneous parts interact.

This idea was first introduced in social networks, a great example of which could be 2 separate networks of people of the same geographical location, with the links between them meaning a social tie (friendship, work relationship, family tie) and a link between 2 nodes that belong in different networks correspondingly is translated to a social tie between 2 people from different geographical locations. Other typical examples of multilayer networks are ecological networks, transportation networks where the nodes of each layer could represent the terminals of a route and the interactions between 2 nodes from different networks could possibly mean that they belong in the same geographical location.

We can define a multilayer network \( M \) by the triple

\[
M = (Y, \vec{G}, \mathcal{G})
\]

where

\[
Y = \{a | a \in \{1, 2, ..., M\}\}
\]

are the number of layers and \( M \) the last layer,

\[
\vec{G} = (G_1, G_2, ..., G_M)
\]

the networks of each layer in our multilayer network, where

\[
G_l = (V_l, E_l)
\]

\( V_l \) and \( E_l \) are the nodes and the internal edges inside the layer. Finally,

\[
\mathcal{G}_{l_1, l_2} = (V_{l_1}, V_{l_2}, E_{l_1, l_2})
\]

the network that forms with the nodes of the layers \( l_1 \) and \( l_2 \) and edges the links between them.

If we focus on the epidemic model we study in our thesis, we conclude that the interdependencies between networks seriously affect the properties of our system. To be more specific, the connections and contacts that people from other communities have could lead to faster spread of a disease, or could lead to the extension of the time that an infection survives in our system. Thus, it is important to know the way that these interdependencies form and to divide multilayer networks to categories.
1.4 Multiplex

Multiplex is a special category of multilayer networks, which is characterized by one to one connections between the nodes of the two networks. The linked nodes belong in different layers but have the same label (i.e. a link is formed between node 1 in layer 1 and node 1 in layer 2). These type of nodes are called replica nodes. There are two ways to illustrate a multiplex network.

1.4.1 Without interlinks

This is a common way to depict the multiplex network. In this case the replica nodes are not connected with interlinks and it is implied that the set of nodes are the same in every layer but the intralinks between the nodes of each layer are different. A network of this type could be a network with nodes the same set of people and the edges between them could illustrate the social tie that each layer represents (for example layer 1 could represent the family ties, layer 2 the friendships and so on). If we consider the way we defined networks, a network of this type would be defined

\[ M = (Y, \vec{G}) \]

where,

\[ \vec{G} = (G_1, G_2, \ldots, G_M) \]

each layer has the same nodes:

\[ V = \{i | i \in \{1, 2, \ldots, N\}\} \]

each layer l of M total layers has an adjacency list \( a^{[l]} \) as defined at the previous section

\[ a^{[l]}_{ij} = \begin{cases} 1 & \text{if node i is linked to node j} \\ 0 & \text{else} \end{cases} \]

Figure 1.8: Multiplex network of 3 layers, with \( N = 10 \) nodes in each layer, without interlinks
1.4.2 With interlinks

In this case the replica nodes appear to be connected with interlinks, and even though it seems that it is just an alternative way of to illustrate the multiplex network, there could be a substantial difference between the no interlinks and with interlinks cases. Having interlinks could allow us to distinguish the identity of the nodes, thus nodes from different layers could represent something different. For example a social network with each layer represents different people and their social ties. Each person will have social ties with people that represent nodes in his layer, and with the person that is being represented by the replica node.

Given the definition of the multilayer networks, a network of this type is defined by the triples \( M = (Y, G, \mathcal{I}) \). Just like the case that we don’t have interlinks

\[
\bar{G} = (G_1, G_2, ..., G_M)
\]

are the ordered \( G_l = (V_l, E_l) \) networks of each layer, with

\[
V_l = \{i | i \in \{1, 2, ..., N\}\}
\]

the nodes and \( E_l \) the intralinks of each layer. \( \mathcal{I}_{l_1,l_2} = (V_{l_1}, V_{l_2}, E_{l_1,l_2}) \) are the networks that are formed with nodes the nodes of any of the layers \( l_1 \) and \( l_2 \) of the network and edges the edges between them, which will probably be between the replica nodes. The adjacency lists will have a different format than before.

\[
a_{ij}^{[l_1,l_2]} = \begin{cases} 
1 & \text{if node i is linked to node j} \\
0 & \text{else}
\end{cases}
\]

are the elements of an \( N \times N \times M \times M \) supra-adjacency matrix \( A \) which defines the edges of the network. As it is obvious, when \( l_1 = l_2 \) the elements refer to the intralinks, while when \( l_1 \neq l_2 \) \( a_{ij}^{[l_1,l_2]} \), represents the interlinks between \( l_1 \) and \( l_2 \). Thus if \( l_1 \neq l_2 \), \( a_{ij}^{[l_1,l_2]} \) will be equal to 1 if \( i = j \) or else equal to 0.

![Multiplex network of 3 layers, with \( N = 10 \) nodes in each layer, with interlinks](image-url)
1.4.3 Other types of networks

In a realistic example of a social interconnected network where the layers represent the people of different towns it would be hard to confine the interlinks just to the replica nodes, because as we observe in reality, people have a certain number of social connections in different towns, hence, the topology of this network would be more complicated than a multiplex network. Thus, there are interconnected networks which have a more sparse distribution of their interlinks than the distribution of multiplex networks, so their supra-adjacency list will transpose to

$$A = \begin{pmatrix}
a^{[1,1]} & a^{[1,2]} & \ldots & a^{[1,3]} \\
a^{[2,1]} & a^{[2,2]} & \ldots & a^{[2,3]} \\
a^{[3,1]} & a^{[3,2]} & \ldots & a^{[3,3]}
\end{pmatrix}$$

As it was mentioned earlier, the number i of each element $a^{[l_1,l_2]}_{ij}$ represents the number of the node of layer $l_1$ and correspondingly j symbolizes the number of the node of layer $l_2$. When the layers $l_1$ and $l_2$ are the same then the element $a^{[l_1,l_2]}_{ij}$ represents the intralinks of the layers, while when $l_1 \neq l_2$, $a^{[l_1,l_2]}_{ij}$ represents the interlinks between $l_1$ and $l_2$.

Figure 1.10: A multilayer network of 3 layers, with $N = 10$ nodes in each layer
1.5 Epidemics

1.5.1 Deterministic SIR

Due to the fact that the fear of an epidemic or even a pandemic is realistic, which is proven by the spread of the virus Covid - 19, epidemics on networks is a well known subject that has been studied thoroughly. Diseases that can be transmitted sexually like HIV, by the air like Influenza, by touch like staphylococcus or diseases that spread due to the interaction of two people, can be modelled in networks with the people being the nodes and the interaction between them being the edges. The simplest of these models is SIR, an epidemic model which suggests that each node of the network belongs to specific states, that describe the node, and the way that the states of these nodes are distributed in the network will have a big impact on the evolution of our model. These states are

- \( S \) (Susceptible). A healthy node which can get infected.
- \( I \) (Infected). An infected node which can spread the disease and infect other susceptible nodes.
- \( R \) (Recovered). A node that is immune to the disease and can not get infected no matter how many infected nodes it contacts.

Each time a susceptible node contacts an infected one, there exists a probability that the infection will be transmitted from the first node to the second one. If we examine this event by the biological side, when two people, one infected and one susceptible contact each other and the susceptible person contracts the disease, a process begins in the organism of the recently infected person. His white blood cells are trying to fight the disease and if they do eliminate the infection the person recovers, otherwise the person dies or becomes permanently ill. In our model we do not take in mind biological details, but we just examine the result of this procedure, i.e. what is the state of the person.

If we were studying the traditional SIR, we would assume that in a group of \( N \) people each interaction would be random. Every time that an infected individual contacted a susceptible one, the first could pass the infection to the second one with one probability which would depend on how contagious the infection is. The numbers of contacts that an I individual could infect on average, is defined \( \beta \), a really important quantity for the model. After an individual is infected, he will either recover from the disease or die, after constant time \( t_I \). For the SIR model a dead node or a node that has permanent immunity from the infection are considered the same cases since there will be no change of their state, they will remain \( R \). Thus, they have no interest for the model anymore.

As we mentioned above, for each infected individual at a certain time step there are on average \( \beta \) contacts that could lead to infection. However not all of these contacts are with susceptible individuals. Considering the total number of people in the group is \( N \), the probability of a person
being susceptible is $\frac{S}{N}$. Thus, for each infected person $\beta \cdot \frac{S}{N}$ contacts would lead to infection. If we multiply this number with $I$ we would compute the number of people that would transpose from $S$ to $I$ in a time step. Additionally, defining $\gamma$ as the rate of people that at a time step become recovered from being infected, the number of people that would recover at a time step would be equal to $\gamma \cdot I$. In conclusion the differential equations of our model are:

$$\frac{dI}{dt} = \beta \cdot \frac{S}{N} \cdot I - \gamma \cdot I$$
$$\frac{dS}{dt} = -\beta \cdot \frac{S}{N} \cdot I$$
$$\frac{dR}{dt} = \gamma \cdot I$$

Considering that we have a group of $N = 10000$ people where $\beta = 1$ and $\gamma = 1/10$, we suppose that in the beginning of time only one of the people is infected and the rest of them are susceptible. If we solve the above written equations numerically, the result would be the following graph.

Figure 1.11: Deterministic SIR applied to a group of 10000 people. $\beta = 1$, $\gamma = 1/10$. Susceptible nodes versus time(green line), Infected nodes versus time(red line), Recovered nodes versus time(blue line). Initial $I = 1$ and $S = 9999$

1.5.2 SIR on networks

An alternative way to apply the SIR model, instead of the deterministic SIR, mentioned in the previous chapter, would be SIR on networks. In this approach we assume that in a network the nodes represent people or animals, and the edges between them symbolize that they contact each other. Considering the fact that in the deterministic model every node interacts with all the other ones, and the contacts that one infected will transmit the disease to, will be random, we could assume that SIR on networks is a more realistic approach, since as we observe in reality the contacts between an infected and other members of society are not random. He will have links that are determined by the place that he lives, his work and his social activities. These links are depicted by a social network.

On networks instead of using the differential equations in each time step, we focus on each node and its interactions with every neighbour node of it, i.e. every node that it is connected with. The basic quantity here, the transmissibility $\tau$ defined as the probability that one infected can infect
a susceptible node, consequently \( \tau = \frac{\beta}{\langle k \rangle} \) where \( \langle k \rangle \) is the average degree of the network. Most of the times the available data are concerned with another quantity with vast impact, \( R_0 \). Hence we compute \( \tau \) indirectly. We define \( R_0 \) as

\[
R_0 = \frac{\beta}{\gamma} \Rightarrow R_0 = \beta \cdot t_I
\]

\( R_0 \) represents the contacts that on average an infected node would infect if all of his neighbours were susceptible. Thus it is logical to assume that it is a quantity that indicates the near future of the spread of the infection. If \( R_0 \) is large the infection will keep spreading, on the contrary the infection will "die" soon if \( R_0 \) is small.

Applying SIR on a Watts Strogatz network with average degree \( \langle k \rangle = 7 \) and \( N = 10000 \) with \( \beta = 1 \) and \( \gamma = 1/10 \) (\( t_I = 10 \))

![Graph](image)

Figure 1.12: Infected versus time(red line), Susceptible versus time(green line), Recovered versus time(blue line) of an application of SIR model on a Watts Strogatz network with \( \langle k \rangle = 7 \), \( N = 10000 \), rewiring probability \( p = 0.1 \), \( \beta = 1 \), \( t_I = 10 \). Initial \( I = 1 \) and \( S = 9999 \). The results are averages of 50 simulations

As we expected, we can notice the difference between the two graphs(Figure 1.11 and Figure 1.12). As we explained above, the fact that it is possible for a node to infect only its neighbour nodes and not all the nodes in the network makes the result different. Moreover Figure 1.12 is strongly dependent on the average degree of the network. The spread and consequently the number of infected nodes each time step would be smaller with a smaller average degree and bigger with a bigger average degree.
1.5.3 SIRS

The model we described in the previous chapter, SIR, is one of the simplest forms of an epidemic model. In this model, the R nodes that have recovered are permanently immune to the virus. However, not all infections fit this model as it is a common knowledge that there are infections where a recovered person could eventually be infected again. In these cases, the suitable model is SIRS. In this type of model, the first steps are the same as SIR, considering that a susceptible node can be infected with a certain probability when it interacts with an infected node. Afterwards, the infected node will become recovered at a certain rate or after a defined time. The difference between SIR and SIRS is that a recovered node can be susceptible again after a defined time, and then the initial steps will be repeated until the time we want the model to stop, or when there are no infected nodes left in our network.

A Susceptible node (S) could become infected (I) then recovered (R), and then again susceptible (S). As we understand, while in the SIR model the infected nodes will either recover or die, in the SIRS model we will have oscillations of the number of infected nodes and sometimes we could have sustained oscillations, meaning that the virus will stay in the network for a lot of time. In the next figures we will apply SIR and SIRS in a network with the same parameters.

![Figure 1.13: Infected(t) for a)SIR and b)SIRS models on a Watts Strogatz network with $N=10000$, rewiring probability $p=0.1$, average $k < k >= 10$. The SIR parameters are $\beta = 0.25$, $t_I = 10$ which are the same in the SIRS model too. In SIRS $t_R = 50$. The results are averages of 50 simulations.](image)

It is important to notice that instead of taking the standard SIRS approach where it is assumed that rates per unit time of converting infected nodes to recovered and recovered nodes to susceptible again are constant, we took the fixed time approach. In this approach each disease has a fixed time $t_I$ where an infected node stays infected before becoming recovered, and a fixed time $t_R$ where the node stays recovered before becoming infected. According to Anderson and May it is a pretty valid simplification and for many infections it applies better than a constant rate.

It would be interesting to apply these methods to interconnected networks, because even in cases that an infection is eliminated in a single network there is a probability of surviving in a multilayer network, as the interlinks can help in the spread of the disease. Additionally, it is realistic...
too. Unless governments take steps to isolate people, networks that represent the contacts of people in a region have interlinks with people in other locations with result to spread the infection.
Chapter 2

Modelling SIRS

2.1 SI

To begin modelling SIRS, we will start by the most fundamental interaction in our model, an interaction between a susceptible and an infected node. As we have mentioned earlier, in this type of interaction there is a probability $\tau$, the transmissibility, that the infected node will spread the infection to the susceptible one. The colours that will represent the states of the nodes will be green for susceptible, red for infected and blue for recovered nodes.

As it is easy to understand, after the interaction there are two possible outcomes which are determined by the value of $\tau$. The infection will be either spread or not.

![Figure 2.1: After the interaction of a S and a I node a)S node was not infected b)S was infected](image)

We consider a small Erdős–Rényi network that consists of only 4 nodes, thus as we have explained in the introduction the total number of the possible edges is equal to $\binom{N}{2} = 6$ edges. We assume that the probability that a possible edge exists is 0.8, so the expected number of edges in our network is around 5. After the creation of the network at time step $t = 0$, where $t$ is the evolution time of our model, we consider one random node of the network infected, and with transmissibility equal to $t = 0.4$ we observe the evolution of our system.
Figure 2.2: Erdős–Rényi network $N = 4$, $p = 0.8$ the probability of a link creation. The network consists of susceptible(green) or infected(red) nodes which interact between them with $\tau = 0.4$. The network is depicted for specific time steps. a) $t = 0$ b) $t = 2$ c) $t = 4$

In this simulation, we observe a network with 6 edges, more than we expected. Each node is labelled so that we can pay attention on the change of the state of it. As we notice, at $t = 0$ the only infected node is D while the rest of them are susceptible. The node D has three links, with nodes B, C and A. Each interaction of them has probability 40% to infect the susceptible(green) node. The outcome is that after one time step none of those three interactions spread the infection, as nodes B, C and A remained susceptible.

However at time step $t = 2$ the edge (D,B) infected B. At $t = 4$ edges (B,A) or (D,A) and (D,C) or (B,C) lead to the infection of A and C respectively. We have came to this conclusion as we have assumed that infected nodes do not infect the susceptible ones in the same time step of their infection. Time step $t = 4$ is when all the nodes of the network are infected.
2.2 SIR

The next step in our model is to consider that every time a node is infected, we should define the time that the infected node will recover. Assuming that as we have mentioned earlier \( t \) is the variable where we store the evolution time of our model, when a node is infected, we define the time step \( t_{R_{\text{node}}} \) that it will recover as,

\[
t_{R_{\text{node}}} = t + t_I
\]

, where is \( t_I \) the constant value we have defined, which represents the time steps that an infected node will remain infected. When \( t \) is equal to \( t_{R_{\text{node}}} \), the node has recovered. We assume that \( t_I = 5 \) and let our model evolve.

Figure 2.3: SIR model with \( \tau = 0.4 \) and \( t_I = 5 \) applied on a Erdős–Rényi network with \( N = 4 \), and probability of a link creation \( p = 0.8 \). The network is depicted at specific timesteps and each colour of a node represents a state. S(green), I(red), R(blue). a)\( t = 5 \) b)\( t = 7 \) c)\( t = 9 \)

The first node that will be recovered will be D as it was expected, since it was the first that was infected. At \( t = 5 \), 5 time steps from \( t = 0 \) the moment that D was infected, it transposes to R.
At \( t = 7 \) node B which was infected at \( t = 2 \) and at \( t = 9 \) nodes A and C which were infected at \( t = 4 \) recover, all three of them 5 time steps after their infection as we defined. If we were focusing in a SIR model, this time step would be the last time step of the simulation as no infected node remained in the network, so there could be no further spread of the infection.

2.3 SIRS

The next and final step to complete the model, is to define the time that each node will change state from recovered to susceptible. Every time that a node becomes recovered we define the time that it will become susceptible again as

\[
t_{S_{\text{node}}} = t + t_R
\]

Consequently, every time \( t \) is equal to \( t_{S_{\text{node}}} \) the node is susceptible again. In this particular simulation we assume that \( t_R = 8 \).

![SIRS model](image)

Figure 2.4: SIRS model with \( \tau = 0.4, t_I = 5, t_R = 8 \), applied on a Erdős–Rényi network with \( N = 4 \), and probability of a link creation \( p = 0.8 \). The network is depicted at specific timesteps and each colour of a node represents a state. S(green), I(red), R(blue) a)\( t = 13 \) b)\( t = 15 \) c)\( t = 17 \)
The first time that a node transposes from recovered to susceptible is $t = 13$, the node D, exactly 8 time steps after the moment that it was infected.

At $t = 15$ node B and $t = 17$ nodes A and C become susceptible $t_R$ time steps after their infection. It is important to notice that this simulation was stochastic. If we would apply this model again, it would be very likely that the outcome would be different, as we could have different edges in our network and different infected nodes. As bigger the network is, the more the combinations of the possible states of the nodes. In conclusion, in order to assume that a result is trustworthy, a big number of simulations is required.
2.4 Application to a larger Network

It would be interesting to apply the SIRS model in a larger network, in order to observe the transition between the states of the nodes. This larger network will be an Erdős–Rényi network with $N = 500$ and $p = 0.05$, parameters which lead to average $k$ equal to $< k > = 24.95$. The parameters of the model are

$$
\tau = 0.4, \quad t_I = 5, \quad t_R = 8
$$

Figure 2.5: SIRS model applied on a Erdős–Rényi network with $N = 500$, and probability of a link creation $p = 0.05$. $\tau = 0.4$, $t_I = 5$, $t_R = 8$. The network is depicted at specific timesteps and each colour of a node represents a state. S(green), I(red), R(blue) a) $t = 0$ b) $t = 1$ c) $t = 7$ d) $t = 15$

We assume that at $t = 0$ one of the nodes will be infected. At the next time step, $t = 1$ we notice that the initial I node infected 16 S nodes. It is not unnatural that this number is higher than the number we would expect if we consider the $< k >$, since there could be nodes that have more connections than $< k >$. One of them was the initial node, as it had $k = 33$ edges.

At $t = 7$ there are a lot of blue (recovered) nodes in our network. These nodes are the ones that were infected until $t = 2$. 
2.5 SIRS ON A WATTS STROGATZ NETWORK

At $t = 15$ the network consists of only blue (recovered) and green (susceptible) nodes. The susceptible ones are those that were recovered more or equal time than $t_R$ time steps before, while it has passed less than $t_R$ from the time that the blue ones recovered. Eventually all the nodes of the network will be susceptible, however they will remain susceptible as there will be no infected node.

2.5 SIRS on a Watts Strogatz network

It is a common truth that the realistic social networks have a lot more nodes than the ones we have presented so far. Thus, the next section will be an application of SIRS on a Watts Strogatz network with $N = 10000$ nodes. The network will have rewiring probability, $p = 0.1$ and average $k, <k> = 14$. Below we present the number of infected nodes each time step of a simulation for different values of the SIRS parameters $\tau$, $t_I$ and $t_R$. The results are averages of 200 simulations.

![Figure 2.6](image)

Figure 2.6: Infected versus time after the application of SIRS model to a Watts Strogatz network with $N = 10000$, $<k> = 14$, $p = 0.1$. The SIRS parameters are $\tau = 0.022$, $t_I = 8$, $t_R = 50$. The results are averages of 200 simulations.

As we notice in the above plot, we can observe oscillations that do not fade out quickly. In the next section our study will be focused on observing how the change of the SIRS parameters of the above plot will affect the evolution of the number of infected nodes. Hence, we will keep constant all the SIRS parameters of figure 2.6 except for one, which will be the title of each subsection.
2.5.1 Variation of parameter $\tau$

Figure 2.7: Infected nodes versus time after the application of SIRS model on a Watts Strogatz network with $N = 10000$, $< k > = 14$, $p = 0.1$. The SIRS parameters are $\tau = 0.017$, $t_I = 8$, $t_R = 50$. The results are averages of 200 simulations.

The first variable we will change is $\tau$. With $\tau = 0.017$, we notice that even though the oscillation do not fade away, we have smaller numbers of infected nodes, as the infection is not contagious enough to be spread into a bigger fraction of the network.

Figure 2.8: Infected nodes versus time after the application of SIRS model on a Watts Strogatz network with $N = 10000$, $< k > = 14$, $p = 0.1$. The SIRS parameters are $\tau = 0.027$, $t_I = 8$, $t_R = 50$. The results are averages of 200 simulations.

On the figure 2.8 we observe that the oscillations of the number of infected nodes have smaller amplitude as the time increases. Even though we would expect an increment in the number of
infected by increasing $\tau$, this is not the case in this situation. In this example, as we can notice by the large first peak in the figure 2.8, a big fraction of the total nodes of the network become infected in a short amount of time. Consequently, these nodes will become recovered and susceptible at time steps that will have small difference. By having a large fraction of the network in the S state at about the same time steps, make the spread of the infection harder.

2.5.2 Variation of parameter $t_I$

Figure 2.9: Infected nodes versus time after the application of SIRS model on a Watts Strogatz network with $N = 10000$, $< k >= 14$, $p = 0.1$ The SIRS parameteres are $\tau = 0.022$, $t_I = 5$, $t_R = 50$. The results are averages of 200 simulations

The decrease of $t_I$ as we would expect leads to significantly lower peaks of infected nodes than figure 2.6. Moreover due to the fact that the infected nodes recover quickly and become susceptible again, the time that an infected node becomes susceptible is lower that in figure 2.6. Consequently the concentration of susceptible nodes before the number of infected nodes start to approach zero, does not allow the oscillations to fade out.
By increasing $t_I$, the outcome is that the infected nodes have more opportunities to infect their neighbours. Thus, a bigger fraction of the total nodes of the network become infected quicker than in figure 2.6 and the peaks become higher.

### 2.5.3 Variation of parameter $t_R$

---

Figure 2.10: Infected nodes versus time after the application of SIRS model on a Watts Strogatz network with $N = 10000$, $< k > = 14$, $p = 0.1$. The SIRS parameters are $\tau = 0.022$, $t_I = 14$, $t_R = 50$. The results are averages of 200 simulations.

Figure 2.11: Infected nodes versus time after the application of SIRS model on a Watts Strogatz network with $N = 10000$, $< k > = 14$, $p = 0.1$. The SIRS parameters are $\tau = 0.022$, $t_I = 8$, $t_R = 30$. The results are averages of 200 simulations.
Decreasing $t_R$ we observe that the number of infected reach a plateau, since the rate of nodes becoming susceptible is high. Consequently, considering that constantly nodes become S, the I nodes infect their neighbours continuously. However, due to the fact that susceptible nodes are not in their peak when this procedure starts, number of infected will not rise by a lot.

![Graph](image)

Figure 2.12: Infected nodes versus time after the application of SIRS model on a Watts Strogatz network with $N = 10000$, $<k> = 14$, $p = 0.1$. The SIRS parameters are $\tau = 0.022$, $t_I = 8$, $t_R = 70$. The results are averages of 200 simulations.

The outcome of the increase of $t_R$ is that the recovered nodes do not become susceptible fast enough in order to be infected again, so the oscillations stop.

2.6 SIRS on a multiplex network of 2 Watts Strogatz networks

An interesting question is whether or not by connecting a Watts Strogatz network on which is applied SIRS model with parameters that produce oscillations of the number of infected nodes that fade out quickly with another network, will create sustained oscillations. One way to do that would be to create one multiplex network, which could be a combination two Watts Strogatz networks. In this model the new parameter is the transmissibility $\tau'$ for every link between two nodes of different networks. The simulation will always start by infecting one random node from the same network where $t_I$ and $t_R$ will be those of figure 2.12, which we will be refering to as the first Watts Strogatz network. The network parameters for both networks will be those we used in the previous section, i.e. $<k> = 14$, rewiring probability $p = 0.1$ and $N = 10000$. In the next figures we assume that the same infection has SIRS parameters that have a small difference in the two networks, as even though the value of these parameters is characteristic of the infection, the network could diversify them slightly.
2.6.1 Variation of parameter $t_{I_2}$

Figure 2.13: Infected of a multiplex network of 2 Watts Strogatz networks after the application of SIRS model. Black line represents the Infected nodes of the first network versus time and yellow line represents Infected nodes of the second network versus time. Each one of those networks has $N = 10000$, $< k >= 14$, $p = 0.1$, $\tau' = \tau/10$. The SIRS parameters are $\tau = 0.022$, $t_{I_1} = 8$, $t_{R_1} = 70$, $t_{I_2} = 6$, $t_{R_2} = 66$. The results are averages of 200 simulations.

Figure 2.14: Infected(t) of a multiplex network of 2 Watts Strogatz networks after the application of SIRS model. Black line represents the Infected nodes of the first network versus time and yellow line represents Infected nodes of the second network versus time. Each one of those networks has $N = 10000$, $< k >= 14$, $p = 0.1$, $\tau' = \tau/10$. The SIRS parameters are $\tau = 0.022$, $t_{I_1} = 8$, $t_{R_1} = 70$, $t_{I_2} = 9$, $t_{R_2} = 66$. The results are averages of 200 simulations.
Figure 2.15: $\text{Infected}(t)$ of a multiplex network of 2 Watts Strogatz networks after the application of SIRS model. Black line represents the Infected nodes of the first network versus time and yellow line represents Infected nodes of the second network versus time. Each one of those networks has $N = 10000$, $< k > = 14$, $p = 0.1$, $\tau' = \tau / 10$. The SIRS parameters are $\tau = 0.022$, $t_{I_1} = 8$, $t_{R_1} = 70$, $t_{I_2} = 10$, $t_{R_2} = 66$. The results are averages of 200 simulations.

We notice that the increase of $t_{I_2}$ unsurprisingly increases the peaks of the infected nodes of the second Watts Strogatz network. Smaller $t_{I_2}$ delays the spread of the infection resulting to peaks in later time steps. In this case, this delay leads to a small difference in the peaks of the first
and the second Watts Strogatz networks. Therefore, when the infected nodes of the first Watts Strogatz approach their minimum, some of its susceptible nodes could become infected from the infected nodes of the second network via interlinks, so that the oscillations of the two networks are sustained. When we increase $t_{I_2}$, we do not observe the same difference in the peaks of the two networks, hence we observe smaller peaks in later time steps.
2.6.2 Variation of parameter $t_{R_2}$

![Figure 2.17](image1)

Figure 2.17: Infected($t$) of a multiplex network of 2 Watts Strogatz networks after the application of SIRS model. Black line represents the Infected nodes of the first network versus time and yellow line represents Infected nodes of the second network versus time. Each one of those networks has $N = 10000$, $< k >= 14$, $p = 0.1$, $\tau' = \tau/10$. The SIRS parameters are $\tau = 0.022$, $t_{I_1} = 8$, $t_{R_1} = 70$, $t_{I_2} = 8$, $t_{R_2} = 65$. The results are averages of 200 simulations.

![Figure 2.18](image2)

Figure 2.18: Infected($t$) of a multiplex network of 2 Watts Strogatz networks after the application of SIRS model. Black line represents the Infected nodes of the first network versus time and yellow line represents Infected nodes of the second network versus time. Each one of those networks has $N = 10000$, $< k >= 14$, $p = 0.1$. The SIRS parameters are $\tau = 0.022$, $t_{I_1} = 8$, $t_{R_1} = 70$, $t_{I_2} = 8$, $t_{R_2} = 68$, $\tau' = \tau/10$. The results are averages of 200 simulations.
Figure 2.19: Infected($t$) of a multiplex network of 2 Watts Strogatz networks after the application of SIRS model. Black line represents the Infected nodes of the first network versus time and yellow line represents Infected nodes of the second network versus time. Each one of those networks has $N = 10000$, $<k> = 14$, $p = 0.1$. The SIRS parameters are $\tau = 0.022$, $t_{I_1} = 8$, $t_{R_1} = 70$, $t_{I_2} = 8$, $t_{R_2} = 72$, $\tau' = \frac{\tau}{10}$. The results are averages of 200 simulations.

Figure 2.20: Infected($t$) of a multiplex network of 2 Watts Strogatz networks after the application of SIRS model. Black line represents the Infected nodes of the first network versus time and yellow line represents Infected nodes of the second network versus time. Each one of those networks has $N = 10000$, $<k> = 14$, $p = 0.1$. The SIRS parameters are $\tau = 0.022$, $t_{I_1} = 8$, $t_{R_1} = 70$, $t_{I_2} = 8$, $t_{R_2} = 75$. The results are averages of 200 simulations.

The result of smaller values of $t_{R_2}$ is that the graph of the infected nodes of the second Watts Strogatz network is similar to 2.11. The recovered nodes of this network become susceptible quickly and the peaks of the infected nodes reach a plateau. Moreover, the peaks of the infected nodes of
the second network are observed in slightly smaller time steps than the peaks of the infected nodes of the first network. This fact could lead to the infection of some of the nodes of the first network via interlinks, therefore we observe that the system has sustained oscillations. On the other hand, bigger values of $t_{R_2}$ lead to smaller peaks of the second network similar to figure 2.12. Thus, as we observe the height of the peaks of the oscillations of the infected nodes are decreasing.

### 2.6.3 Variation of parameter $\tau$

![Figure 2.21: Infected(t) of a multiplex network of 2 Watts Strogatz networks after the application of SIRS model. Black line represents the Infected nodes of the first network versus time and yellow line represents Infected nodes of the second network versus time. Each one of those networks has $N = 10000$, $<k> = 14$, $p = 0.1$. The SIRS parameters are $\tau = 0.017$, $t_{I_1} = 8$, $t_{R_1} = 70$, $t_{I_2} = 6$, $t_{R_2} = 66$, $\tau' = tau/10$. The results are averages of 200 simulations](image-url)
In this subsection we adjust the transmissibility $\tau$ that the infection is spreading via intralinks in both networks. The form of the infected nodes of the two networks when $\tau$ will be equal to relatively low values, will be similar to figure 2.7. On the contrary, as we have explained above, when $\tau$ is increased, a large fraction of the network could be increased in small period of time steps. If we consider that $\tau'$ is also increased ($\tau' = \tau/10$) and nodes of the two networks are also being infected via interlinks, we conclude that an even bigger fraction of the networks are infected during that time steps. The result is that a lot of nodes are being recovered and susceptible again within a small difference of timesteps, thus the oscillations of infected nodes, "die".
2.6.4 Variation of parameter $\tau'$

Figure 2.23: Infected($t$) of a multiplex network of 2 Watts Strogatz networks after the application of SIRS model. Black line represents the Infected nodes of the first network versus time and yellow line represents the Infected nodes of the second network versus time. Each one of those networks has $N = 10000$, $< k > = 14$, $p = 0.1$. The SIRS parameters are $\tau = 0.022$, $t_{I_1} = 8$, $t_{R_1} = 70$, $t_{I_2} = 6$, $t_{R_2} = 66$, $\tau' = \text{tau}/500$. The results are averages of 200 simulations.

Figure 2.24: Infected($t$) of a multiplex network of 2 Watts Strogatz networks after the application of SIRS model. Black line represents the Infected nodes of the first network versus time and yellow line represents the Infected nodes of the second network versus time. Each one of those networks has $N = 10000$, $< k > = 14$, $p = 0.1$. The SIRS parameters are $\tau = 0.022$, $t_{I_1} = 8$, $t_{R_1} = 70$, $t_{I_2} = 6$, $t_{R_2} = 66$, $\tau' = 1$. The results are averages of 200 simulations.
Small tranmissibility $\tau'$ equals lower probability of infection via interlinks, therefore we observe a slower start that the nodes of the second network are infected as well as the peaks that represent the number of infected nodes will be lower than if $\tau'$ was equal to $\tau/10$. When $\tau'$ approaches high values, everytime a node of one network is infected, automatically it infects one node of the other one. Hence, as before, a big fraction of the network is infected in a short period of time and the oscillations "die".
Chapter 3

Community detection

In the previous chapters we studied networks of many nodes. The problem when our research is based on networks with big size, is that it is not easy to observe interactions between single nodes as the big number of the interactions makes it difficult to focus in just one. To solve this problem we can break the network to smaller groups of nodes that interact mainly with each other, and rarely with nodes from other groups.

Figure 3.1: Communities of a network

By breaking the network into smaller communities helps us understand the network structure, as it is easier to see which nodes interact more frequently. This information in a social network for
instance, could help us understand how the people of a town are organized i.e. who live nearby, or who work together, depending the type of relations that we are studying. Moreover, this procedure could have an important practical meaning. It would be possible that a property of the network has different values for different communities. For instance, a community which represents a country that none of its citizens is infected from a pandemic disease will have a different approach towards the lockdown metres compared to a different community of the network with a big fraction of people infected.

3.1 Louvain Algorithm

One of the most well known algorithms to organize communities of a large network is Louvain algorithm. Louvain algorithm, named after its inventors’ home town, is a very fast algorithm, which is based on the changes of modularity during the evolution of the algorithm. Modularity is a quantity which indicates whether or not most of the links of the nodes of a community are connecting them. High modularity means more connections between the nodes of the community and low modularity means less connections between the nodes of the community. To begin explaining the algorithm, initially, we assume that each node of the network belongs to one community.

- Every node is moved to the communities of each and everyone of its neighbours and after that we calculate the difference in modularity. When this procedure stops, we assume that the node belongs to the community where we had the most increase in modularity. If no increases were observed, the node remains in its initial community. The process is repeated until no more increases in modularity are observed.

- In the second phase of the algorithm we assume that nodes of the network are the communities created in the previous phase. The same procedure that was followed in phase one is repeated in the new network, by moving the whole community and observing the increase in modularity. The procedure stops when no further increase in modularity is observed.

The first two steps are repeatedly applied to the network until no increase to modularity is possible.
3.2 Modelling SIR on communities

In our thesis we will create a model by applying SIR in a network where we have distinguished its communities. This is important as the properties of each community will determine whether or not this community should be in quarantine or not, something that will change the way that the infection spreads. The most fundamental differences of our models is the criteria we should use to quarantine a community, and what should quarantine mean for the network. Below we will present models that have used different algorithms to cut the links of the quarantined communities and different criteria that are used in order to quarantine a community. These models will be an application of SIR in a Watts Strogatz network, with $N = 10000$ and $t_I = 15$. Exactly as in the SIRS model, at $t = 0$ all nodes are considered susceptible except for one random node which we assume infected. We must also point out that in these models a significant parameter is the initial $R_0$. By applying the equations that relate $R_0$ with $\tau$ we compute the probability that each infected node has to infect a susceptible node. This probability will be the same during the whole simulation, as we assume that all links between a susceptible and an infected node are equally infectious.

For each one of the models, every time we adjust one of its parameters, we will display some basic properties of the network after the application of SIR. These properties, the values of which will be averages of a lot of simulations, will be presented as a function of time. They are the number of infected nodes, the cumulative number of new infected nodes, $R_0$ of the whole network and the number of quarantined communities. In order to calculate $\beta(t)$ so that it will possible to calculate $R_0$ we have to divide the sum of the number of the times that all infected nodes would transmit the disease to their neighbours in a time step to the number of infected nodes. Moreover, we will present a distribution of the numbers of total infected nodes after the end of every simulation.

3.3 1st model

In this model we assume that each community will be quarantined if the fraction $\frac{I_{Comm}}{N_{Comm}}$, (where $I_{Comm}$ are the number of infected nodes of this community and $N_{Comm}$ the total nodes of the community) has a bigger value than $P_{qrn}$ which will be defined by us. Additionally we have to assume that when a community is quarantined every link has 80% probability to be considered cut. The community exits quarantine when there is no infected node left in the community. The parameters of the network are

$$< k > = 14 \quad p = 0.1 \quad N = 10000$$

The parameters that will determine the outcome of this model will be the percentage of total community nodes $P_{qrn}$ that the infected nodes need to be in order for the community to be quarantined, and the initial $R_0$. In every figure, one of those parameters will be adjusted and the other one will be constant. The parameter that will be adjusted will be presented on the title of each subsection. Below we present the graphs of the basic properties of the network.
3.3.1 Variation of parameter initial $R_0$

Figure 3.2: Application of the 1st model to a Watts Strogatz network with $N = 10000$, $p = 0.1$, $<k> = 14$. Initial $R_0 = 1.5$, $P_{qn} = 0.1$, $I_t = 15$. a) Infected nodes versus time b) cumulative function of time of New Infected nodes c) histogram of total Infected nodes at the end of each simulation with mean: 284.1 nodes d) $R_0$ of the network versus time e) number of quarantined communities. The results are averages of 1000 simulations.
3.3. 1ST MODEL

Figure 3.3: Application of the 1st model to a Watts Strogatz network with $N = 10000$, $p = 0.1$, $< k > = 14$. Initial $R_0 = 1.8$, $P_{qrn} = 0.1$, $t_I = 15$. a) Infected nodes versus time b) cumulative function of time of New Infected nodes c) histogram of total Infected nodes at the end of each simulation with mean: **2535.1** nodes d) $R_0$ of the network versus time e) number of quarantined communities. The results are averages of 1000 simulations.
Figure 3.4: Application of the 1st model to a Watts Strogatz network with $N = 10000$, $p = 0.1$, $< k > = 14$. Initial $R_0 = 2.4$, $P_{qrn} = 0.1$, $t_I = 15$. a)Infected nodes versus time b) cumulative function of time of New Infected nodes c) histogram of total Infected nodes at the end of each simulation with mean: 5435.2 nodes d) $R_0$ of the network versus time e) number of quarantined communities. The results are averages of 1000 simulations.
3.3. 1ST MODEL

Figure 3.5: Application of the 1st model to a Watts Strogatz network with $N = 10000$, $p = 0.1$, $< k > = 14$. Initial $R_0 = 2.8$, $P_{qrn} = 0.1$, $t_I = 15$. a) Infected nodes versus time b) cumulative function of time of New Infected nodes c) histogram of total Infected nodes at the end of each simulation with mean: 6309.8 nodes d) $R_0$ of the network versus time e) number of quarantined communities. The results are averages of 1000 simulations
We have to notice how close the graphs of $R_0$ of the network and the number of quarantined communities are related. Everytime that quarantined communities number increases $R_0$ of the network decreases, which is reasonable as in this case the $<k>$ drops and $\beta$ decreases. Concerning the parameters of our model, the initial $R_0$ determines the probability of an infected transmitting the disease to a susceptible node. The bigger initial $R_0$ is the bigger transmissibility $\tau$ will be. Hence, as we can verify from the graphs above, bigger initial $R_0$ means bigger peaks of Infected nodes, even double peaks sometimes. This phenomenon occurs due to the fact that when initial $R_0$ is big the concentration of infected nodes and the transmissibility is big enough so that when communities exit quarantine, they are being infected by the infected nodes of the network. As we would expect the increase in $\tau$ means more total Infected nodes during simulations, bigger values of the network $R_0$.

In order to analyze the impact of $P_{qrn}$ in our model, we set a high initial $R_0$ equal to 3 and observe how the disease is spread by changing the values of $P_{qrn}$. 
3.3. 1ST MODEL

3.3.2 Variation of parameter $P_{qrn}$

Figure 3.6: Application of the 1st model to a Watts Strogatz network with $N = 10000$, $p = 0.1$, $< k > = 14$. Initial $R_0 = 3$, $P_{qrn} = 0.05$, $t_I = 15$. a)Infected nodes versus time b)cumulative function of time of New Infected nodes c)histogram of total Infected nodes at the end of each simulation with mean: 3223.5 nodes d)$R_0$ of the network versus time e)number of quarantined communities. The results are averages of 1000 simulations.
Figure 3.7: Application of the 1st model to a Watts Strogatz network with \( N = 10000, p = 0.1, < k > = 14 \). Initial \( R_0 = 3, P_{qrn} = 0.1 \), \( t_I = 15 \). a) Infected nodes versus time b) cumulative function of time of New Infected nodes c) histogram of total Infected nodes at the end of each simulation with mean: 6656.6 nodes d) \( R_0 \) of the network versus time e) number of quarantined communities. The results are averages of 1000 simulations.
3.3. 1ST MODEL

Figure 3.8: Application of the 1st model to a Watts Strogatz network with $N = 10000$, $p = 0.1$, $< k > = 14$. Initial $R_0 = 2$, $P_{qrn} = 0.15$, $t_I = 15$. a) Infected nodes versus time b) cumulative function of time of New Infected nodes c) histogram of total Infected nodes at the end of each simulation with mean: 7121.7 nodes d) $R_0$ of the network versus time e) number of quarantined communities. The results are averages of 1000 simulations.
Figure 3.9: Application of the 1st model to a Watts Strogatz network with $N = 10000$, $p = 0.1$, $<k> = 14$. Initial $R_0 = 3$, $P_{qrn} = 0.2$, $t_I = 15$. a) Infected nodes versus time b) cumulative function of time of New Infected nodes c) histogram of total Infected nodes at the end of each simulation with mean: 7557.2 nodes d) $R_0$ of the network versus time e) number of quarantined communities. The results are averages of 1000 simulations.
As we expected, as the $P_{qrn}$ the decreases, the number of infected nodes decreases too and the curve flattens. For big $P_{qrn}$ percentages, we can also notice curves with two peaks, and the explanation is the same as before. When the concentration of infected nodes becomes high before we contain the spread, it is possible that we could observe a second wave of the disease, as the number of communities that are quarantined decreases and their susceptible can be infected via interlinks between communities. We also observe that in bigger values of $P_{qrn}$ the number of the total infected nodes increases, fact that it would logical to assume if we consider that the communities are quarantined in later stages of the spread. Finally, $R_0$ decreases more drastically in smaller $P_{qrn}$.

3.4 2nd model

In this model, the communities are considered to be quarantined if their $R_0$ exceeds $R_{0}^{qrn}$, which we will be defined by us, and it will exit quarantine when its $R_0 = 0$. The links of a quarantined community will be cut the same way as they were cut in the 1st model, i.e. every link will have 80% probability to be cut. $R_0$ of each community will be computed following the same method that we used when we calculated $R_0$ of the whole network, with the difference that we will compute $\beta$ of each community separately each time step. The parameters that will be adjusted will be the initial $R_0$ and $R_{0}^{qrn}$.

3.4.1 Variation of parameter initial $R_0$

In the figures we present below the parameter that is being changed is the initial $R_0$. $R_{0}^{qrn}$ is constant and equal to

$$R_{0}^{qrn} = 1$$
Figure 3.10: Application of the 2nd model to a Watts Strogatz network with $N = 10000$, $p = 0.1$, $<k> = 14$. Initial $R_0 = 1.5$, $R_0^{\text{qrn}} = 1$, $t_f = 15$. a) Infected nodes versus time b) cumulative function of time of New Infected nodes c) histogram of total Infected nodes at the end of each simulation with mean: 16.9 nodes d) $R_0$ of the network versus time e) number of quarantined communities. The results are averages of 1000 simulations.
Figure 3.11: Application of the 2nd model to a Watts Strogatz network with $N = 10000$, $p = 0.1$, $< k > = 14$. Initial $R_0 = 1.8$, $R_0^\text{wm} = 1$, $t_I = 15$. a) Infected nodes versus time b) cumulative function of time of New Infected nodes c) histogram of total Infected nodes at the end of each simulation with mean: **123.5** nodes d) $R_0$ of the network versus time e) number of quarantined communities. The results are averages of 1000 simulations.
Figure 3.12: Application of the 2nd model to a Watts Strogatz network with $N = 10000$, $p = 0.1$, $<k> = 14$. Initial $R_0 = 2.4$, $R_0^{quar} = 1$, $t_f = 15$. a) Infected nodes versus time b) cumulative function of time of New Infected nodes c) histogram of total Infected nodes at the end of each simulation with mean: 2629.4 nodes d) $R_0$ of the network versus time e) number of quarantined communities. The results are averages of 1000 simulations.
3.4. 2ND MODEL

Figure 3.13: Application of the 2nd model to a Watts Strogatz network with $N = 10000$, $p = 0.1$, $< k > = 14$. Initial $R_0 = 2.8$, $R_{0, in} = 1$, $t_f = 15$. a) Infected nodes versus time b) cumulative function of time of New Infected nodes c) histogram of total Infected nodes at the end of each simulation with mean: 4314.8 nodes d) $R_0$ of the network versus time e) number of quarantined communities. The results are averages of 1000 simulations.

The increase of the initial $R_0$ equals the increase in the transmissibility of the infection.
Similarly as the model where each community is quarantined based on the number of its infected nodes, this increase induces bigger peaks of infected nodes, more total infected nodes, bigger network $R_0$. If we compare the two models, we conclude that when the communities are quarantined based on their $R_0$, we observe smaller peaks of infected nodes, single peaks and less total infected nodes. Thus we deduce that the infection is controlled better.

### 3.4.2 Variation of parameter $R_{qrn}^0$

In the next figures, the initial $R_0$ is considered equal to a high value, so that we can observe if the change of $R_{qrn}^0$ will restrict the spread of the infection.

\[ R_0 = 3 \]
Figure 3.14: Application of the 2nd model to a Watts Strogatz network with $N = 10000$, $p = 0.1$, $< k > = 14$. Initial $R_0 = 3$, $R_{0qrn} = 1.5$, $t_I = 15$. a) Infected nodes versus time b) cumulative function of time of New Infected nodes c) histogram of total Infected nodes at the end of each simulation with mean: 4998.3 nodes d) $R_0$ of the network versus time e) number of quarantined communities. The results are averages of 1000 simulations.
Figure 3.15: Application of the 2nd model to a Watts Strogatz network with $N = 10000$, $p = 0.1$, $<k> = 14$. Initial $R_0 = 3$, $R_{qrn} = 2$, $t_I = 15$. a) Infected nodes versus time b) cumulative function of time of New Infected nodes c) histogram of total Infected nodes at the end of each simulation with mean: 5126.1 nodes d) $R_0$ of the network versus time e) number of quarantined communities. The results are averages of 1000 simulations.
Figure 3.16: Application of the 2nd model to a Watts Strogatz network with \( N = 10000, p = 0.1, < k >= 14 \). Initial \( R_0 = 3, R_0^{qrn} = 2.5, t_I = 15 \). a)Infected nodes versus time b) cumulative function of time of New Infected nodes c) histogram of total Infected nodes at the end of each simulation with mean: \( 5636.7 \) nodes d) \( R_0 \) of the network versus time e) number of quarantined communities. The results are averages of 1000 simulations.
Figure 3.17: Application of the 2nd model to a Watts Strogatz network with \( N = 10000 \), \( p = 0.1 \), \( < k > = 14 \). Initial \( R_0 = 3 \), \( R_0^{\text{qrn}} = 3 \), \( t_I = 15 \). a) Infected nodes versus time b) cumulative function of time of New Infected nodes c) histogram of total Infected nodes at the end of each simulation with mean: 5991.6 nodes d) \( R_0 \) of the network versus time e) number of quarantined communities. The results are averages of 1000 simulations.

As we would expect, smaller \( R_0^{\text{qrn}} \) slows down the spread of the disease. Generally speaking,
we observe low numbers of total infected nodes and low peaks in this model comparing it to the 1st one. Double peaks are not observed even if \( \tau \) is high. However the question is how realistic this model is as the criterion used in order for a community to be quarantined could be considered strict in realistic terms.

### 3.5 3rd model

The difference between this model and the other two ones is the algorithm that cuts the links of the the nodes that belong in the quarantined communities. In this case we consider that instead of every link of a quarantined community having the same probability to be cut, now for each link there will be different probabilities of being cut. We assume that all the nodes will be divided into two categories, high and low priority nodes. The majority of the nodes are considered to be low priority, i.e. nodes that lose most of their connections. However we also have to acknowledge that during quarantine there are some nodes which lose a small percentage of their connections mainly due to their profession, such as doctors or super market workers nodes that will be considered as high priority ones. Consequently the types of links are

- Links between high and high priority nodes. We assume that these links have 0% probability to be cut if the community is quarantined.
- Links between high and low priority nodes. These links will have 50% probability to be cut if the community is quarantined.
- Links between low and low priority nodes. These links have 100% probability to be cut if the community is quarantined.

The category each node belongs to is determined by a probability we define. In this model we assumed that each node has \( p_{high} = 0.2 \) probability to be high priority and \( p_{low} = 0.8 \) probability to be low priority. The criterion of a community being quarantined will be the same as the first model, i.e. if the percentage of the infected nodes of a community is bigger than \( P_{qrn} \). Applying this algorithm one single time, if all the communities of the network are quarantined, the mean values of the percentage of connections the low priority and high priority nodes lose and the percentage of the total connections that were cut are respectively equal to

\[
< P_{cut,low} > \simeq 90\% \quad < P_{cut,high} > \simeq 40\% \quad P_{cut,total} \simeq 80\%
\]

We have to note that in this model, it is important how many connections each node has. Thus in order to compare it with the first model, we will increase the diversity of the distribution of \( k \) of the nodes of the network by increasing the rewiring probability to 0.6. Moreover, because this algorithm requires big computational time, we will decrease the average \( k \) of the network to \(< k > = 10 \) which is a realistic value. In the next section we will present for the third model the plots for Infected, new Infected, \( R_0 \) of the network, and number of communities for initial \( R_0 = 2 \), and two values of \( P_{qrn} \). Finally we will compare the 1st and the 3rd model for the same values of \( P_{qrn} \) considering that when we apply the 1st model, every link of the community has 80% percent probability to be cut if the community is in quarantine.
3.5.1 \( P_{qrn} = 0.05 \)

Figure 3.18: Application of the 3rd model to a Watts Strogatz network with \( N = 10000 \), \( p = 0.6 \), \( < k > = 10 \), \( t_I = 15 \). Initial \( R_0 = 2 \), \( P_{qrn} = 0.05 \). a) Infected nodes versus time b) cumulative function of time of New Infected nodes c) histogram of total Infected nodes at the end of each simulation with mean: 3888.9 nodes d) \( R_0 \) of the network versus time e) number of quarantined communities. The results are averages of 1000 simulations.
Then, we apply the first model to a Watts Strogatz with the same network properties

\[ N = 10000 \quad p = 0.6 \quad < k >= 10 \]

and we compare the 1st and the 3rd model

![Graphs showing infected nodes versus time for the 1st (blue line) and the 3rd (red line) models.](a)

![Histogram of the total infected nodes of the 1st model after the end of simulations with mean: 4038.5 nodes.](b)

**Figure 3.19:** Application of the 1st model to a Watts Strogatz network with \( N = 10000 \), \( p = 0.6 \), \( < k >= 10 \), \( t_I = 15 \). Initial \( R_0 = 2 \), \( P_{qrn} = 0.05 \). a) Infected nodes versus time for the 1st (blue line) and the 3rd (red line) models. b) Histogram of the total infected nodes of the 1st model after the end of simulations with mean: **4038.5** nodes. The results are averages of 1000 simulations.
3.5.2 $P_{qrn} = 0.1$

Figure 3.20: Application of the 3rd model to a Watts Strogatz network with $N = 10000$, $p = 0.6$, $< k >= 10$, $t_I = 15$. Initial $R_0 = 2$, $P_{qrn} = 0.1$. a) Infected nodes versus time b) cumulative function of time of New Infected nodes c) histogram of total Infected nodes at the end of each simulation with mean: $4264.5$ nodes d) $R_0$ of the network versus time e) number of quarantined communities. The results are averages of 1000 simulations.
3.5. 3RD MODEL

Figure 3.21: Application of the 1st model to a Watts Strogatz network with \( N = 10000 \), \( p = 0.6 \), \(< k > = 10\), \( t_I = 15 \). Initial \( R_0 = 2 \), \( P_{qrn} = 0.1 \). a) Infected nodes versus time for the 1st (blue line) and the 3rd (red line) models. b) Histogram of the total Infected nodes of the 1st model after the end of simulations with mean: 4400 nodes. The results are averages of 1000 simulations.

We conclude that the graphs of the properties produced by the 1st and the 3rd model are very similar with the difference that we observe less infected nodes as a function of time with small difference when we apply the 3rd model. Thus, the total infected nodes in this case are less.
Conclusions

To conclude this thesis, we have researched two topics. The first topic was an application of a SIRS model in a multiplex network of two Watts Strogatz networks and we focused on the graph of the Infected nodes as a function of time. We noticed that this system was very sensitive to the changes of the SIRS parameters $t_I$, $t_R$, $\tau$, $\tau'$, since a very small change in only one of those parameters could result in a totally different graph. As we observed, some combinations of those parameters could lead to the increment of the number of infected nodes of one or both of the networks and helping to preserve the disease in the system for many time steps. However we observed the opposite scenario too, i.e. combinations of SIRS parameters which led the oscillations of infected to "die" quicker than if the disease was spreading to only one network.

The second topic was an application of SIR on a single Watts Strogatz network where the communities of the network were being inserted into quarantine. This part of the thesis has taught us how important the quarantine metres the way we perceived them in our models are, as they slow down sometimes significantly the disease spreading. We proposed different models in order to approach a realistic system that is being quarantined and observed the importance of the criterion that was necessary in order for the communities to be quarantined, as well as how much the properties of the network are being affected after changing this criterion. The conclusion was that the more strict this criterion was, the less people were infected as we would expect.
Appendices
Appendix A

Source code of 2.6

```python
import networkx as nx
import random
import numpy as np
import sys

class SIRS:
    def __init__(self, tau1, tI1, tR1, tI2, tR2, S, I, pWatts, nei, maxtime):
        self.maxtime = maxtime
        self.tI1 = tI1
        self.tR1 = tR1
        self.tI2 = tI2
        self.tR2 = tR2
        self.N = S + I
        self.different = np.zeros(self.N)
        self.total = np.array([])
        self.total = np.append(self.total, 1)

        self.tau1 = tau1  # transmissibility of the intralinks of the 1st network
        self.tau2 = self.tau1
        self.tau_inter = self.tau1 / 10  # transmissibility

        # we create the networks
        self.graph1 = nx.watts_strogatz_graph(self.N, k=nei, p=pWatts)
        self.graph2 = nx.watts_strogatz_graph(self.N, k=nei, p=pWatts)

        self.neighbours1 = []
        self.neighbours2 = []

        komvoi1 = np.array(nx.nodes(self.graph1))
        komvoi2 = np.array(nx.nodes(self.graph2))

        for i in komvoi1:
            self.neighbours1.append(list(self.graph1.neighbors(i)))

        for i in komvoi2:
            self.neighbours2.append(list(self.graph2.neighbors(i)))

        self.S_node1 = komvoi1  # initially all nodes are S
        self.I_node1 = np.array([], int)
        self.R_node1 = np.array([], int)

        self.S_node2 = komvoi2
        self.I_node2 = np.array([], int)
        self.R_node2 = np.array([], int)

        self.arr = random.randint(0, self.N - 1)  # random initial sick
        arrwstoIndexTmp = np.where(self.S_node1 == self.arr)

        if len(arrwstoIndexTmp) != 1:
            sys.exit('error')

        arrwstoIndex = arrwstoIndexTmp[0][0]
```

APPENDIX A. SOURCE CODE OF 2.6

```python
self.different [self.arr] = 1
self.S_node1 = np.delete(self.S_node1, arrwstoIndex)
self.I_node1 = np.append(self.I_node1, self.arr) #the initial infected node

self.t1array1 = np.zeros(self.N) + 2*self.maxtime #time that each node will recover
#initially we assume a big value so that they won't recover
self.t1array1[self.arr] = self.t1
self.tRarray1 = np.zeros(self.N) + 2*self.maxtime
self.tRarray1 = np.zeros(self.N) + 2*self.maxtime

self.Infected1 = np.array([])
self.Infected1 = np.append(self.Infected1, 1)
#number of infected(t) nodes of the 1st network
self.Infected2 = np.array([])
self.Infected2 = np.append(self.Infected2, 0)
#number of infected(t) nodes of the 2nd network

self.t = 1
self.susceptible = np.array([])
self.recovered = np.array([])
self.susceptible = np.append(self.susceptible, S)
self.recovered = np.append(self.recovered, 0)
self.newInf1 = np.array([1])
#new Infected(t) nodes of the 1st network

def run(self):
    while self.t<self.maxtime:
        newI1 = np.array([], int)
        newI2 = np.array([], int)

        for node in self.I_node1: #node spreads the disease via intralinks and interlinks
            for kapoios in self.neighbours1[node]:
                if (random.random() < self.tau1): #kapoios is infected
                    if kapoios in self.S_node1 and kapoios not in newI1:
                        self.t1array1[kapoios] = self.t + self.t1 #time kapoios will recover
                        arrwstoTpl = np.where(self.S_node1==kapoios)
                        if len(arrwstoTpl) != 1:
                            sys.exit('error')
                        arrwsto = arrwstoTpl[0][0]
                        self.S_node1 = np.delete(self.S_node1, arrwsto)
                        newI1 = np.append(newI1, kapoios)

        if random.random() < self.tau_inter:
            if node in self.S_node2 and node not in newI2:
                arrwstoTpl = np.where(self.S_node2==node)
                if len(arrwstoTpl) != 1:
                    sys.exit('error')
                arrwsto = arrwstoTpl[0][0]
                self.S_node2 = np.delete(self.S_node2, arrwsto)
                newI2 = np.append(newI2, node)
                self.t1array2[node] = self.t + self.t1

        if self.t >= self.t1array1[node]: #node recovers
            self.t1array1[node] = self.t + self.tR1
            recover_tpl = np.where(self.I_node1==node)
            if len(recover_tpl) != 1:
                sys.exit('error')
            recover_ind = recover_tpl[0][0]
            self.I_node1 = np.delete(self.I_node1, recover_ind)
            self.R_node1 = np.append(self.R_node1, node)

        for node in self.I_node2: #node spreads the disease via intralinks and interlinks
            for kapoios in self.neighbours2[node]:
                if (random.random() < self.tau2):
                    if kapoios in self.S_node2 and kapoios not in newI2:
                        self.t1array2[kapoios] = self.t + self.t1
                        newI2 = np.append(newI2, kapoios)
                        arrwstoTpl = np.where(self.S_node2==kapoios)
                        if len(arrwstoTpl) != 1:
                            sys.exit('error')
                        arrwsto = arrwstoTpl[0][0]
                        self.S_node2 = np.delete(self.S_node2, arrwsto)
```

```
```python
self.S_node2 = np.delete(self.S_node2, arrwsto)

if random.random() < self.tau_inter:
    if node in self.S_node1 and node not in new1:
        arrwstoTpl = np.where(self.S_node1==node)
        if len(arrwstoTpl) != 1:
            sys.exit('error')
        arrwsto = arrwstoTpl[0][0]
        self.S_node1 = np.delete(self.S_node1, arrwsto)
        new1 = np.append(new1, node)
        self.tIarray1[node] = self.t + self.tI1

    if self.t >= self.tIarray2[node]:
        self.tRarray2[node] = self.t + self.tR2
        recover_tpl = np.where(self.I_node2==node)
        if len(recover_tpl) != 1:
            sys.exit('error')
        recover_ind = recover_tpl[0][0]
        self.I_node2 = np.delete(self.I_node2, recover_ind)
        self.R_node2 = np.append(self.R_node2, node)

self.I_node1 = np.append(self.I_node1, new1)
self.newInf1 = np.append(self.newInf1, len(new1))
self.I_node2 = np.append(self.I_node2, new1)

for node in self.R_node1:
    if self.t >= self.tRarray1[node]:  # node becomes susceptible
        susTpl = np.where(self.R_node1==node)
        if len(susTpl) != 1:
            sys.exit('error')
        susInd = susTpl[0][0]
        self.R_node1 = np.delete(self.R_node1, susInd)
        self.S_node1 = np.append(self.S_node1, node)

self.Infected1 = np.append(self.Infected1, len(self.I_node1))
self.Infected2 = np.append(self.Infected2, len(self.I_node2))
self.susceptible = np.append(self.susceptible, len(self.S_node1))
self.recovered = np.append(self.recovered, len(self.R_node1))

self.t += 1
random.shuffle(self.I_node1)
random.shuffle(self.I_node2)

return [self.Infected1, self.Infected2, self.susceptible, self.recovered, self.newInf1]
```

2NetsParallel.py
Appendix B

Source code of 3.3

```
import numpy as np
import random
import networkx as nx
import sys
import community
from collections import Counter
import copy

# Class SIR qrnt

class SIRqrnt():
    def __init__(self, S, I, R0, pWatts, nei, tI, maxtime, cutPerc, percQRNT):
        self.tI = tI
        beta = R0/tI
        self.tau = beta/nei
        self.maxtime = maxtime
        self.N = S + I
        self.graph = nx.watts_strogatz_graph(self.N, k=nei, p=pWatts)  #create the network
        nodesGraph = np.array(nx.nodes(self.graph))
        self.S_node = nodesGraph
        #initially all nodes are S
        self.I_node = np.array([], int)
        self.R_node = np.array([], int)
        self.percQRNT = percQRNT
        #percentage of allowed infection before quarantine
        self.cutPerc = cutPerc
        #percentage of the connections we will cut
        self.inf = np.zeros(self.N)
        self.beta_t = np.zeros(maxtime+1)
        self.R0_t = np.zeros(maxtime+1)
        self.beta_t[0] = beta
        self.R0_t[0] = R0
        self.numCom = np.zeros(maxtime + 1)
        self.numCom[0] = 0

        arrwsto = random.randint(0, self.N - 1)  #initial infected
        arrwstoIndexTmp = np.where(self.S_node==arrwsto)
        if len(arrwstoIndexTmp) != 1:
            sys.exit("error")
        arrwstoIndex = arrwstoIndexTmp[0][0]
        self.S_node = np.delete(self.S_node, arrwstoIndex)
        self.I_node = np.append(self.I_node, arrwsto)
        self.inf[arrwsto] = 1  #node has been infected
        self.tIarray = np.zeros(self.N) + self.maxtime + 1
        self.tIarray[arrwsto] = tI

        self.neighbours = []
        for i in nodesGraph:
            self.neighbours.append(list(self.graph.neighbors(i)))

        self.neighboursQRNT = copy.deepcopy(self.neighbours)
        for i in range(len(self.neighbours)):
```

78
for j in self.neighbours[i]:
    if i<j:
        if random.random() < self.cutPerc:
            self.neighboursQRNT[i].remove(j)
            self.neighboursQRNT[j].remove(i)

self.partition = community.best_partition(self.graph) #communities
self.sumCommunities = Counter(self.partition.values()) #total nodes of each community
self.infCommunity = dict.fromkeys(self.sumCommunities.keys(), 0) #dictionary with keys the communities #and values the infected
self.Infected = np.array([])
self.Infected = np.append(self.Infected, 1)
self.QRNT = np.zeros(len(set(self.partition.values()))) #booleans for each community
self.newInfected = np.array([])
self.newInfected = np.append(self.newInfected, 1)

def run(self):
    t = 1
    while True:
        newI = np.array([], int)
        for i in self.I_node:
            com = self.partition[i]

            if self.QRNT[com] == 0:
                for j in self.neighbours[i]:
                    if j in self.S_node and j not in newI:
                        newI = np.append(newI, j) #it is infected now
                        arrwstoTpl = np.where(self.S_node==j)
                        if len(arrwstoTpl) != 1:
                            sys.exit('error')
                        arrwsto = arrwstoTpl[0][0]
                        self.S_node = np.delete(self.S_node, arrwsto) #not S anymore
                        self.tIarray[j] = t + self.tI #time it will recover

            else:
                for j in self.neighboursQRNT[i]:
                    if random.random() < self.tau:
                        self.beta_t[t] += 1
                        if j in self.S_node and j not in newI:
                            newI = np.append(newI, j) #it is infected now
                            arrwstoTpl = np.where(self.S_node==j)
                            if len(arrwstoTpl) != 1:
                                sys.exit('error')
                            arrwsto = arrwstoTpl[0][0]
                            self.S_node = np.delete(self.S_node, arrwsto) #not S anymore
                            self.tIarray[j] = t + self.tI #time it will recover

            if self.tIarray[i] <= t:
                recover_tpl = np.where(self.I_node==i)
                if len(recover_tpl) != 1:
                    sys.exit('error')
                recover_ind = recover_tpl[0][0]
                self.I_node = np.delete(self.I_node, recover_ind)
                self.R_node = np.append(self.R_node, i)

    self.I_node = np.append(self.I_node, newI) #append the new infected to the initial list
    self.Infected = np.append(self.Infected, len(self.I_node)) #number of infected
    self.newInfected = np.append(self.newInfected, len(newI))
self.infCommunity = dict.fromkeys(self.sumCommunities.keys(), 0)  # infected of each community

self.beta_t[t] = self.beta_t[t-1] / self.Infected[t-1]  # infected of each community
self.R0_t[t] = self.beta_t[t] * self.t

for node in self.I_node:  # count infected nodes of each community
    com = self.partition[node]  # community of each infected
    self.infCommunity[com] += 1
    if self.inf[d][node] != 1:
        self.inf[d][node] = 1  # particular node has been infected

for i in self.infCommunity.keys():
    if self.QRNT[i] == 0:  # criterion to get into quarantine
        if (self.infCommunity[i] / self.sumCommunities[i]) > self.percQRNT:
            self.QRNT[i] = 1
        else:
            if self.infCommunity[i] == 0:  # now we get out of quarantine
                self.QRNT[i] = 0

self.numCom[t] = sum(self.QRNT)

if len(self.I_node) == 0:
    break

for node in self.I_node:
    random.shuffle(self.I_node)

return [self.Infected, totalInf, self.R0_t, self.newInfected, self.numCom]
Appendix C
Source code of 3.4

```python
import numpy as np
import random
import networkx as nx
import sys
import community
from collections import Counter
import copy

class SIRqrint():
    def __init__(self, S, I, R0, R0qrnt, pWatts, nei, tI, maxtime, cutPerc):
        self.tI = tI
        self.maxtime = maxtime
        self.N = S + I
        self.graph = nx.watts_strogatz_graph(self.N, k=nei, p=pWatts)  # we create the network
        nodesGraph = np.array(nx.nodes(self.graph))
        beta = R0/self.tI
        self.tau = beta/nei  # transmissibility
        self.S_node = nodesGraph  # initially all nodes are S
        self.I_node = np.array([], int)
        self.R_node = np.array([], int)
        self.cutPerc = cutPerc  # percentage of the connections we will cut
        self.R0qrnt = R0qrnt
        self.infd = np.zeros(self.N)  # different infected nodes
        self.beta_t = np.zeros(self.maxtime+1)
        self.R0_t = np.zeros(self.maxtime+1)
        self.beta_t[0] = beta
        self.R0_t[0] = R0
        self.numCom = np.zeros(self.maxtime + 1)
        self.numCom[0] = 0
        self.partition = community.best_partition(self.graph)  # communities
        self.sumCommunities = Counter(self.partition.values())  # total nodes of each community
        self.infCommunity = dict.fromkeys(self.sumCommunities.keys(), 0)  # dictionary with keys the communities and values the infected

        arrwsto = random.randint(0, self.N - 1)  # initial infected
        arrwstoIndexTmp = np.where(self.S_node==arrwsto)
        if len(arrwstoIndexTmp) != 1:
            sys.exit('error')
        arrwstoIndex = arrwstoIndexTmp[0][0]
        self.S_node = np.delete(self.S_node, arrwstoIndex)
        self.I_node = np.append(self.I_node, arrwstoIndex)
        self.infd[arrwsto] = 1  # node has been infected
        com = self.partition[arrwsto]
        self.infCommunity[com] += 1
        self.init = np.zeros(self.maxtime + 1)
        self.init[0] = tI
```

81
APPENDIX C. SOURCE CODE OF 3.4

```python
    self.neighbours = []
    for i in nodesGraph:  # neighbours
        self.neighbours.append(list(self.graph.neighbors(i)))

    self.neighboursQRNT = copy.deepcopy(self.neighbours)

    for i in range(len(self.neighbours)):
        for j in self.neighbours[i]:
            if i < j:
                if random.random() < self.cutPerc:
                    self.neighboursQRNT[i].remove(j)
                    self.neighboursQRNT[j].remove(i)

    self.Infected = np.array([])
    self.Infected = np.append(self.Infected, 1)
    self.newInfected = np.array([])
    self.newInfected = np.append(self.newInfected, 1)
    self.QRNT = np.zeros(len(set(self.partition.values())))  # booleans for each community
    self.R0_Com = np.zeros(len(set(self.partition.values())))  # R0 of each community

    def run(self):
        t = 1
        see = []
        while True:
            self.beta_Com = np.zeros(len(set(self.partition.values())))
            newI = np.array([], int)
            for i in self.I_node:
                com = self.partition[i]  # if it didn’t recover, it contacts neighbours
                if self.QRNT[com] == 0:
                    for j in self.neighbours[i]:
                        inactive = False
                        comj = self.partition[j]
                        if self.QRNT[comj] == 1:
                            if i not in self.neighboursQRNT[j]:
                                inactive = True
                    if inactive == False:
                        if random.random() < self.tau:
                            self.beta_t[t] += 1  # beta of the community
                            if j in self.S_node and j not in newI:
                                newl = np.append(newl, j)  # it is infected now
                                arrwstoTpl = np.where(self.S_node == j)
                                if len(arrwstoTpl) != 1:
                                    sys.exit('error')
                                arrwsto = arrwsto[0][0]
                                self.S_node = np.delete(self.S_node, arrwsto)  # not S anymore
                                self.tIarray[j] = t + self.tI  # time it will recover
                        else:
                            for j in self.neighboursQRNT[i]:
                                if random.random() < self.tau:
                                    self.beta_com[i] += 1  # beta of the community
                                    if j in self.S_node and j not in newI:
                                        newl = np.append(newl, j)  # it is infected now
                                        arrwstoTpl = np.where(self.S_node == j)
                                        if len(arrwstoTpl) != 1:
                                            sys.exit('error')
                                        arrwsto = arrwstoTpl[0][0]
                                        self.S_node = np.delete(self.S_node, arrwsto)  # not S anymore
                                        self.tIarray[j] = t + self.tI  # time it will recover

            if self.tIarray[i] <= t:
```
recover_tpl = np.where(self.I_node==i)
if len(recover_tpl) != 1:
sys.exit('error')
recover_ind = recover_tpl[0][0]
sel.I_node = np.delete(self.I_node, recover_ind)
sel.R_node = np.append(self.R_node, i)

sel.I_node = np.append(sel.I_node, newI)  #append the new infected to the initial list
sel.Infected = np.append(sel.Infected, len(sel.I_node))  #number of infected
sel.newInfected = np.append(sel.newInfected, len(newI))

sel.beta_t[t] = sel.beta_t[t] / sel.Infected[t-1]

for comm in range(len(self.beta_Com)):
    if sel.infCommunity[comm]!=0:
    else:
        sel.beta_Com[comm] = 0

sel.R0_Com = sel.beta_Com * sel.tI

sel.infCommunity = dict.fromkeys(sel.sumCommunities.keys(), 0)  #infected of each community
for node in sel.I_node:  #count infected nodes of each community
    com = sel.partition[node]  #community of each infected
    sel.infCommunity[com] += 1
    if sel.infd[node] != 1:
        sel.infd[node] = 1  #particular node has been infected

for i in sel.infCommunity.keys():
    if sel.QRNT[i] == 0:  #community not quarantined
        if sel.R0_Com[i] > sel.R0qrnt:
            sel.QRNT[i] = 1
            see.append(sel.infCommunity[i] / sel.sumCommunities[i])
    else:  #community in quarantine
        if sel.R0_Com[i] <= 0.1:  #R0 of community = 0
            sel.QRNT[i] = 0

sel.numCom[t+1] = sum(sel.QRNT)

if len(sel.I_node) == 0:
    break
random.shuffle(sel.I_node)
totalInf = sum(sel.infd)
return [sel.Infected, totalInf, sel.beta_t, sel.R0_t, sel.numCom, sel.newInfected]

2ndModel.py
import numpy as np
import random
import networkx as nx
import sys
import community
from collections import Counter
import copy

class SIRqrnt:
    def __init__(self, S, I, R0, pWatts, nei, tI, maxtime, cutPerc, percQRNT):
        self.tI = tI
        beta = R0/tI
        self.b = beta/nei
        self.maxtime = maxtime
        self.N = S + I
        self.graph = nx.watts_strogatz_graph(self.N, k=nei, p=pWatts)  # we create the network
        nodesGraph = np.array(nx.nodes(self.graph))
        self.S_node = nodesGraph
        # initially all nodes are S
        self.I_node = np.array([[]], int)
        self.R_node = np.array([[]], int)
        self.percQRNT = percQRNT  # percentage of allowed infection before quarantine
        self.cutPerc = cutPerc  # percentage of the connections we will cut
        self.inf = np.zeros(self.N)
        priority = 0.2  # concentration of high priority nodes
        highPr = []
        self.beta_t = np.zeros(self.maxtime+1)
        self.R0_t = np.zeros(self.maxtime+1)
        self.beta_t[0] = beta
        self.R0_t[0] = R0

        for i in nodesGraph:
            if random.random() < priority:
                highPr.append(i)  # high priority nodes

        arrwsto = random.randint(0, self.N - 1)  # random initial infected
        arrwstoIndexTmp = np.where(self.S_node == arrwsto)
        if len(arrwstoIndexTmp) != 1:
            sys.exit('error')
        arrwstoIndex = arrwstoIndexTmp[0][0]
        self.S_node = np.delete(self.S_node, arrwstoIndex)
        self.I_node = np.append(self.I_node, arrwsto)
        self.inf[i] = 1  # node i has been infected
        self.tIarray = np.zeros(self.N) + self.maxtime + 1
        self.tIarray[arrwstoIndex] = tI

        self.neighbours = []
        for i in nodesGraph:  # neighbours
            for j in self.graph.neighbors(i):
                if j != i:
                    self.neighbours.append(j)

for i in nodesGraph:
    # neighbours

self.neighbours.append(list(self.graph.neighbors(i)))

self.neighboursQRNT = copy.deepcopy(self.neighbours)

for i in highPr:
    for j in self.neighbours[i]:
        if j not in highPr:
            if random.random() < 0.5:
                self.neighboursQRNT[i].remove(j)
                self.neighboursQRNT[j].remove(i)

for i in nodesGraph:
    if i not in highPr:
        keepJ = []
        for j in self.neighboursQRNT[i]:
            if j < i:
                if j not in highPr:
                    keepJ.append(j)

    for j in keepJ:
        self.neighboursQRNT[i].remove(j)
        self.neighboursQRNT[j].remove(i)

self.partition = community.best_partition(self.graph)  #communities
self.sumCommunities = Counter(self.partition.values())  #total nodes of each community
self.infCommunity = dict.fromkeys(self.sumCommunities.keys(), 0)  #dictionary with keys the communities
#and values the infected
self.Infected = np.array([], int)
self.Infected = np.append(self.Infected, 1)
self.QRNT = np.zeros(len(set(self.partition.values())), bool)  #booleans for each community
self.newInfected = np.array([], int)
self.newInfected = np.append(self.newInfected, 1)

def run(self):
    t = 1
    while True:
        newI = np.array([], int)
        for i in self.S_node:
            com = self.partition[i]
            #if it didn't recover, it contacts neighbours
            if self.QRNT[com] == 0:
                for j in self.neighbours[i]:
                    #if it is susceptible and not already infected
                    inactive = False
                    comj = self.partition[j]
                    if self.QRNT[comj] == 1:
                        inactive = True
                    if inactive is False:
                        if random.random() < self.beta:
                            self.beta_t[t] += 1
                            if j in self.S_node and j not in newI:
                                newI = np.append(newI, j)  #it is infected now
                                arrwstoTpl = np.where(self.S_node==j)
                                if len(arrwstoTpl) != 1:
                                    sys.exit('error')
                                arrwsto = arrwstoTpl[0][0]
                                self.S_node = np.delete(self.S_node, arrwsto)  #not S anymore
                                self.tIarray[j] = t + self.tI  #time it will recover
            else:
                for j in self.neighboursQRNT[i]:
                    if random.random() < self.beta:
                        self.beta_t[t] += 1
                        if j in self.S_node and j not in newI:
                            newI = np.append(newI, j)  #it is infected now
                            arrwstoTpl = np.where(self.S_node==j)
APPENDIX D. SOURCE CODE OF 3.5

```python
if len(arrwstoTpl) != 1:
    sys.exit('error')
arrwsto = arrwstoTpl[0][0]
sel.S_node = np.delete(self.S_node, arrwsto)  # not S anymore
self.tIarray[j] = t + self.tI  # time it will recover

if self.tIarray[i] <= t:
    recover_tpl = np.where(self.I_node==i)
if len(recover_tpl) != 1:
    sys.exit('error')
recover_ind = recover_tpl[0][0]
self.I_node = np.delete(self.I_node, recover_ind)
self.R_node = np.append(self.R_node, i)

self.I_node = np.append(self.I_node, newI)  # append the new infected to the initial list
self.Infected = np.append(self.Infected, len(self.I_node))  # number of infected
self.newInfected = np.append(self.newInfected, len(newI))
self.infCommunity = dict.fromkeys(self.sumCommunities.keys(), 0)  # infected of each community

self.beta_t[t] = self.beta_t[t]/self.Infected[t-1]
self.R0_t[t] = self.beta_t[t]*self.tI

for node in self.I_node:  # count infected nodes of each community
    com = self.partition[node]  # community of each infected
    self.infCommunity[com] += 1
if self.infd[node] != 1:
    self.infd[node] = 1  # particular node has been infected

for i in self.infCommunity.keys():
    if self.QRNT[i] == 0:  # criterion to get into quarantine
        if (self.infCommunity[i]/self.sumCommunities[i])>self.percQRNT:
            self.QRNT[i] = 1
        else:
            if self.infCommunity[i]==0:  # now we get out of quarantine
                self.QRNT[i] = 0
    if len(self.I_node) == 0:  # break
        break
    t += 1
    random.shuffle(self.I_node)
totInf = sum(self.infd)
return [self.Infected, totInf, self.newInfected, self.R0_t]
```

3rdModel.py
Bibliography


[3] Lucas D. Valdez, Lidia A. Braunstein, and Shlomo Havlin. *Epidemic spreading on modular networks: the fear to declare a pandemic*. 1Department of Physics, Boston University, Boston, Massachusetts 02215, USA 2Instituto de Investigaciones Físicas de Mar del Plata (IFIMAR)-Departamento de Física, FCEyN, Universidad Nacional de Mar del Plata-CONICET, Mar del Plata, Argentina. 3Department of Physics, Bar Ilan University, Ramat Gan, Israel 4Tokyo Institute of Technology, Yokohama, Japan. September 24, 2019


[6] Petter Holme *Model versions and fast algorithms for network epidemiology*, Department of Energy Science, Sungkyunkwan University, Suwon 440-746, Korea IceLab, Department of Physics, Umeå University, 90187 Umeå, Sweden Department of Sociology, Stockholm University, 10691 Stockholm, Sweden


[8] Nino Antulov-Fantulin, Alen Lanciicb, Hrvoje Stefancic, Mile Sikic, FastSIR Algorithm: A Fast Algorithm for simulation of epidemic spread in large networks by using SIR compartment model, aDivision of Electronics, Laboratory for Information Systems, Rudjer Boskovic Institute, Zagreb, Croatia bFaculty of Science, Department of Mathematics, University of Zagreb, Zagreb, Croatia Theoretical Physics Division, Rudjer Boskovic Institute, Zagreb, Croatia dFaculty of Electrical Engineering and Computing, Department of Electronic Systems and Information Processing, University of Zagreb, Croatia eBioinformatics Institute, A*STAR, Singapore, Republic of Singapore, February 2012.

[10] https://towardsdatascience.com/infection-modeling-part-1-87e7464568a