The probability for infection in SIR and SIS networks

by

Jing Kjeldsen

THESIS
for the degree of
MASTER IN MATHEMATICS

(Master of Science)

Faculty of Mathematics and Natural Sciences
University of Oslo

May 24, 2013

Acknowledgement

I am deeply thankful to my supervisor Geir Dahl for giving me this master, for his advices and guidance. I want to express a thank to Ingrid Ødegaard Notø for improving the language and finding typos in this thesis. Also I want to thank Torgeir Børresen for the discussions and for giving me valuable feedback on this thesis.

I am thankful for the competition I have got from my fellow students, where we have pushed each others to take more advanced subjects and solve harder mathematical and algorithmic problems. Competing in an algorithm-team together with Markus, Torgeir and later Ola, have really been a great motivation.

I would give a thanks to the friends I have met at the University of Oslo for giving me five wonderful years. This have been a great journey filled with memories I will never forget. A special thanks goes to the people at the north side of the study hall in the eight floor of Abel, who I have shared the last year of my master writing with. I am grateful for my family for their love and support. At last I want to give thanks to the Creator and Engineer of all the universe, the One who made it all possible.

Table of contents

CHAPTER 1

Introduction

In 1927, W.O. Kermacks and A.G. McKendric created a model where they considered a fixed population, and looked at the number of inidividuals in the population at time t which was suspected $S(t)$, infected $I(t)$ and resistant $R(t)$ (see [**9**]). $S(t)$ is the number of the popluation which have not been infected at the given time. $I(t)$ is the number of the population which has been infected and can spread the disease further to those in the suspected category. $R(t)$ is the number of the population who in the past was infected, and either has recovered or died. The assumption that was made with this model, is that an suspected individual has probability β for getting the disease, and an infected individual γ for recovering. In this model we does not take in account who intereacts with who. For infections like the flu, it is impossible to track the infection route since a walk in the street set you at risk for being infected by hundreds of persons.

For infections like tuberculosis which are not very contagious, we are most likely to be infected by a person we have contact with for long periods. For these cases, a model that fits better is by one that has a underlying friendship graph, with transmission edges between two persons if they are friends. Some people have a greater chance for getting the infection because of poor immune system, while others have lower change of infection because of better immune. Some people, called hub, has a lot of contact with other people, and therefore may have greater risk of infection. Others have a small and closed friendship circle, and may not face the same risk. Since people are different, we need to take in to account the social structure, and have individual probabilites for each transmission link. We may represent the social structure mathematically by a graph.

In this thesis we will work with two models of epidemic network called the SIS network and the SIR network. We are interested in probabilites for infection and have primary found three properties that holds for both networks: The monotonicity properties, the epidemic threshold and the that computing the probabilites is #P-hard (which means we almost surely can know that exists no polynomial algorithm). Also we will describe two algorithms computing the probabilities for infection, such that we can test these properties.

Chapter 2 introduces some background materials. This involves some basic graph theory, probability theory in the language of measure theory and a classifaction of problems after time complexity. The theory from this chapter will be applied throughout the master.

In Chapter 3 we introduce two epidemics models, the SIS-network and SIR-network. My contribution to this chapter has been to create a notation that is familiar in probability theory.

In Chapter 4 we prove the monotonicity property for the two networks. Primary, this means that by vaccinating a person may cause a backfire and increase the probability for a epidemic. My contribution to this chapter has been to fill out the proofs of the first section, making it a complete proof. I have contributed with all of the second and third section proving in two different ways that the monotonicity properties also holds for the SIS-grahs.

In Chapter 5 we talk about epidmic thresholds for the two networks. If there exists an epidemic threshold, then by satisfying the condition there is a probability for an epidemic breakout. My contribution to this chapter has been to generalize the theorems such that it also holds for networks with individual probabilites. For SIR-graphs this means that the network may have individal edge probabilies. For the SIS-graph, this means that network may have both indiviual edge and vertex probabilites.

In Chapter 6 we try to classify how hard the it is to compute the probability for SIR and SIS-networks. My contribution in this chapter is in the first section filling out the proofs from the sources and combining it to a complete proof. In the second section I have contributed the completed proof, showing that that the SIS problem also is #P-hard.

In Chapter 7 we create two algorithms for computing the probability, an exact algorithm and one Monte-Carlo algorithm. Chapter 8 we do some testruns using the algorithms we made. All of this is my contribution.

We conclude this with Chapter 9 summarizing the theories that have been done in this thesis, and telling about further work.

CHAPTER 2

Background theory

We need some background theory before we can start working with epidemic models. In the first section we will start with an introduction to graph theory, using theories from Bondy and Murty [**2**]. We will start with defining a digraph, and recall some central definitions in graph theory, and go through some basic algorithms. In the second section we introduce probability theory in the language of measure theory, Monte-Carlo simulations and Markov chains. This section is based on the books of Teschl [**15**], Grinstad and Scnell [**7**],Stein and Shakarchi [**14**], Nualart[**12**], and lecture notes by Ieyengard [**8**]. In the third section we talk about time-complexity of problems. We first talk about the Big O-notatation and the NP class taken from the book of Cormen [**4**]. Then we introduce the #P-class which is based on the article of Valiant [**13**].

1. Graph theory

1.1. Introduction. Graph theory is the study of the properties of mathematical structures of graphs, which consists of points, and lines connecting together two points. Let us define the vertices as the points, and the edges as the lines. There are many situations in real life which can be represented by graphs. In social medias, we can let the every person be represented by a vertice, and let there be an edge between to vertices if there is a friendship between them. How many friends does two people have in common? Given two person A and B, what is the shortest sequence that begins with person A, ends with person B and two consequtive vertices is consequtive if and only if they are friends? All these questions can be answered by well known algorithms in graph theory. We will only work with digraphs in this master, and start by defining it.

DEFINITION 2.1. *A digraph G is a ordered pair $G = (V, E)$ consisting of*

(i) the set V of vertices

(ii) the set E of edges (disjoint from V)

(iii) the injective function $\psi : E \to V \times V$ (ordered)

We can represent a digraph graphically by letting the vertices be points is the plane, and draw an arrow from u pointing at v if there is an edge $e \in E$. If e is an edge and u and v are vertices such that $\psi(e) = (u, v)$, e is said to join u and v, and may call the edge $e_{u,v}$. Then u is called the source, and v is called the sink. An vertex w is a a predecessor of v, if there exists an edge $e_{w,v}$, and a successor of v, if there exists an edge $e_{v,w}$.

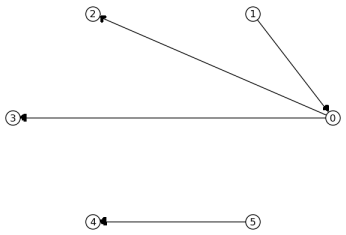

FIGURE 1. Example of a digraph

The ψ function for representing the edges in a graph is not convenient to work with either in computer science or mathematics for studying their properties. Therefore we rather use the adjancy matrix or adjancy list to represent the graph. These datastructures can easily be represented on the computer.

DEFINITION 2.2. *The adjacency matrix of a graph G is the $n \times n$ matrix $A = (a_{uv})$. a_{uv} is the number of edges joining uv, loop counting as two.*

A path is graph arranged in a linear sequence such that two vertices are adjacent if and only if they are consecutive in the sequence. A cycle is a simple graph on three vertices or more arranged in a cyclic sequence such that two vertices are adjacent if and only if they are consecutive in the sequence. A walk is a sequence of vertices and edges such that the vertices and edges are adjacent typical $v_0e_1v_1...v_{k-1}e_kv_k$.

In a digraph we look at the strongly connected components. These are the components with the property that u and v is in the same component iff there exists a walk from u to v, and a walk from v to u.

1.2. Basic search algorithm. Let us number the vertices from 1 to n. Breadth-First-Search and Depth-first search is a tool we often use to solve graph-related problems. They are algorithms that traverse the graph like a tree.

In Depth-First-Search, shortened to DFS, we start with an empty stack. Whenever a vertex is added to the stack, it is also added to the tree. We start by adding the start vertex to the stack. At each stage, we take the top vertex in the stack, search for the first successor, and add it to the stack. If all the successors are visited, we remove the vertex from the stack. This result in the following code.

```
1  input: a graph G and start s
2  output: a rooted spanning tree of G
3
4  visited = [false...]
5  parent = [-1 ... ]
6  stack = []
7  stack.push(s)
8  visited[r] = true
9  while len(stack)>0:
10     x = stack.top()
11     if x has unvisited neighbour y:
```

```
12          visited[y] = true
13          p(y)=x
14          stack.push(y)
15      else:
16          stack.pop()
17
18  return parent
```

In Breadth-First-Search, shortened to BFS, we start with an empty queue. Whenenver a vertex is added to the queue, it is also added to the tree. We start by adding the start vertex to the stack. At each stage, we take the first vertex v in the queue, search for all the unvisited successors, add them to the queue, and remove vertex v from the queue. This result in the following code.

```
1   input: a graph G and start s
2   output: a rooted spanning tree of G
3
4   visited = [false...]
5   parent = [-1 ... ]
6   queue = []
7   queue.push(s)
8   visited[r] = true
9   while len(stack)>0:
10      x = stack.top()
11      for every unvisited. neighbour y:
12          visited[y] = true
13          p(y)=x
14          stack.push(y)
15      else:
16          stack.pop()
17
18  return parent
```

2. Probability theory

We will in this chapter recall some important theories of random variables and specially Markov chain.

2.1. Introduction. A probability space is denoted by (Ω, Σ, P) and is defined as follows

DEFINITION 2.3. *(Ω, Σ, P) is a probability space iff*
Ω is the sample space denoting all possible outcomes.

Σ is the family of setsets such that

(i) $\emptyset) \in \Sigma$

(ii) If $A \in \Sigma$, then its complement is also in Σ

(iii) If $A_1, ... \in \Sigma$, then $\cup_{i=1}^{\infty} A_i \in \Sigma$

$P : \Sigma \to [0, 1]$ is a measure-function if it satisfies three conditions:

(i) $P(\emptyset) = 0$

(ii) $P(\cup_{i=1}^{\infty} A_j) = \cup_{i=1}^{\infty} P(A_j)$ *if* A_j *is pairwise disjoint.*

(iii) $P(\Omega) = 1$

We also need to define a random variable.

DEFINITION 2.4. *X is a random variable iff*
$X : \Omega \to \mathbb{F}$, where $\mathbb{F}$ is a field and is Σ-measurable, that is $X^{-1}(B) \in \Sigma$, where B is any Borel set in $\mathbb{F}$.

For random variables giving discrete values, we often describe the variable by a probability density function $f_X : \mathbb{F} \to [0, 1]$ such that

$$f_X(x) = P(\{\omega \in \Omega : X(\omega) = x\}).$$

Throughout this paper, we will mainly work with random variables that are discrete and have finite outcome.

We want to find out the probability that an event E occurs. Examples of events could be $E = \{\omega \in \Omega : X(\omega) = 1\}$ or $E = \{\omega \in \Omega : X(\omega) > 10\}$. In general we get $E = \{\omega \in \Omega : \text{ condition }\}$. We also sometimes use the shortform by only write the condition of E instead of E itself. Such that we write $P(\text{condition}) = P(\{\omega \in \Omega : \text{ condition }\})$.

We now need to define integrals on a probability space (Ω, Σ, P) for discrete variables.

$$\int_A sdP = \sum_{x \in A} s(x)P(x)$$

Then the integral has the following properties (taken from Gerald Teschl [**15**])

THEOREM 2.1. *These properties hold*

(i) $\int_A sdP = \int_\Omega \chi_A sdP$.

(ii) $\int_{\cup_{j=1}^{\infty} A_j} sdP = \sum_{j=1}^{\infty} \int_{A_j} dP$ *when* $A_j \cap A_k = \emptyset$ *for* $j \neq k$.

(iii) $\int_A \alpha sdP = \alpha \int_A sdP, \alpha \in \mathbb{R}_+$.

(iv) $\int_A (s + t)dP = \int_A sdP + \int_A tdP$

(v) $A \subset B \to \int_A sdP \leq \int_B sdP$

(vi) $s \leq t \to \int_A sdP \leq \int_A tdP$

The expected value $E[X]$ and variance $Var[X]$ is defined by

$$E[X] = \int_\Omega XdP$$

$$Var[X] = E[(X - E[X])^2] = \int_\Omega (X - E[X])^2 dP$$

Conditional probability is the probability that an event occur, given that another event has already occured and is defined the following way.

DEFINITION 2.5. *(Conditional probability) Given events A and B. The conditional probability for A occuring given B is.*

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

as long as $P(B)$ is not zero.

Markov's inequality theorem is a theorem about the relation between the expected value and probability (taken from Stein and Shakarchi [**14**]).

THEOREM 2.2. *Let X be a Σ-measurable function. Then for any $a > 0$*

$$P(\{\omega \in \Omega | X(\omega) \geq a\}) \leq \frac{E[X]}{a}$$

PROOF. Choose an $a > 0$ and let $A = \{\omega \in \Omega : |X| \geq a\}$. Let χ_A be the indicator function of A. Then we see that

$$a\chi_A(\omega) = \begin{cases} a & \text{for } \omega \in A \\ 0 & \text{else} \end{cases}$$

Thus $a\chi_A \leq |f(\omega)|$ for all $\omega \in \Omega$. By Theorem 2.1(iii), we have that

$$\int_\Omega a\chi_A dP = a \int_\Omega \chi_A dP \tag{2.1}$$

and by Theorem 2.1 -(v)

$$\int_\Omega a\chi_A dP \leq \int_\Omega |X| dP \tag{2.2}$$

Hence by definition of $E[|X|]$ and by applying Equation 2.1 and 2.2 we have

$$\frac{1}{a}E[|X|] = \frac{1}{a}\int_\Omega |X| dP \geq \int_\Omega \chi_A dP = \int_A dP$$

The result follows. □

2.2. Monte Carlo simulation. Often in probability theory, we need to simulate the different random variables with the corresponding probability distrubution. A method for doing this is called the inverse transfomation method. We will cover the discrete case. We first start with a definition.

DEFINITION 2.6. *Let X be a random variable, and let F_X be its cumulative distrubution function. The inverse is defined by*

$$F_X^{-1}(x) = \min\{y : F_x(y) \geq x\}$$

Then we can create a generator with this cumulative distrubution from a uniform distribution by the following theorem (theorem from lecture note of Ieyengard [**8**])

THEOREM 2.3. *Inverse transform sampling for one variable*
Let U be the uniform distrubution $[0, 1]$ and let X be a discrete random variable having the probability distrubution $P(X = x_i) = p_i$ for $i = 1, ..., m$. Then $X = F_X^{-1}(U)$.

PROOF. We will show that X and $F_X^{-1}(U)$ has the same distrubution.

$$\begin{aligned} P(F_X^{-1}(U) = x_j) &= P(\min\{y : F_X(y) \geq U\} = x_j) \\ &= P(\sum_{i=0}^{j-1} p_i < U < \sum_{i=0}^{j} p_i) \\ &= p_j \end{aligned}$$

□

The algorithm proposed by the theorem is quite simple. First generate a random number u between 0 and 1. Then the corresponding random variable X is $\min\{y : F_x(y) \geq u\}$.

Let Y be composed of several discrete random variables $X_1, ..., X_n$ such that $Y = f(X_1, X_2, .., X_k)$. and $f : \mathbb{F}^m \to \mathbb{F}^n$. For generating a random variable Y, this method proposes that we can use the inverse transform sampling on each of the random variables. Thus $Y = f(F_{X_1}^{-1}(U_1), F_{X_2}^{-1}(U_2), .., F_{X_k}^{-1}(U_k))$.

The Law of Large Numbers is a result telling us that when an experience is repeated a large number of times, the average will come close to the actual value. To prove this, we first need an inequality called the Chebyshev Inequality. (Both theorems are taken from Grinstead and Schnell [**7**]).

THEOREM 2.4. *Let X be a discrete random variable with expected value λ and variance σ^2. Then*

$$P(|X - \lambda| \geq \epsilon) \leq \frac{\sigma^2}{\epsilon^2}$$

COROLLARY 2.1. *(The Law of Large Numbers)*
Let $X_1, ..., X_n$ be independent random variables with expected value λ and variance σ^2. and let $S_n = \frac{1}{n}(X_1 + ... + X_n)$. Then

$$P(|S_n - \mu| \geq \epsilon) \leq \frac{\sigma^2}{n\epsilon^2}$$

PROOF. This is a very simple proof where we first by Theorem 2.1(iv) finds that that $E(S_n) = \lambda$ and $V(S_n) = \frac{1}{n}\sigma^2$. Then by applying Chebyshev's inequality, we are getting the wanted result. □

Monte Carlo method is a collecting term for getting a numeric result by running the experience a large number of times. By inverse transform sampling and law og large numbers, this proposes a method for doing a Monte Carlo simulation to find the expected value for any random variable.

2.3. Discrete markov chain. A discrete Markov chain is a stochastic process or a series of random variables $(X_t)_{t\in\mathbb{N}}$ where $X_t : \Omega \to \mathbb{F}$ describing the state at the time t. A Markov chain has the timeless-property that says X_t is only dependent on X_{t-1} and not on earlier events $X_{t-2}, ...X_0$. An example of this is the roulette, where earlier results do not affects the probabilty of the current game. X_t could represent the amount of money you have at time t.

Let I be the state space, the set with all the possible values X_t may have. In this paper we will primarily look at discrete, time-homogenous Markov chains with finite state space.

DEFINITION 2.7. *Discrete time-homogenous Markov chain with finite state space. A stochastic process $(X_t)_{t\in\mathbb{N}}$ is a Markov chain iff*

(i) $P(X_{n+1} = i_{n+1}|X_0 = i_0, ..., X_n = i_n) = P(X_{n+1} = i_{n+1}|X_n = i_n)$

(ii) $P(X_{n+1} = x|X_n = i) = P(X_n = x|X_{n-1} = i)$

for all $t \in \mathbb{N}$ and $i_0, i_1, i_2, ..., i_{n+1} \in I$.

The property (i) is often called the Markov property, and states the independence of earlier events.

X_t is the called the state of the Markov chain at time t. A state i is called an absorbing state if it is impossible to leave the state, thus if $P(X_{m+n} = i|X_m = i) = 1$ and $P(_{m+n}= j|X_m = i) = 0$ for all $m, n \in \mathbb{N}$.

An important result for the Markov chain is the Chapman-Kolmogorov equation (taken from Grinstead and Schnell [**7**]).

THEOREM 2.5. *(Chapman-Kolmogorov equation)*
Given a Markov chain $(X_t)_{t\in\mathbb{N}}$. Then for any $n, m \geq 0$ and $i, j \in I$, we have

$$P(X_{n+m} = i|X_0 = j) = \sum_{k\in I} P(X_{n+m} = i|X_n = k)P(X_n = k|X_0 = j)$$

PROOF. The first thing we do is observing that $X_1, ..., X_n$ has only a finite number of possibilites. We can then apply Theorem 2.1(ii), inspecting all the possibilites for $X_n = k$.

$$\begin{aligned} P(X_{n+m} = i|X_0 = j) &= P(\{\omega \in \Omega : X_{n+m}(\omega) = j\}|X_0 = i) \\ &= \sum_{k\in S} P(\{\omega \in \Omega : X_{n+m}(\omega) = j, X_n(\omega) = k\}|X_0 = i) \end{aligned}$$

Then

$$\begin{aligned} &= \sum_{k\in S} \frac{P(X_{n+m} = j, X_n = k, X_0 = i)}{P(X_0 = i)} \\ &= \sum_{k\in S} \frac{P(X_{n+m} = j|X_n = k, X_0 = i)P(X_n = k, X_0 = i)}{P(X_0 = i)} \\ &= \sum_{k\in S} P_{k,j}^{(m)} \frac{P(X_n = k, X_0 = i)}{P(X_0 = i)} \\ &= \sum_{k\in I} P_{i,k}^{(n)} P_{k,j}^{(m)} \end{aligned}$$

The first line is by property (ii) of a measure function. In the second, third and fourth line we are using Definition 2.5(Conditional probability). □

An observation is that given a Markov chain $(X_t)_{t\in\mathbb{N}}$ with initial distrubution π and transition matrix P, we can also measure the probability that a given sequence will happen.

THEOREM 2.6. *Given states $i_0, ..., i_n \in I$.*

$$P(X_0 = i_0, ..., X_n = i_n) = \pi(i_0) \prod_{j=1}^{n} P(X_j = i_j | X_{j-1} = i_{j-1})$$

PROOF. This can be shown by induction. For $n = 0$, this is clear by the definition of Markov chain. For the inductive step, we apply the definition of conditional probability and get.

$$\begin{aligned} &P(X_0 = i_0, ..., X_n = i_n) \\ &\quad = P(X_n = i_n | X_{n-1} = i_{n-1}, ..., X_0 = i_n) P(X_{n-1} = i_{n-1}, ..., X_0 = i_n) \end{aligned}$$

Then we use the Markov property and get

$$= P(X_n = i_n | X_{n-1}) P(X_{n-1} = i_{n-1}, ..., X_0 = i_n)$$

The induction hypothesis completes the proof

$$\begin{aligned} &= P(X_n = i_n | X_{n-1}) \pi(i_0) \prod_{j=1}^{n-1} P_{i_{j-1}, i_j} \\ &= \pi(i_0) \prod_{j=1}^{n} P(X_j = i_j | X_{j-1} = i_{j-1}) \end{aligned}$$

□

3. Classifying problems by timecomplexity

A question we often ask is whether there is an efficient algorithm which solves the specified problem, and its running time. The Big O notation is often used, specially in computer science and optimization theory, to tell the upper bound asymptotic behaviour of the input size. The Big O notation has the following defintion:

DEFINITION 2.8. *The Big O notation*
If $f(n) = O(g(n))$, then there exists constants $c, N > 0$ such that for all $n \geq N, f(n) \leq cg(n)$.

When we describe a polynomial we will only use the term with highest degree, and drop the constant such that $a_n x^n + ... + a_0 = O(x^n)$. Often we have multiple input variables, but the O notation still make sence, by letting $f, g : \mathbb{R}^n \to \mathbb{R}$. For example, the Breath-First-Search and Depth-First-Search algorithm which as we have described has the running time $O(|V| + |E|)$, where V, E and r is the input.

We can classify algorithms according to their running time and properties. If we have problems where there exists an polynomial algorithm solving it en we says the problem is in P. The class of NP problems consists of the problems where given an output of the problem, it can be verified in polynomial time if it satisfies the

constrains of the problems. An example of a such problem is the SAT problem. In the SAT problem you are given a set $X = \{x_1, ..., x_n\}$, where each element is assigned either true or false We are given clauses $c_1, c_2, ..., c_n$ where $c_i = (y_{i1} \vee y_{i2} ... \vee y_{ik_i})$ where $y_{ij} \in X$. The question is if we can find a assignment $X \to \{T, F\}$ such that $F = c_1 \wedge c_2 ... \wedge c_r$ is true. To verify if the assignment of X is a solution or not takes polynomial time.

Valient suggested a new class called the #P problems. In this class, a solution can be verified in polynomial time if it satisfies the constrains of the problem. But instead of finding a solution, we try to find the number of solutions that satisfies the constrains of the problem. We define #PC or #P-complete problems to be the problems that is at least as hard as any other problems in #P. #P-hard problems, is the problem that not necessarily is in #P, but is at least as hard as the problems that is #PC. We do not know if there are any polynomial solution for problems in #PC, but if there is, this means that all the other problems in #PC can be solve in polynomial time. Throughout the thesis, we will assume that #P≠P, ie. there is no polynomial algorithm solving the problems in #PC.

DEFINITION 2.9. *A problem $p \in \#P$ is #PC if every problem in #P can be tranformed to p in polynomial time.*

DEFINITION 2.10. *A problem p is #P-hard if every problem in #P can be tranformed to p in polynomial time.*

Some problems have a polynomial algorithm when we are trying to find find a solution, but is #PC when we need to count the total number of solutions. While the 2SAT problem in NP has a polynomial solution, the corresponding problem called monotone 2SAT-problem is #PC (See [**16**] for proof).

DEFINITION 2.11. *Monotone 2-SAT*
We are given clauses $c_1, c_2, ..., c_n$ where $c_i = (y_{i,1} \vee y_{i,2})$ where $y_{ij} \in X$. X are the set consisting of the boolean variables $x_1, ..., x_n$. How many ways can we assign each variables of X such that $F = c_1 \wedge c_2 ... \wedge c_n$ is true.

THEOREM 2.7. *The monotone 2SAT problem is in #PC.*

We will be using this as a starting point for finding other problems in #PC. The procedure we will use to show that a problem p is in #PC is following. First, we show that given a output of the problem, we can verify at polynomial time if it satisfies the constraints. The last step is to take a problem p in #PC, and show that p can be reduced to q in polynomial time. This is often done by assuming that p can be solved in polynomial time, and deduce that then a problem $q \in$#PC can be solved in polynomial time, which is absurdum since $\#P \neq P$.

The two following theorems by Valiant are useful for showing that a problem is in #PC and will be used later in this master (see [**16**] for proof).

THEOREM 2.8. *If p(x) is an n-th degree polynomial and its value is known at rational points $x_1, ..., x_n + 1$ all of size at most m, the coefficients of p can be deduced in time polynomial in n, m and the size of the largest value.*

THEOREM 2.9. *Let $\{a_i\}$ and $\{b_i\}$ be sets of positive integers bounded by $A > 2$. If the one of the following function is known for a point x or (x, y), then the value of each a_i and b_i is deduced in polynomial time in n, m, x_0, y_0 and its value.*

(i) $\sum_0^n a_i x^i$ *if* $x_0 \geq A^2$ *or* $0 < x \leq A^{-2}$

(ii) $\sum_0^n a_i (1-x)^{n-i}$ *if* $0 < x \leq A^{-2}$

(iii) $\sum_{i=0}^n \sum_{j=0}^m b_{ij} x^i (1-x)^{n-i} y^i (1-y)^{m-i}$ *if* $x = A^{-2}$ *and* $0 < y < A^{-3n}$

CHAPTER 3

Models for epidemics

Given a digraph $G = (V, E)$, an infected vertex u can only infect a another vertex v if there is an edge from u to v. For example, the graph could represent a computer network, where the vertices are the computers, and there is an edge between two computers if they have iteracted, and the the infection could be a computer virus. Or maybe we are trying to investigate a HIV-epidemic, and letting the graph be persons with edges between to persons if they have had intercourse.

Over the years, many models have been developed for this purpose, and we will mainly focus on the SIR network as described in [**5**][**6**], and the SIS network as described in [**3**]. In the first section, we will describe the SIR network, and in the second section we will describe the SIS network in the language of probability theory. The goal is to connect these models to the theory of probability and Markov chains.

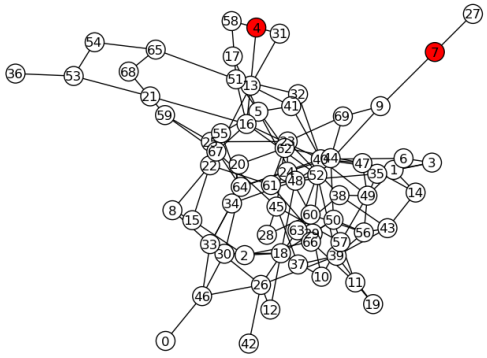

FIGURE 1. Example of an graph modeling an epidemic, where the red vertices are the infected, and the white vertices are not infected

1. SIR network

The SIR network is the simplest of the epidemic models we are going to work with. This graph structures is used to model diseases with the property that the receiver only gets the infection once and either dies or get immune. Then the

disease c Turbeculosis is an example of such an infection, which is an infectious disease caused by Mycobacterium tuberculosis, spreads though air for example by sneeze and spits.

An instance of the SIR network is given by $N = (G, \mu)$, where G is the underlying digraph, and $\mu : E \to [0, 1]$ is a function such that $\mu(e)$ is the probability that an edge transmit infection. We will make an important assumption in this model, which is the independence of edges.

We define a state ϕ as a function such that $\phi : V \to \{S, I, R\}$. A vertex is labeled S if it is suspected, I if it is infected, and R if it is resistant. Only suspected vertices can receive infection, and it receive it from an infected predecessor. For infected vertices, it becomes resistance in the next timestep. We say than an edge $e_{x,y}$ is in play in state ϕ, if $\phi(x) = I$ and $\phi(y) = S$, and we also says that the vertex y is in play at that state. The set of all edges in play at state ϕ we denote by P_ϕ, and the set of all edges in play at state ϕ with sink v we denote by $P_\phi(v)$.

DEFINITION 3.1. *Given the states ϕ_1 and ϕ_2, ϕ_2 is a possible successor of ϕ_1 if it satisfies the following conditions:*

(i) If $\phi_1(v) = R$, then $\phi_2(v) = R$

(ii) If $\phi_1(v) = I$, then $\phi_2(v) = R$

(iii) If $\phi_1(v) = S$, then $\phi_2(v) \in \{S, I\}$

(iv) If $\phi_2(v) = I$, then v is in play in ϕ_1

for every $v \in V$.

Denote $\Psi_{t,N} : V \to \{S, I, R\}$ to be the state at timestep t in network N. If it is clear which network we are working with, we only use the notation Ψ_t. Let $\Psi_t = \phi$, where ϕ is a state. We may think the edges e in play as random variables with probability

$$P(X_e = x) = \begin{cases} \mu(e) & \text{if } x = 1 \\ (1 - \mu(e)) & \text{if } x = 0 \end{cases}$$

where 1 denotes that X_e transmit infection, and 0 if it does not transmit infection. Then Ψ_{t+1} is determined by the edge random variables in play, and Ψ_t.

The probability that an vertex in play v becomes infected at the next timestep

$$\begin{aligned} P(\Psi_{t+1}(v) = I | \Psi_t = \phi) &= 1 - \prod_{\{e \in P_{\phi(v)}\}} P(X_e = 0) \\ &= 1 - \prod_{\{e \in P_{\phi(v)}\}} (1 - \mu(e)) \end{aligned} \tag{3.1}$$

. This induces the following probability for all states ϕ, ψ.

$$P(\Psi_{t+1} = \psi | \Psi_t = \phi) = \begin{cases} A & \text{if } \psi \text{ successor of } \phi \\ 0 & \text{else} \end{cases}$$

$$A = \prod_{\{v \in P_\phi | \psi(v) = I\}} P(\Psi_{t+1}(v) = I | \Psi_t = \phi) \prod_{\{v \in P_\phi | \psi(v) = S\}} (1 - P(\Psi_{t+1}(v) = I | \Psi_t = \phi))$$

The stochastic process $\Psi_0, ..., \Psi_t$ is a discrete time-homogenious Markov chain. By applying Theorem 2.6 we can measure the following probability for all series of states $\phi_0, ..., \phi_n$.

$$P(\Psi_0, .., \Psi_n = \phi_0, ..., \phi_n | \Psi_0 = \phi_0) = \prod_{i=1}^{n} P(\Psi_n = \phi_i | \Psi_{i-1} = \phi_{i-1}) \tag{3.2}$$

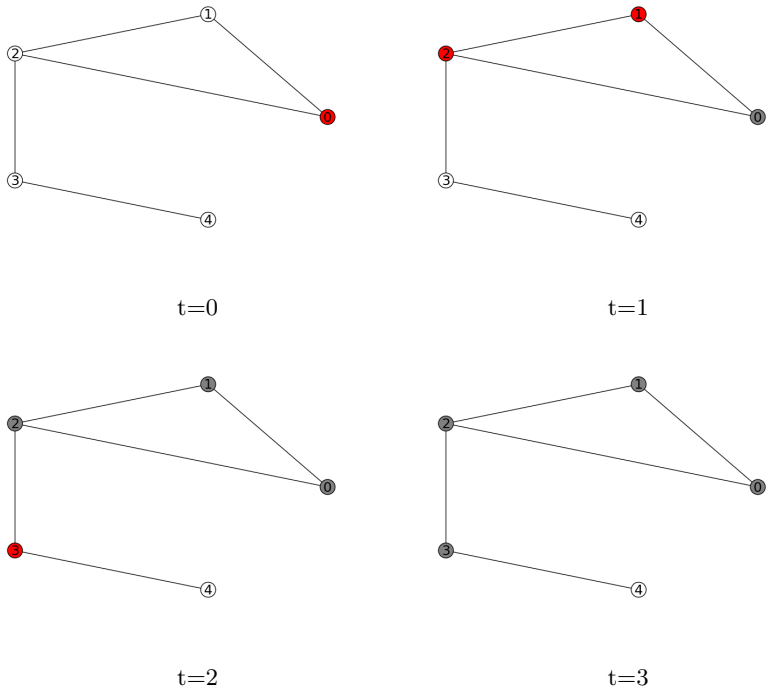

FIGURE 2. An example of a possible disease through a SIR network. The blank vertices are suspectable(S), the red vertices are infected(I), and the gray vertices are resistant(R).

In Figure 2, we have an example of an possible development of states $\phi_0, ..., \phi_3$. where

$$\phi_0(v) = \begin{cases} I & \text{if } v = 0 \\ S & \text{else} \end{cases}$$

$$\phi_1(v) = \begin{cases} I & \text{if } v \in \{1, 2\} \\ R & \text{if } v = 0 \\ S & \text{else} \end{cases}$$

$$\phi_2(v) = \begin{cases} I & \text{if } v = 3 \\ R & \text{if } v \in \{0, 1, 2\} \\ S & \text{else} \end{cases}$$

$$\phi_3(v) = \begin{cases} R & \text{if } v \in \{0, 1, 2, 3\} \\ S & \text{else} \end{cases}$$

The absorbing states are all ϕ such that for all $v \in V$ $\phi(v) = \{S, R\}$.

Let us at last define a partial order for the SIR-graphs and their states. These definitions will be used later.

DEFINITION 3.2. *Let ϕ and ψ be two states in a SIR network.*
We define the partial order $\phi \subseteq \psi$ iff $\phi^{-1}(I) \subseteq \psi^{-1}(I)$.

DEFINITION 3.3. *Let N and N' be two SIR networks. We define the partial order $N \leq N'$ iff $\mu_N(e) \leq \mu_{N'}(e)$ for all edges e.*

2. SIS network

The SIS-network is derived from the SIR-network, with a little difference: The infected vertices in the graph will never be removed, and has only the possibility in the next timestep to stay infected or get cured. An example of such a disease is the common coat, which is a viral infectious disease, caused most commonly by the rhinovirus. The disease usually lasts for one to or weeks, but the person will after curation not receive immunity, and will therefore risk to be infected again.

An instance of the SIS network is given by $N = (G, \mu, \rho)$ where G is the underlying digraph, $\mu : E \to [0, 1]$ is a function such that $\mu(e)$ is the probability that the edge e transmit infection and $\rho : V \to [0, 1]$ is a function such that $\rho(v)$ is the probability that the vertex v which is infected gets cured. We make an assumption that both transmission through edges and cureness in vertices happens independently.

In a SIS network, we define a state to be a function $\phi : V \to \{S, I\}$. A vertex is labeled S if its is suspected, and I if it is infected. An infected vertex in this model can either stay infected, get healed, or get healed and imediately recieve infection from a predecessor in the same timestep. Suspected vertices can either stay suspected or receive infection from one of its predecessors. We say that an edge $e_{x,y}$ is in play in state ϕ, if $\phi(x) = I$ and $\phi(y) = S$. and we also says that the vertex y is in play at that state. We say also the vertex y is also in play if $\phi(x) = I$. The set of all edges in play at state ϕ we denote by P_ϕ, and the set of all edges in play at state ϕ with sink v we denote by $P_\phi(v)$.

DEFINITION 3.4. *Given the states ϕ_1 and ϕ_2, ϕ_2 is a possible successor of ϕ_1 if it satisfies the following condition:*

(i) If $\phi_2(x) = I$, then x was in play in ϕ_1

Denote $\Upsilon_{t,N} : V \to \{S, I\}$ to be the state at timestep t in network N. If it is clear which network we are working with, we only use the notation Υ_t. Let ϕ by any state in $St(G)$. We may think of the edges e in play as random variables with probability

$$P(X_e = x) = \begin{cases} \mu(e) & \text{if } x = 1 \\ (1 - \mu(e)) & \text{if } x = 0 \end{cases}$$

where 1 denotes that e transmit infection, and 0 if it does not transmit infection. Similar we may think of the vertices v in play as random variables with probability

$$P(X_v = x) = \begin{cases} \rho(e) & \text{if } x = 1 \\ (1 - \rho(e)) & \text{if } x = 0 \end{cases}$$

where 1 denotes that v get cured, and 0 if it does not get cured. Then Υ_{t+1} is determined by the edge and vertex random variables in play, and Υ_t.

The probability that a vertex in play v gets infected at the next timestep

$$\begin{aligned} P(\Upsilon_{t+1}(v) = I | \Upsilon_t = \phi) \\ &= 1 - (\chi_S(\phi(v)) + P(X_v = 1)\chi_I(\phi(v))) \prod_{\{e_{x,y} \in P_\phi(E)\}} P(X_e = 1) \\ &= 1 - (\chi_S(\phi(y)) + \rho(y)\chi_I(\phi(y))) \prod_{\{(x,y) \in P_\phi(E)\}} (1 - \mu(e)) \end{aligned} \tag{3.3}$$

where $\chi_I(x)$ is the identity function for I return 1 if $x = I$ and 0 else, and χ_S is the identity function for S. This induces the probability

$$P(\Upsilon_{t+1} = \psi | \Upsilon_t = \phi) = \begin{cases} A & \text{if } \psi \text{ successor of } \phi \\ 0 & \text{else} \end{cases}$$

$$A = \prod_{\{v \in P_\phi | \psi(v) = I\}} P(\Upsilon_{t+1}(v) = I | \Upsilon_t = \phi) \prod_{\{v \in P_\phi | \psi(v) = S\}} (1 - P(\Upsilon_{t+1}(v) = I | \Upsilon_t = \phi))$$

for all states ϕ, ψ.

The stochastic process $\Upsilon_0, ..., \Upsilon_t$ is a discrete time-homogenious Markov chain. As for the case of the SIR-networks by applying Theorem 2.6, given a series of states $\phi_0, ..., \phi_n$, we can measure the probability of such an series by

$$P(\Upsilon_0, .., \Upsilon_n = \phi_0, ..., \phi_n | \Upsilon_0 = \phi_0) = \prod_{i=1}^{n} P(\Upsilon_n = \phi_i | \Upsilon_{i-1} = \phi_{i-1}) \tag{3.4}$$

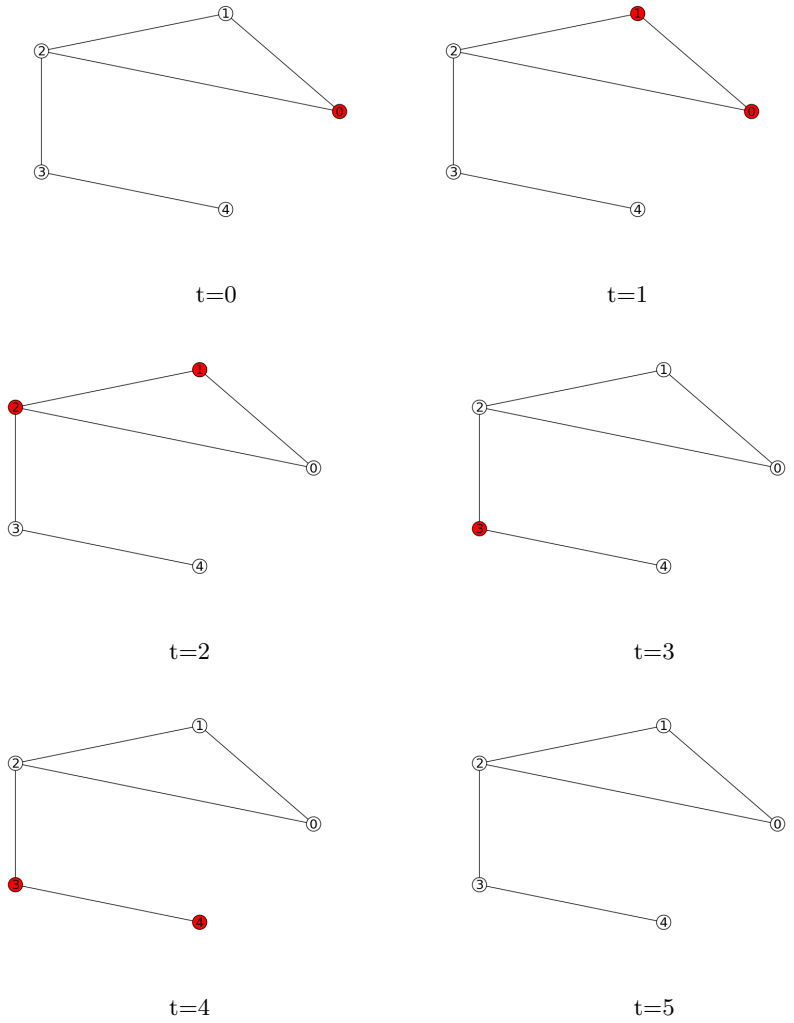

FIGURE 3. An example of a development of disease through an SIS network. In a state, the white nodes are the suspected (S), and the red nodes the the infected (I).

In Figure 3 we can see an example of such a sequence $\phi_0, ..., \phi_5$.

$$\phi_0(v) = \begin{cases} I & \text{if } v = 0 \\ S & \text{else} \end{cases}$$

$$\phi_1(v) = \begin{cases} I & \text{if } v \in \{0, 1\} \\ S & \text{else} \end{cases}$$

$$\phi_2(v) = \begin{cases} I & \text{if } v \in \{1, 2\} \\ S & \text{else} \end{cases}$$

$$\phi_3(v) = \begin{cases} I & \text{if } v = 3 \\ S & \text{else} \end{cases}$$

$$\phi_4(v) = \begin{cases} I & \text{if } v \in \{3, 4\} \\ S & \text{else} \end{cases}$$

$$\phi_5(v) = S$$

The only absorbing state here is the state where $\phi(v) = S$ for all v. In some cases there can tak a long time before it reaches this state, and in some cases it will never reach it.

Let us also define a partial order for the SIS-graphs and their states, similar to what we did for the SIR-networks.

DEFINITION 3.5. *Let ϕ and ψ be two states in a SIS network.*
We define the partial order $\phi \subseteq \psi$ iff $\phi^{-1}(I) \subseteq \psi^{-1}(I)$.

DEFINITION 3.6. *Let N and N' be two SIS networks. We define the partial order $N \leq N'$ iff $\mu_N(e) \leq \mu_{N'}(e)$ for all edges e and $\rho_N(v) \geq \rho_{N'}(v)$ for all vertices v.*

CHAPTER 4

The monotonicity properties

Given a SIR or SIS network, we want to find out how adding some vertices or removing from the initial state will affect the probability that a particulary person gets infected. Another question we ask is that if lower the edge transmisssion rates between some vertices will affect the probabiliies for infection. The third question is specifically for SIS networks, and is what will happen for infection if we can increase the probability of a vertex being cured after infection. The monotonicity properties tells us about how the probabilities of $(\Psi_t)_{t\in\mathbb{N}}$ will evolve by changing the initiale state ϕ, edge transmission probability $\mu(e)$ for some e, or vertex cureness probability $\rho(e)$.

In the first section, we will describe a proof by Floyd, Lelie Kay and Shapiro [**6**] showing the monotonicity properties of SIR networks. In the second section, We will prove that the monotonicity properties also holds for a SIS networks by using some of the concepts from the first section. In the last section, we will make an alternativ proof for showing that the monotonicity properties holds for SIS networks by using the idea of creating a covering graph. (See [**6**] for an different case where the covering graph have been used).

1. A proof for the monotonicity properties for SIR networks

THEOREM 4.1. *Monotonicity properties for SIR networks*

(i) Given start states ϕ, ψ with $\phi \leq \psi$ in network N, then for any vertice v and $t \geq 0$ we have $P(\Psi_t(v) \in \{I, R\} | \Psi_0 = \phi) \leq P(\Psi_t(v) \in \{I, R\} | \Psi_0 = \psi)$

(ii) Given networks N_0, N_1 with $N_0 < N_1$, then for any initial state ϕ, vertice v and $n \geq 0$ we have $P(\Psi_{t,N_0} \in \{I, R\} | \Psi_{0,N_0} = \phi) \leq P(\Psi_{t,N_1} \in \{I, R\} | \Psi_{0,N_1} = \phi)$

The first observation we make is that given an edge $e = (x, y)$ and chain of possible sucessors $\phi_0, ..., \phi_k$, there is at most one state ϕ_j such that e is in play. This can easily be proved. When e is in play in ϕ_j, this means $\phi_j(x) = I$ and $\phi_j(y) = S$. Then for all $i < j$, $\phi_i(x) = S$ and for all $i > j$, $\phi_i(x) = R$. Thus ϕ_j is the only state where e can be in play. Let $X_E = (X_e)_{e\in E}$ where X_e is independent Bernoulli random variables giving 1 with probability $\mu(e)$ and 0 iwth probability $(1 - \mu(e))$. We may look at X_E as a discrete random variable, and may measure the probability that X_E hits $\zeta \in \{0, 1\}^E$ by

$$P(X_E = \zeta) = \prod_{\{e|\zeta(e)=1\}} \mu(e) \prod_{\{e|\zeta(e)=0\}} (1 - \mu(e))$$

Let us create a map $\epsilon_\zeta : \{S, I, R\}^V \to \{S, I, R\}^V$ where

$$\epsilon(\zeta, \phi)(v) = \begin{cases} R & \text{if } \phi(v) = R \\ R & \text{if } \phi(v) = I \\ I & \text{if there is } e = (x, y) \text{ where } e \text{ is in play and } \zeta(e) = 1 \\ S & \text{otherwise} \end{cases}$$

for any $\zeta \in \{0, 1\}^E$ and $\phi \in St(G)$.

The following theorem tells us about the relationship between X_E and $(\phi_t)_{t\in\mathbb{Z}_+}$.

THEOREM 4.2. *Properties of the map $\epsilon_\zeta : St(G) \to St(G)$*

(i) Given state ϕ and $\zeta \in \{0, 1\}^{|E|}$, the sequence $(\epsilon_\zeta^t(\phi))_{t\in\mathbb{N}}$ is possible series of successors

(ii) Every series of successors arises in this manner

(iii) Given states ζ, ψ and $\zeta \in \{0, 1\}^{|E|}$,
$P(\Psi_t = \psi | \Psi_0 = \phi) = P(X_e \in \{\zeta | \epsilon_\zeta^n(\phi) = \psi\})$

PROOF. (i) Only the vertices in play can be infected, and the infected will be removed in the sucessor, and the removed stay removed. By doing this the inductive step, this shows that every consecutive states are a successor.

(ii) is shown by induction. An edge is only in play in one state in an series of successors. The base and inductive case uses the same argument. Let ψ be a successor of ϕ. Then $\psi(v) = I$ only if v is in play in ϕ. Let ζ be such that for every edge $e = (x, y)$ in play, and where $\psi(v) = I$, $\zeta(e) = 1$. This will not affect the other edges, since they will not be in play in any other states.

(iii) is shown by induction of t. First let $E' \subset E$. Let $\tau : \{0, 1\}^E \to \{0, 1\}^{E'}$ be the map restricting from ζ to E'. The measure of probability that $X_{E'}$ hits a $\zeta' \in \{0, 1\}^{E'}$ is $P(X_{E'} = \zeta') = \prod_{\{e\in E' | \zeta'(e)=1\}} \mu(e) \prod_{\{e\in E' | \zeta'(e)=0\}} (1 - \mu(e))$. But more interesting is that by independence of variables, we have that for any ζ',

$$P(X_{E'} = \zeta') = P(X_E = \tau^{-1}(\zeta')). \tag{4.1}$$

For the base case $t = 0$, this is clear that $P(\Psi_0 = \psi | \Psi_0 = \phi) = P(X_e \in \{\zeta | \epsilon_\zeta^0(\phi) = \psi\})$, since the probability is 1 if $\psi = \phi$ and 0 otherwise.

Let us now show the inductive step. First, by Equation 4.1 we get

$$\begin{aligned} P(\Psi_t = \psi | \Psi_{t-1} = \phi) &= \prod_{v\in V} \mu(X_{P_\phi(v)} \in \{\zeta' \epsilon_{\zeta'}(\phi)(v) = \psi(v)) \\ &= P(X_E \in \{\zeta | \epsilon_\zeta(\phi) = \psi\}) \end{aligned} \tag{4.2}$$

Then

$$\begin{aligned}
P(\Psi_t = \psi|\Psi_0 = \phi) &= \sum_{\varphi \in St(G)} P(\Psi_t = \psi|\Psi_{t-1} = \varphi)P(\Psi_{t-1} = \varphi|\Psi_0 = \phi) \\
&= \sum_{\varphi \in St(G)} P(X_E \in \{\zeta|\epsilon_\zeta^{t-1}(\phi) = \psi\})P(\Psi_{t-1} = \varphi|\Psi_0 = \phi) \\
&= \sum_{\varphi \in St(G)} P(X_E \in \{\zeta|\epsilon_\zeta^{t-1}(\phi) = \varphi\})P(X_E \in \{\zeta|\epsilon_\zeta(\varphi) = \psi\}) \\
&= \sum_{\varphi \in St(G)} P(X_E \in \{\zeta|\epsilon_\zeta^{t-1}(\phi) = \varphi\ \epsilon_\zeta(\varphi) = \psi\}) \\
&= P(X_E = \{\zeta|\epsilon_\zeta^{n}(\phi) = \psi\}
\end{aligned}$$

The first equality is by Theorem 2.5(Chapman Kolmogorov Equation),. The second equality is by induction hypothesis. The third quality is by Equation 4.2. The fourth step is by the fact that an edge is only in play once, and thus the two events are independent and can be merged. The last step is because of the events are disjoints, and can thus be summed together. □

THEOREM 4.3. *Given $\zeta \in \{0,1\}^E$, initial state $\phi_0 \in \{S,I\}^V, n \in \mathbb{N}$ and $v \in V$. Then $\epsilon_\zeta^n(\phi_0)(v) \in \{I,R\}$ iff $\zeta^{-1}(1)$ contains a directed path from an infected vertex of ϕ_0 to v of length at most n.*

PROOF. If we show that $\epsilon_\zeta^n(\phi_0)(v) = I$ iff ζ^{-1} contains a directed path from an infected vertex of ϕ_0 to v of length n we are done. For the base case $n = 0$ this is clear. In the inductive step, we need to prove both suffient and necessary condition.

First assume that $\epsilon_\zeta^n(\phi_0)(v) = I$. Then $\epsilon_\zeta^{n-1}(\phi_0)(w) = I$ for an predecessor w and $\zeta((v,w)) = 1$. By inductive hypothesis ζ^{-1} contains a directed path from an infected vertex of ϕ_0 to v of length $n-1$.

For the necessary condition, assume $v_1, ..., v_n = v$ is the path from an infected vertex to v, and $\zeta((v_i, v_{i+1}) = 1$. By induction hypothesis $\epsilon_\zeta^{n-1}(v) = I$. Then by the definition of ϵ, $\epsilon_\zeta^n(v) = \epsilon_\zeta(\epsilon_\zeta^{n-1})(v) = I$. □

We are now prepared to prove the first monotonic property. Assume that $\phi \subseteq \psi$. Let us denote η by

$$\eta(\phi, v) = \{\zeta|\text{there is a path in } \zeta^{-1}(1) \text{ of length at most } n \text{ from } \phi_0^{-1}(I) \text{ to } v\}$$

By definition we have

$$\begin{aligned}
P(\Psi_t = \psi|\Psi_0 = \phi) &= \mu(\eta(\phi, v)) \\
P(\Psi_t = \psi|\Psi_0 = \phi) &= \mu(\eta(\psi, v))
\end{aligned}$$

By assumption $\phi^{-1}(I) \subseteq \psi^{-1}(I)$. If there is a path from $\phi^{-1}(I)$ to v, then it is clear this is also a path from $\psi^{-1}(I)$ to v. Thus the first monotocity property holds.

Let us now prove the second monotonic property. Since $X_E = (X_{e_1}, ..., X_{e_m})$ we can apply the method of random sampling to generate random $X_E = (f_{X_e}^{-1}(U))_{e \in E}$

by drawing $|E|$ times from the random uniform distrubution. Let $(z_e) \in [0,1]^E$ be such a drawing. Then

$$X_E(e) = \begin{cases} 1 & \text{if } z_e \leq \mu(e) \\ 0 & \text{else} \end{cases}$$

Let $\{\sigma_1, ..., \sigma_s\}$ be the directed paths from an infected vertex of the initial vertex to v with length n. For each σ_i, the subset $S_i \in \{0,1\}^E$ iff $S_i = \{(z_1, .., z_{|E|}) | z_j \leq \mu(e_j)$if e_j is an edge of $\sigma_i\}$. Then ζ infects v in n or fewer steps iff it chooses $(z_1, .., z_m) \in S = \bigcup S_i$. By doing the same procedure using edge probability $\mu'(e_i) \geq \mu(e_i)$ we end up with a S' with $S \subset S'$. By definition of measure function $P(S) \leq P(S')$ and the second monotic properties is shown.

2. A proof for the monotonicity properties for SIS networks

THEOREM 4.4. *Monotonicity properties for SIS networks*

(i) Given start states ϕ, ψ with $\phi \leq \psi$ in network N, then for any vertice v and $t \geq 0$ we have $P(\Upsilon_t(v) = I | \Upsilon_0 = \phi) \leq P(\Upsilon_t(v) = I | \Upsilon_0 = \psi)$

(ii) Given networks N_0, N_1 with $N_0 < N_1$, then for any initial state ϕ, vertice v and $n \geq 0$ we have $P(\Upsilon_{t,N_0} = I | \Upsilon_{0,N_0} = \phi) \leq P(\Upsilon_{t,N_1} = I | \Upsilon_{0,N_1} = \phi)$

By the previous section, we have ideas to show that the monotonicity properties also holds for SIS graph. The proof will be more complicated than for SIR-graphs, since the vertices also has the ability to cure itself. This may lead to that an edge may transmit an infection more than once.

Let $X = (X_{e_1}, ..., X_{e_m}, Y_{v_1}, ..., Y_{v_n})$ where X_{e_i} and X_{v_j} are independent Bernoulli random variables. X_{e_i} has probability distribution 1 with probability $\mu(e_i)$ and 0 otherwise. Y_{v_j} has probability distribution 1 with probability $\rho(v_j)$ and 0 otherwise.

It is clear that X is a random variable , and can be measured by the probability that X hits $\zeta \in \{0,1\}^{EV}$ by

$$P(X = \zeta) = \prod_{\{e|\zeta(e)=1\}} \mu(e) \prod_{\{e|\zeta(e)=0\}} (1 - \mu(e)) \prod_{\{v|\zeta(v)=1\}} \rho(v) \prod_{\{v|\zeta(v)=1\}} (1 - \rho(v))$$

We create the successor map $\epsilon_\zeta \times St(G) \to St(G)$ such that

$$\epsilon(\zeta, \phi)(v) = \begin{cases} I & \text{if } e = vw \in \delta_1(e), \phi(w) = I \text{ and } \zeta(e) = 1 \\ I & \text{if } \phi(v) = I \text{ and } \zeta(v) = 0 \\ S & \text{else} \end{cases}$$

for any $\zeta \in \{0,1\}^V \times \{0,1\}^E$ and $\phi \in St(G)$. We also introduce the notation that $\epsilon^k_{\zeta_1,...,\zeta_n}(\phi) = \epsilon_{\zeta_n}(\epsilon^{k-1}_{\zeta_1,...,\zeta_{n-1}}(\phi))$ for $k > 0$, and $\epsilon^0_{\zeta_1,...,\zeta_n}(\phi) = \phi$.

The following theorem, corresponds to Theorem 4.2 for the SIS graph, and tells about the relationship between $(X_{E,V})$ and $(\Upsilon_t)_{t \in \mathbb{Z}_+}$. We denote $X^{(1)}, ..., X^{(n)}$ to be n independent drawings of the random variable X.

THEOREM 4.5. *Properties of the map $\epsilon_\zeta : St(G) \to St(G)$*

(i) Given state ϕ and $\zeta_1, ..., \zeta_n \in \{0,1\}^{EV}$, the sequence $(\epsilon^i_{\zeta_1,...,\zeta_i}(\phi_t))_{t\in\mathbb{Z}_+}$ is a series of successors

(ii) Every possible series of successors arises in this manner

(iii) Given states ϕ, ψ and $\zeta \in \{0,1\}^{|E|} \times \{0,1\}^{|V|}$,
$P(\Upsilon_t = \psi | \Upsilon_0 = \phi) = P(X^{(1)}, ..., X^{(n)} \in \{\zeta_1, ..., \zeta_n | \epsilon^n_{\zeta_1,...,\zeta^n}(\phi) = \psi\})$

PROOF. (i) A successor of a state in a SIS network needs only to satisfy one property, that a vertex is only infected if it was in play at the previous timestep. For the map, we see that the only possibility for a vertex for ϵ_{X_e} to be infected is if the vertex is in play.

(ii) is shown by induction. Let ψ be a successor ϕ. We do this for every vertex. Then for $\psi(v) = I$ iff v was in play in ϕ. Then either if v has a edge $e = (w, v)$ with infected source w, or v was infected at state ϕ. For the first case fix $\zeta(e) = 1$ and the other case, let $\zeta(v) = 0$. If $\psi(v) = S$,for all e edges with sink v, let $\zeta(v) = 0$, and set $\zeta(v) = 1$. Then $\epsilon_\zeta(\phi) = \psi$.

(3) The proof of this is very similar with the proof for Theorem 4.2(iii), and will therefore be ommited.

□

THEOREM 4.6. *Let $\zeta_1, \zeta_2 ... \in \{0,1\}^{VE}$. Define $\phi_k = \epsilon^k_{\zeta_1,...,\zeta_k}(\phi)$ and $\phi'_t = \epsilon^k_{\zeta_1,...,\zeta_k}(\phi')$. Then*

$$\phi_k \subseteq \phi'_k$$

for all $v \in V$ and $k \geq 0$.

PROOF. We will prove this Equation by induction on k. For the base case we have by assumption, $\phi_0 \leq \phi'_0$.

For the inductive case, let us assume that $\phi_k(v) = I$, and show that $\phi'_k(v) = I$. We know that ϕ_k is the successor of ϕ_{k-1} and inspect the cases for how this can occur. The first case is that there exists an edge $e_{w,v}$ such that $\zeta_{k-1}(e_{w,v}) = 1$ and $\phi_{k-1}(w) = I$. By the induction hypothesis we have $\phi'_{k-1}(w) = I$, and thus $\phi'_k(v) = \epsilon_{\zeta_{k-1}}(\phi'_{k-1})(v) = I$. The second case is that $\phi_{k-1} = I$ and $\zeta_{k-1}(v) = 0$. By the induction hypthesis we have $\phi'_{k-1} = I$, and thus $\phi'_k(v) = \epsilon_{\zeta_k}(\phi'_{k-1})(v) = I$. Thus $\phi_k \leq \phi'_k$. The result follows. □

Then

$$\begin{aligned} P(\Upsilon_t(v) = I | \Upsilon_0 = \phi) &= P((\Upsilon^{(1)}, ..., \Upsilon^{(n)}) \in \{\zeta_1, ..., \zeta_n : \epsilon_{\zeta_1,...,\zeta_n}(\phi_0)(v) = I\}) \\ &\leq P((\Upsilon^{(1)}, ..., \Upsilon^{(n)}) \in \{\zeta_1, ..., \zeta_n : \epsilon_{\zeta_1,...,\zeta_n}(\phi'_0)(v) = I\}) \\ &= P(\Upsilon_t(v) = I | \Upsilon_0 = \phi') \end{aligned}$$

The first line is by Theorem 4.5. The second line is by Theorem 4.6. The third line is by Theorem 4.5. This shows the first monotonicity property.

It is now time to prove the second monotonicity property. We will first assume there is only an edge e' ,$\mu_{N_0}(e') < \mu_{N_1}(e')$, for all other edges e, $\mu_{N_0}(e) = \mu_{N_1}(e)$ and for all vertices v, $\rho_{N_0}(v) = \mu_{N_1}(v)$

Let $\zeta \in \{0,1\}^{EV}$ such that $\zeta(e') = 0$. Define the operation $o_{e'} : \{0,1\}^{EV} \to \{0,1\}^{EV}$ such that

$$o_{e'}(\zeta)(j) = \begin{cases} 1 & \text{if } j = e' \\ \zeta(j) & \text{else} \end{cases}$$

The first thing we need to prove is that for

THEOREM 4.7. *Given a start-state ϕ_0 and $\zeta_1, ..., \zeta_n$, we have for any k*

$$\epsilon_{\zeta_1,...,\zeta_{k-1},\zeta_k,\zeta_{k+1}...,\zeta_n}(\phi_0) \subseteq \epsilon_{\zeta_1,...,\zeta_{k-1},o_{e'}(\zeta_k),\zeta_{k+1}...,\zeta_n}(\phi_0)$$

PROOF. Let $\psi = \epsilon_{\zeta_1,...,\zeta_{k-1}}(\phi_0)$. Then the above expression is the same as

$$\epsilon_{\zeta_k,\zeta_{k+1},...,\zeta_{k-1}}(\psi) \subseteq \epsilon_{o_e(\zeta_k),\zeta_{k+1},...,\zeta_{k-1}}(\psi)$$

Let $\epsilon_{\zeta_k}(\phi_{k-1}) = \phi_k$ and $\epsilon_{o_{e'}(\zeta_k)}(\phi_{k-1}) = \phi'_k$ Assume that for a $v \in V$, we have that $\phi_k(v) = I$. Then there is two cases where this event can occur. Either $\phi_{k-1}(v) = I$ and $\zeta_k(v) = 1$, or there exists an edge $e_{w,v}$ such that $\phi_{k-1}(w) = I$ and $\zeta_k(e) = 1$. In both case, this will not coincide with the operation $o_{e'}$ since this operation that transform $\zeta(e') = 0$ to $o_{e'}(\zeta)(e') = 1$. Thus $\phi_k = \epsilon_{\zeta_k}(\phi_{k-1}) \leq \epsilon_{o_{k'}(\zeta_k)}(\phi_{k-1}) = \phi'_k$. By Theorem 4.6,

$$\epsilon_{\zeta_{k+1},...,\zeta_n}(\epsilon_{\zeta_k}(\phi_{k-1})) \subseteq \epsilon_{\zeta_{k+1},...,\zeta_n}(\epsilon_{\zeta_k}(\phi_{k-1}))$$

and the result follows. □

We define $X_N^{(k)}$ to be a random variable where the edges and vertices are drawn with probabilities from network N at time k.

THEOREM 4.8. *Let $Z = \{\zeta_1, ..., \zeta_n | \epsilon_{\zeta_1,...,\zeta_n}(\phi_0)(v) = I\}$. Then for any $i : 0 < i \leq n$ we have*

$$P(X_{N_1}^{(1)}, ..., X_{N_1}^{(i-1)}, X_{N_0}^{(i)}, ..., X_{N_0}^{(n)} \in Z) \leq P(X_{N_1}^{(1)}, ..., X_{N_1}^{(i)}, X_{N_0}^{(i+1)}, ..., X_{N_0}^{(n)} \in Z)$$

PROOF. First let us check the case $\xi = \zeta_1, ..., \zeta_n \in Z$ where $\zeta_i(e') = 0$. Then by Theorem 4.7, there is a corresponding element $\xi' = \zeta_0, ..., o_{e'}(\zeta_i), ..., \zeta_m \in Z$. We define the set containing all such ξ and corresponding ξ' by Z_0. Then

$$P(X_{N_1}^{(1)}, ..., X_{N_1}^{(i-1)}, X_{N_0}^{(i)}, ..., X_{N_0}^{(n)} \in Z) \in \{\xi, \xi'\}) \tag{4.3}$$

is equal to

$$\begin{aligned}
&P(X_{N_1}^{(1)}, ..., X_{N_1}^{(i-1)}, X_{N_0}^{(i)}, ..., X_{N_0}^{(n)} = \xi) + P(X_{N_1}^{(1)}, ..., X_{N_1}^{(i-1)}, X_{N_0}^{(i)}, ..., X_{N_0}^{(n)} = \xi') \\
&= P(X_{N_1}^{(1)}, ..., X_{N_1}^{(i-1)} = \zeta_1, ..., \zeta_{i-1}) P(X_{N_0}^{(i)} = \zeta_i) P(X_{N_0}^{(i+1)}, ..., X_{N_0}^{(n)} = \zeta_{i+1}, ..., \zeta_n) \\
&\quad + P(X_{N_1}^{(1)}, ..., X_{N_1}^{(i-1)} = \zeta_1, ..., \zeta_{i-1}) P(X_{N_0}^{(i)} = o_{e'}(\zeta_i)) P(X_{N_0}^{(i+1)}, ..., X_{N_0}^{(n)} = \zeta_{i+1}, ..., \zeta_n) \\
&= P(X_{N_1}^{(1)}, ..., X_{N_1}^{(i-1)} = \zeta_1, ..., \zeta_{i-1}) P(X_{N_1}^{(i+1)}, ..., X_{N_1}^{(n)} = \zeta_{i+1}, ..., \zeta_n) \\
&\qquad \times \left(P(X_{N_0}^{(i)} = \zeta_i) + P(X_{N_0}^{(i)} = o_{e'}(\zeta_i)) \right)
\end{aligned}$$

The first equality is since ξ and ξ' are disjoint events, we can by property of measure split them to a sum. The second equality is by independence of the random variables $X^{(i)}$.

By probability measure of X_{N_0} we get

$$
\begin{aligned}
&P(X_{N_0}^{(i)} = \zeta_i) \\
&= \prod_{e:\zeta_i(e)=1} \mu_{N_0}(e) \prod_{e:\zeta_i(e)=0} (1-\mu_{N_0}(e)) \prod_{v:\zeta_i(v)=1} \rho_{N_0}(v) \prod_{v:\zeta_i(v)=0} (1-\mu_{N_0}(v)) \\
&= \mu_{N_0}(e') \\
&\quad \times \prod_{e:\zeta_i(e)=1, e\neq e'} \mu_{N_0}(e) \prod_{e:\zeta_i(e)=0, e\neq e'} (1-\mu_{N_0}(e)) \prod_{v:\zeta_i(v)=1} \rho_{N_0}(v) \prod_{v:\zeta_i(v)=0} (1-\mu_{N_0}(v)) \\
&= \mu_{N_0}(e') \\
&\quad \times \prod_{e:\zeta_i(e)=1, e\neq e'} \mu_{N_1}(e) \prod_{e:\zeta_i(e)=0, e\neq e'} (1-\mu_{N_1}(e)) \prod_{v:\zeta_i(v)=1} \rho_{N_1}(v) \prod_{v:\zeta_i(v)=0} (1-\mu_{N_1}(v))
\end{aligned}
$$

by a similar argument we get that

$$
\begin{aligned}
&P(X_{N_0}^{(i)} = o_{e'}(\zeta_i)) \\
&= (1-\mu_{N_0}(e')) \\
&\quad \times \prod_{e:\zeta_i(e)=1, e\neq e'} \mu_{N_1}(e) \prod_{e:\zeta_i(e)=0, e\neq e'} (1-\mu_{N_1}(e)) \prod_{v:\zeta_i(v)=1} \rho_{N_1}(v) \prod_{v:\zeta_i(v)=0} (1-\mu_{N_1}(v))
\end{aligned}
$$

ey combining these two equation we get

$$
\begin{aligned}
&P(X_{N_0}^{(i)} \in \{\zeta_i, o_{e'}(\zeta_i\}) \\
&= (\mu_{N_0}(e') + 1 - \mu_{N_0}(e')) \\
&\quad \times \prod_{e:\zeta_i(e)=1, e\neq e'} \mu_{N_1}(e) \prod_{e:\zeta_i(e)=0, e\neq e'} (1-\mu_{N_1}(e)) \prod_{v:\zeta_i(v)=1} \rho_{N_1}(v) \prod_{v:\zeta_i(v)=0} (1-\mu_{N_1}(v)) \\
&= \prod_{e:\zeta_i(e)=1, e\neq e'} \mu_{N_1}(e) \prod_{e:\zeta_i(e)=0, e\neq e'} (1-\mu_{N_1}(e)) \prod_{v:\zeta_i(v)=1} \rho_{N_1}(v) \prod_{v:\zeta_i(v)=0} (1-\mu_{N_1}(v)) \\
&= (\mu_{N_1}(e') + 1 - \mu_{N_1}(e')) \\
&\quad \times \prod_{e:\zeta_i(e)=1, e\neq e'} \mu_{N_1}(e) \prod_{e:\zeta_i(e)=0, e\neq e'} (1-\mu_{N_1}(e)) \prod_{v:\zeta_i(v)=1} \rho_{N_1}(v) \prod_{v:\zeta_i(v)=0} (1-\mu_{N_1}(v)) \\
&= P(X_{N_1}^{(i)} = \zeta_i) + P(X_{N_1}^{(i)} = o_{e'}(\zeta_i))
\end{aligned}
$$

By putting them in Equation 4.3 we get that

$$
P(X_{N_1}^{(1)}, ..., X_{N_1}^{(i-1)}, X_{N_0}^{(i)}, ..., X_{N_0}^{(n)} \in \{\xi, \xi'\}) = P(X_{N_1}^{(1)}, ..., X_{N_1}^{(i)}, X_{N_0}^{(i+1)}, ..., X_{N_0}^{(n)} \in \{\xi, \xi'\})
$$

and therefore

$$
\begin{aligned}
&P(X_{N_1}^{(1)}, ..., X_{N_1}^{(i-1)}, X_{N_0}^{(i)}, ..., X_{N_0}^{(n)} \in Z_0) \\
&\qquad = P(X_{N_1}^{(1)}, ..., X_{N_1}^{(i)}, X_{N_0}^{(i+1)}, ..., X_{N_0}^{(n)} \in Z_0)
\end{aligned} \tag{4.4}
$$

since Z_0 consists of all the pairs.

Now let us check the case $\xi = \zeta_1, ..., \zeta_n \in Z$ where $\zeta_i(e') = 1$. Then

$$\begin{aligned}
&\mu_{N_0}(\zeta_i) \\
&= \prod_{e:\zeta_i(e)=1} \mu_{N_0}(e) \prod_{e:\zeta_i(e)=0} (1-\mu_{N_0}(e)) \prod_{v:\zeta_i(v)=1} \rho_{N_0}(v) \prod_{v:\zeta_i(v)=0} (1-\mu_{N_0}(v)) \\
&= \mu_{N_0}(e') \\
&\quad \times \prod_{e:\zeta_i(e)=1, e\neq e'} \mu_{N_1}(e) \prod_{e:\zeta_i(e)=0} (1-\mu_{N_1}(e)) \prod_{v:\zeta_i(v)=1} \rho_{N_1}(v) \prod_{v:\zeta_i(v)=0} (1-\mu_{N_1}(v)) \\
&< \mu_{N_1}(e') \\
&\quad \times \prod_{e:\zeta_i(e)=1, e\neq e'} \mu_{N_1}(e) \prod_{e:\zeta_i(e)=0} (1-\mu_{N_1}(e)) \prod_{v:\zeta_i(v)=1} \rho_{N_1}(v) \prod_{v:\zeta_i(v)=0} (1-\mu_{N_1}(v)) \\
&= \mu_{N_0}(\zeta_i)
\end{aligned}$$

Thus

$$P(X^{(1)}_{N_1}, ..., X^{(i-1)}_{N_1}, X^{(i)}_{N_0}, ..., X^{(n)}_{N_0} = \xi\}) < P(X^{(1)}_{N_1}, ..., X^{(i)}_{N_1}, X^{(i+1)}_{N_0}, ..., X^{(n)}_{N_0} = \xi\})$$

and therefore

$$P(X^{(1)}_{N_1}, ..., X^{(i-1)}_{N_1}, X^{(i)}_{N_0}, ..., X^{(n)}_{N_0} = Z - Z_0\}) \tag{4.5}$$

(4.6)

$$< P(X^{(1)}_{N_1}, ..., X^{(i)}_{N_1}, X^{(i+1)}_{N_0}, ..., X^{(n)}_{N_0} = Z - Z_0\})$$

We can now conclude using Equation 4.4 and 4.5 that

$$\begin{aligned}
&P(X^{(1)}_{N_1}, ..., X^{(i-1)}_{N_1}, X^{(i)}_{N_0}, ..., X^{(n)}_{N_0} \in Z) \\
&\leq P(X^{(1)}_{N_1}, ..., X^{(i)}_{N_1}, X^{(i+1)}_{N_0}, ..., X^{(n)}_{N_0} \in Z)
\end{aligned}$$

□

By applying Theorem 4.8 n times we get

$$P(X^{(1)}_{N_0}, ..., X^{(n)}_{N_0} \leq P(X^{(1)}_{N_1}, ..., X^{(n)}_{N_1}) \tag{4.7}$$

Thus

$$\begin{aligned}
P(\Upsilon_{t,N_0} = I | \Upsilon_{0,N_0} = \phi) &= P(X^{(1)}_{N_0}, ..., X^{(n)}_{N_0} \in \{\zeta_1, ..., \zeta_n : \epsilon_{\zeta_1,...,\zeta_n}(\phi_0)(v) = I\}) \\
&\leq P(X^{(1)}_{N_0}, ..., X^{(n)}_{N_0} \in \{\zeta_1, ..., \zeta_n : \epsilon_{\zeta_1,...,\zeta_n}(\phi_0)(v) = I\}) \\
&= P(\Upsilon_{t,N_1} = I | \Upsilon_{0,N_1} = \phi)
\end{aligned}$$

where first line is by Theorem 4.5. The second line is Equation 4.7. The second third line is by Theorem 4.5.

This holds when there is only one edge with $\mu(e)$ different. The other case is that for a chosen vertex v' we have $\rho_{N_0}(v') > \rho_{N_1}(v')$, for all other vertices v $\rho_{N_0}(v) = \mu_{N_1}(v)$, and for all edges e $\mu_{N_0}(e) = \mu_{N_1}(e)$. We can show that the

second monotonicity property holds for this case by using similar argument, but by using the operation $o_{v'} : \{0,1\}^{EV} \to \{0,1\}^{EV}$ such that

$$o_{v'}(\zeta)(j) = \begin{cases} 0 & \text{if } v = v' \\ \zeta(j) & \text{else} \end{cases}$$

Because of the similarities, this proof will be ommited.

Now we are ready to prove the second monotonicity property for the SIS network. Let N_0 and N_1 be the two networks such that $N_0 \leq N_1$. If they are equal, then we do not need to process further. By starting with N_0 and and changing probability μ_{N_0} to μ_{N_1} for one edge at a time, and afterwards probability ρ_{N_0} to ρ_{N_1} for one vertex at a time such that we in the at last end up with N_1 we know that the property $P(\Upsilon_{t,N_0} = I | \Upsilon_{0,N_0} = \phi) \leq P(\Upsilon_{t,N_1} = I | \Upsilon_{0,N_1} = \phi)$ still holds, and the result follows.

3. An alternative proof for the monotonicity properties for SIS networks

We will in this section give an alternative proof for showing that the monotonicity properties holds for SIS-network N by constructing a cover SIR-network $\tilde{N}$, and find a mapping between N and $\tilde{N}$.

Let $N = (G, \mu, \rho)$ be a SIS network where $G = (V, E)$. First, let us describe the cover graph $\tilde{N} = (\tilde{G}, \mu')$ where $\tilde{G} = (\tilde{V}, \tilde{E})$. Define the vertices $\tilde{V}$ as a tuple of the vertices in V, and the natural integers. Let m be the maximum timestep we want to inspect.

$$\tilde{V} = (V \cup \{T\}) \times \mathbb{N}_m$$

The edges we define in the following way

$$\begin{aligned} \tilde{E} = &\{e_{(v,i),(v,i+1)} \text{ for all } v \in V \text{ and } 0 \leq i \leq m-1\} \\ &\cup \{e_{(v,i),(w,i+1)} \text{ for all } (v,w) \in E \text{ and } 0 \leq i \leq m-1\} \\ &\cup \{e_{(T,i),(t,i+1)} \text{ for all } 0 \leq i \leq m-1\} \end{aligned}$$

Define the measure function $\mu : \tilde{E} \to [0,1]$ for the cover graph in the following way:

$$\begin{aligned} \mu(e_{(v,i),(v,i+1)}) &= 1 - \rho(v) \\ \mu(e_{(v,i),(w,i+1)}) &= \mu(e_{v,w}) \\ \mu(e_{(T,i),(T,i+1)}) &= 1 \end{aligned}$$

We can think of the vertices of the cover graph as levels, where the i-th level is the vertices $(v,i)_{v \in V}$. The idea is that every vertex and timestep in the SIS network represents a vertex in the SIR network. This means that (v,i) in the SIR graph represent the state of vertex v at time i. The vertices $T \times N$ can be seen as the timeline, which indicate which time we are at the current moment.

First, let us define $Init(\tilde{N})$.

DEFINITION 4.1. *A state ϕ is in $Init(\tilde{N})$ if it satisfy the following condition.*

(i) For all $v \in \tilde{V}$ and $t > 0$: $\phi(v,t) = S$.

Now, let us define the space $Val(\tilde{N})$.

DEFINITION 4.2. *A state ϕ is in $Val(\tilde{N})$ iff There exists a $t \in \mathbb{N}_m$ such that:*

(i) $\phi(T,t) = I$.

(ii) For all $j < t$: $\phi(T,j) = R$.

(iii) For all $j > t$: $\phi(T,j) = S$.

For a $\phi \in Val(\tilde{N})$, there exists only one t in $\mathbb{N}_m$ such that $\phi(T,t) = I$. We let the function $\gamma : Val(N) \to \mathbb{N}_m$ be defined by $\gamma(\phi) = t$.

Now it is time to define the map $\Phi : Val(\tilde{N}) \to St(N)$ between the cover graph and SIS graph. Define $\Phi(\phi)$ to be such that for every $v \in V$

$$\Phi(\phi)(v) = \phi(v, \gamma(\phi))$$

THEOREM 4.9. *The following properties holds*

(i) If $\phi_0, ..., \phi_n$ is a chain of possible successors in $\tilde{N}$, and where $\phi_0 \in Init(\tilde{N})$, then $\Phi(\phi_0), ..., \Phi(\phi_n)$ is a chain of possible sucessors in N.

(ii) Given a successors of states $\psi_0, \psi_1, ..., \psi_n$, there exist an unique successors of states $\phi_0, \phi_1, ..., \phi_n$ in $St(\tilde{N})$ and where $\phi_0 \in Init(\tilde{N})$ such that $\Phi(\phi_i) = \psi_i$.

(iii) Φ carries the function f on probability measures on $Val(\tilde{N})$ to $St(N)$ This means for $\phi_0, ..., \phi_n$ where $\phi_0 \in Init(\tilde{N})$,

$$\begin{aligned} &P(\Psi_0, ..., \Psi_n = \phi_0, ..., \phi_n | \Psi_0 = \phi_0) \\ &\quad = P(\Upsilon_0, ..., \Upsilon_n = \Phi(\phi_0), ..., \Phi(\phi_n) | \Upsilon_0 = \Phi(\phi_0)). \end{aligned}$$

PROOF. (i) The condition that $\Phi(\phi_i)$ is a successor of $\Phi(\phi_{i-1})$ is that for any node v such that $\Phi(\phi_i)(v) = I$, it was infected by the neighbour-node or itself at the previous timestep. We know that since ϕ_i is a sucessor of ϕ_{i-1}, we have that $\phi_{i+1}((t, \gamma(\phi_{i-1}) + 1) = I$. But this can only happen if either $\phi_{i-1}(v', \gamma(\phi_i))$ and edge $e_{v,v'}$ exist or $\phi_{i-1}(v, \gamma(\phi_i - 1)) = I$ by the property of successor of a SIR-graph. The mapping Φ maps $(v', \gamma(\phi_{i-1}))$ and $(v, \gamma(\phi_{i-1}))$ to v' and v which satisfies the condidition.

(ii) is proved by induction. The base case ϕ_0 is uniquely determined by

$$\phi_0(j,i) = \begin{cases} \psi(v) & \text{if } j \in V, i = 0 \\ I & \text{if } j = T, i = 0 \\ S & \text{else} \end{cases}.$$

Let us look at the inductive case. By the hypothesis ϕ_k is uniquely determined. Then for all $v \in V$ and $i \leq \gamma(\phi_k)$ we have $\phi_{k+1}(v,i)$ uniquely determined, either S or R, $\phi(T,i) = R$. The edge from $e_{(T,k),(T,k+1)}$ has probability 1 for transmission,

and thus $\phi(T, k+1) = I$. And the infected nodes are determined by $\psi(v)$, since it maps the vertices from $(v, i+1)$ in $Val(\tilde{G})$ to v in $St(G)$.

(iii) By construction of the cover graph we see that

$$P(\Psi_{t+1}(v) = I|\Psi_t = \phi_t) = P(\Upsilon_{t+1}(v) = I|\Psi_t = \Phi(\phi_t)).$$

Then

$$P(\Psi_{t+1} = \phi_{t+1}|\Psi_t = \phi_t) = P(\Upsilon_{t+1} = \phi_{t+1}|\Psi_t = \Phi(\phi_t)).$$

And therefore by Equation 3.2 and 3.4, the result follows.

□

We are now ready to show the first monotonicity property for SIS networks. Assume that $\phi_0 \leq \psi_0$. For any vertex v in the SIS, we get

$$\begin{aligned}
&P(\Upsilon_t(v) = I|\Upsilon_0 = \phi_0) \\
&\quad = \sum_{\{\phi_0,...,\phi_t|\phi_t(v)=I\}} P(\Upsilon_0, ..., \Upsilon_t = \phi_0, ..., \phi_t|\Upsilon_0 = \phi_0) \\
&\quad = \sum_{\{\phi_0,...,\phi_t|\phi_t(v)=I\}} P(\Psi_0, ..., \Psi_t = \Phi(\phi_0), ..., \Phi(\phi_t)|\Psi_0 = \Phi(\phi_0)) \\
&\quad = P(\Psi_t(v,t) \in \{I, R\})|\Psi_0 = \Phi(\phi)) \\
&\quad \leq P(\Psi_t(v,t) \in \{I, R\})|\Psi_0 = \Phi(\psi)) \\
&\quad = \sum_{\{\phi_0,...,\phi_t|\phi_t(v)=I\}} P(\Psi_0, ..., \Psi_t = \Phi(\phi_0), ..., \Phi(\phi_t)|\Psi_0 = \Phi(\psi_0)) \\
&\quad = \sum_{\{\phi_0,...,\phi_t|\phi_t(v)=I\}} P(\Upsilon_0, ..., \Upsilon_t = \phi_0, ..., \phi_t|\Upsilon_0 = \psi_0) \\
&\quad = P(\Upsilon_t(v) = I|\Upsilon_0 = \psi)
\end{aligned}$$

In the first line, we sum over all possible of t-successors of states. This can be done since the events are disjoints. In the second line, we use Theorem 4.9 - (iii) . In the third line, by Theorem 4.9 - (ii), we know that $\Phi(\phi_0), ..., \Phi(\phi_t)$ spans the possible series of sucessors of length t in $Val(\tilde{N})$ starting in $Init(V)$. The fourth line, we apply Theorem 4.1-(i) (The first monotonicity property for SIR-networks). The rest of the lines, we apply the arguments the converse way. The result follows

The second monotonicity for SIS networks is shown similar.

$$
\begin{aligned}
&P(\Upsilon_{t,N_0}(v) = I | \Upsilon_{0,N_0} = \phi_0) \\
&\quad = \sum_{\{\phi_0,...,\phi_n | \phi_n(v)=I\}} P(\Upsilon_{0,N_0}, ..., \Upsilon_{n,N_0} = \phi_0, ..., \phi_n | \Upsilon_{0,N_0} = \phi_0) \\
&\quad = \sum_{\{\phi_0,...,\phi_n | \phi_n(v)=I\}} P(\Psi_{0,N_0}, ..., \Psi_{n,N_0} = \Phi(\phi_0), ..., \Phi(\phi_n) | \Psi_{0,N_0} = \Phi(\phi_0)) \\
&\quad = P(\Psi_{t,N_1}(v,t) \in \{I,R\}) | \Psi_{0,N_0} = \Phi(\phi_0)) \\
&\quad \leq P(\Psi_{t,N_1}(v,t) \in \{I,R\}) | \Psi_{0,N_1} = \Phi(\phi_0)) \\
&\quad = \sum_{\{\phi_0,...,\phi_n | \phi_n(v)=I\}} P(\Psi_{0,N_1}, ..., \Psi_{n,N_1} = \Phi(\phi_0), ..., \Phi(\phi_n) | \Psi_{0,N_1} = \Phi(\phi_0)) \\
&\quad = \sum_{\{\phi_0,...,\phi_n | \phi_n(v)=I\}} P(\Upsilon_{0,N_1}, ..., \Upsilon_{n,N_1} = \phi_0, ..., \phi_n | \Upsilon_{0,N_1} = \phi_0) \\
&\quad = P(\Upsilon_{t,N_1}(v) = I | \Upsilon_{0,N_1} = \phi_0)
\end{aligned}
$$

In the forth line we apply the Theorem 4.1-(ii) (The second monotonicity property for SIR-networks). The rest of the lines, we may argument as for the first monotonicity property. The result follows.

CHAPTER 5

Epidemic Threshold

A question we may ask in this chapter is if we can tell something of how will the epidemic develops, knowing the graph topology and probabilites. In SIR-networks we want to find out if the total number of nodes that will be removed is small or large. In SIS-networks, where vertices may be infected several times, we rather ask if the infection will die out, or will it survive. With theory of this threshold, a relevant question is which vertices in the graph should we remove from the graph to lower the epidemic as much as possible. Draief,Ganesh and Massoulie [**5**] showed that for networks where there exists a β such that $\mu(E) = \beta$, there exists an epidemic threshold. Chakrabarti, Wang, wang, Leskovec and Falaoutsos [**3**] showed that for network where there exists a δ, β such that $\rho(V) = \delta$ and $\mu(E) = \beta$, there exists an epidemic threshold.

In the first section of this chapter, we will look for the epidemic threshold for SIR-networks, but not make any assumption for μ. This means that we may have individual probabilies for edges. In the second section, we will look for the epidemic thresholds for SIS-networks, but drop the assumption for μ and ρ. This means that the network may have individual probabilites both for edges and vertices. In the last section we discuss removal strategy on basis of the discussion of Chakrabarti [**3**].

1. Epidemic Threshold for SIR networks

We define the norm $|\cdot|$ on a state ϕ by $|\phi| = |\{v \in V : \phi(v) \in \{I, R\}|$ We are looking at the process $(\Psi_t)_{t\in\mathbb{N}}$ starting in ϕ_0 of a SIR-graph.

THEOREM 5.1. *Assume the graph in the network is strongly connected. Let $S = [s_{ij}]$ be the matrix such that*

$$s_{i,j} = \begin{cases} \mu(e_{i,j}) & \text{if } e_{i,j} \text{ an edge in } G \\ 0 & \text{else} \end{cases}$$

Let $\lambda_{1,S}$ be the greatest eigenvalue of this matrix, and assume $\lambda_{1,S} < 1$.

Then the total number of nodes removed $|\Psi_\infty|$ satisfies, for any $\epsilon > 0$ and for some constant $C > 0$ depending on the graph,

$$P(|\Psi_\infty| > \sqrt{|\Psi_0|} n^{\epsilon+\frac{1}{2}}) \leq Cn^{-\epsilon}$$

If the graph is also such that there exists an $\alpha \in R$ such that for each vertex v, we have that $\sum_{w\in V} \mu(e_{w,v}) = \alpha$. Then for any $\epsilon > 0$ and for some constant $C > 0$

depending on the graph,

$$P(|\Psi_\infty| > |\Psi_0|n^\epsilon) \leq C'n^{-\epsilon}$$

PROOF. Let $\boldsymbol{p} : V \to [0,1]$ be a vector such that $\boldsymbol{p}(v) = P(\Psi_\infty(v) = 1)$. Let $v \in V$. If $v \in \Psi_0(v)$, then it is clear that $\boldsymbol{p}(v) = 1$. Otherwise $\boldsymbol{p}(v) = \sum_{w \in N(i)} \mu(e_{w,v})p(w)$, since an edge is only in play once. Thus we have

$$(I - S)\boldsymbol{p} \leq \Psi_0$$

and by definition of expected value

$$(I - S)E[\Psi_\infty] \leq \Psi_0 \tag{5.1}$$

By definition of matrix norm and property of the largest eigenvalue we get

$$\lim_{k\to\infty} \|S^k\| = \lim_{k\to\infty} \sup_{\|\boldsymbol{v}\|=1} \|S^k\boldsymbol{v}\| = \lim_{k\to\infty} \sup_{\|\boldsymbol{v}\|=1} \|(\lambda_{1,S})^k\boldsymbol{v}\| \to 0.$$

Then

$$\sum_{k=0}^{m} S^k = (I - S^{m+1})(I - S)^{-1} = (I - S)^{-1}$$

which means that $(I - S)^{-1}$ can be written as a convergent geometric series. By Equation 5.1 and multiplying with $(I - \beta S)^{-1}$ on both sides, we get

$$E[\Psi_\infty] \leq (I - S)^{-1}\Psi_0$$

and therefore

$$E(|\Psi(\infty)|) \leq 1^T(I - S)^{-1}\Psi_0 \leq \|1\|\|(I - S)^{-1}\|\|\Psi_0\| \tag{5.2}$$

where we use the Euclidean norm for a vector and matrix operator norm for the matrix. The operator norm of a diagonalizable matrix is by the Spectral Theorem(see [**10**]) the spectral radius λ_1. Then

$$\begin{aligned}\|(I - S)^{-1}\| = \|\sum_{k=0}^{m} S^k\| &= \sum_{k=0}^{k} \lambda_{1,S}^k \\ &= (1 - \lambda_{1,S})^{-1}\end{aligned}$$

By definition of the Euclidean norm, it is clear that

$$\|\Psi_0\| = \sqrt{\sum_{v\in V} \Psi_0(v)} = \sqrt{|\Psi_0|}$$
$$\|1\| = \sqrt{n}$$

Putting it all together we then have

$$E(|\Psi_\infty|) \leq (1 - \lambda_{1,S})^{-1}\sqrt{n|\Psi_0|} \tag{5.3}$$

and therefore

$$\begin{aligned} P(|\Psi_\infty)| > \sqrt{|\Psi_0|}n^{\epsilon+\frac{1}{2}}) &\leq \frac{E(|\Psi_\infty|)}{\sqrt{|\Psi_0}n^{\epsilon+\frac{1}{2}}} \\ &\leq \frac{(1-\lambda_{1,S})^{-1}\sqrt{n|\Psi_0|}}{\sqrt{|\Psi_0}n^{\epsilon+\frac{1}{2}}} \\ &= (1-\lambda_{1,S})^{-1}n^{-\epsilon} \\ &= Cn^{-\epsilon} \end{aligned}$$

where the first line comes from Theorem 2.2(Markovs inequality) and second line putting in Equation 5.3. Then we can conclude that

$$P(|\Psi_\infty| > \sqrt{|\Psi_0|}n^{\epsilon+\frac{1}{2}}) \leq Cn^{-\epsilon}$$

and the first part of the theorem is proven.

For the second part of the theorem, we use the Spectral Decomposition (see [**10**], and since $(I-S)^{-1}$ can be written as a convergent geometric series,

$$(I-S)^{-1} = \sum_{i=1}^{n}(I-\lambda_{i,S})^{-1}w_i w_i^T$$

where $\lambda_1, ..., \lambda_n$ are the eigenvalues and $w_1, ..., w_n$ are the corresponding eigenvectors.

By assumption the rowsum of the system matrix S has probability α. Hence, $w_1 = \frac{1}{\sqrt{n}}\mathbf{1}$ is an eigenvector with corresponding eigenvalue α. By Perron-Frobenius Theorem (see [**10**]), the only eigenvector with only positive entries is the vectors associated with the largest eigenvalue. and thus is α the largest eigenvalue and all the other eigenvectors $w_2, .., w_n$ are orthonormal to w_1. By using Equation 5.2, and inserting for $(1-S)^{-1}$ we get

$$\begin{aligned} E(|\Psi_\infty|) &\leq 1^T(I-S)^{-1}\Psi_0) \\ &= 1^T\sum_{i=1}^{n}(1-\lambda_{i,S})^{-1}w_i w_i^T\Psi_0 \\ &= \sum_{i=1}^{n}(1-\lambda_{i,S})^{-1}1^T w_i w_i^T\Psi_0 \\ &= (1-\lambda_{1,S})^{-1}w_1^T\Psi_0 \\ &= (1-\lambda_{1,S})^{-1}|\Psi_0| \end{aligned}$$

By the Markov inequality we therefore have

$$P(|\Psi_\infty| > |\Psi_0|n^\epsilon) \leq (1-\lambda_{1,S})^{-1}n^{-\epsilon} = C'n^{-\epsilon}$$

and the second part is proven. □

The result of this theorem says that when $\lambda_{1,S} < 1$, and we start with a small infection population, there is a high probability that the total number of vertices ever infected is small.

Let us fix $\mu(e) = \beta$ for all edges e and for a $\beta \in \mathbb{R}$. Then $S = \beta A$, where A is the adjency matrix, and by applying Theorem 5.1 we get the following corollar (Note that this is one of the result of Draief,Ganesh and Massoulie[**5**]).

COROLLARY 5.1. *Let A be the adjency matrix for the graph of the network, and let $\beta\lambda_{1,A} < 1$. the total number of nodes removed $|\Psi_\infty|$ satisfies, for any $\epsilon > 0$ and for some constant $C > 0$ depending on the graph,*

$$P(|\Psi_\infty| > \sqrt{|\Psi_0|}n^{\epsilon+\frac{1}{2}}) \leq Cn^{-\epsilon}$$

If the graph is regular with node degree d, for any $\epsilon > 0$ and for some constant $C > 0$ depending on the graph,

$$P(|\Psi_\infty| > |\Psi_0|n^{\epsilon}) \leq C'n^{-\epsilon}$$

2. Epidemic Threshhold for SIS networks

THEOREM 5.2. *We are given a SIS network $N = (G, \mu, \rho)$ Let $S = [s_{ij}]$ denote the matrix such that*

$$s_{i,j} = \begin{cases} 1 - \rho(i) & \text{if } i = j \\ \mu(e_{i,j}) & \text{if } e_{i,j} \text{ an edge in } G \\ 0 & \text{else} \end{cases}$$

If $\lambda_{1,S} < 1$, then the infection will die out over time.

PROOF. A vertex v does not receive infections from the its predecessors if the infected neighbours does not transmit infection, and has probability

$$\begin{aligned} \eta_t(v) &= \prod_{w \in N_g(v)} (1 - \mu(e_{w,v})P(\Upsilon_t(w) = 1)) \\ &\geq 1 - \sum_{w \in N_g(i)} \mu(e_{w,v})P(\Upsilon_t(w) = 1) \\ &= 1 - \sum_{w \in V} \mu(e_{w,v})P(\Upsilon_t(w) = 1) \end{aligned} \tag{5.4}$$

where the second line holds because all terms are positive and equal or less than 1.

For all v in V, we have

(5.5)

$$\begin{aligned} &P(\Upsilon_t(v) = 1) = 1 - \rho(v)P(\Upsilon_{t-1}(v)) - \eta_t(v)(1 - P(\Upsilon_{t-1}(v))) \\ &\leq 1 - \rho(v)P(\Upsilon_{t-1}(v)) - (1 - \sum_{w \in V} \mu(e_{w,v})P(\Upsilon_{t-1}(w) = 1))(1 - P(\Upsilon_{t-1}(v))) \\ &= (1 - \rho(v))P(\Upsilon_{t-1}(v)) + (1 - P(\Upsilon_{t-1}(v))) \sum_{w \in V} \mu(e_{w,v})P(\Upsilon_{t-1}(w) = 1) \\ &\leq (1 - \rho(v))P(\Upsilon_{t-1}(v)) + \sum_{w \in V} \mu(e_{w,v})P(\Upsilon_{t-1}(w) = 1) \end{aligned}$$

The first line holds because v is infected at time t only if it either was infected at time t and did not get cured or gets an infection from a predecessor. In the second line we insert Equation 5.4.

Let $\boldsymbol{p}_t : V \to [0,1]$ be a vector such that $\boldsymbol{p}_t(v) = P(\Upsilon_t(v) = 1)$. Then

$$\begin{aligned}\boldsymbol{p}_t \leq & S\boldsymbol{p}_{t-1} \\ \leq & S^2\boldsymbol{p}_{t-2} \\ & \vdots \\ \leq & S^t\boldsymbol{p}_0\end{aligned}$$

By spectral decompositition (see [**10**]),

$$\boldsymbol{p}_t \leq \sum_i \lambda_{i,S}^t \boldsymbol{w}_i \boldsymbol{w}_i^T \boldsymbol{p}_0$$

Let $\lambda_1, ..., \lambda_n$ denote the eigenvalues for A. When $t \to \infty$, we sees that $\lambda_{i,S}^t \to 0$ for all i. Thus $\boldsymbol{p_t}$ goes to 0 as t increases, and the result follows. □

Fix the edge transmission probability such that $\mu(E) = \beta$, and the cureness probability such that $\mu(V) = \delta$. Let A be the adjancy matrix, and let $\lambda_{1,A}$ be the greatest eigenvalue. Assume that $\beta/\delta < \lambda_{1,A}^{-1}$. Then $S = \beta A + (1-\delta)I$ is the system matrix, The largest eigenvalue of S is $\lambda_{1,S} = \beta\lambda_{1,A} + 1 - \delta$. If $\lambda_{1,S} < 1$, then $\beta/\lambda < \lambda_{1,A}^{-1}$. By applying Theorem 5.2 we get the following result. (Note that this is one of the main result of Chakrabarti [**5**])..

COROLLARY 5.2. *Let $N = (G, \mu, \rho)$ be a SIS network with $\mu(E) = \beta$ and $\rho(V) = \delta$. Let A be the adjency matrix for the graph G. If $\beta/\delta < \lambda_{1,A}^{-1}$, then the infection will die out over time.*

Chakrabarti [**3**] describes also that if the converse way of the theorem is true, i.e., if the infection dies out over time, then $\beta/\delta < \lambda_{1,A}^{-1}$. In Chapter 8, we do testruns on epidemic thresholds, and it turns out there exists some cases of β, δ such that $\beta/\delta > \lambda_{1,A}^{-1}$ where the infection still dies out. This shows that the converse way does not necessarly need to be true.

We start with the definition of that a fix point is asymtotically stable.

DEFINITION 5.1. *Asymptotically stable*
For a dynamic system $\{\boldsymbol{x}_t\}$ to be asymptotically stable at point $\boldsymbol{y}$ it has to satisfy two conditions:

1. *For all ϵ, there exists an δ such that if $\|\boldsymbol{x}_0 - \boldsymbol{y}|| < \delta$, then $||\boldsymbol{x}_t - \boldsymbol{y}|| < \epsilon$ for all $t > 0$.*
2. *There exists a δ such that if $||\boldsymbol{x}_0 - \boldsymbol{y}|| < d$, then $\boldsymbol{x}_t \to \boldsymbol{y}$ as $t \to \infty$.*

The first condition is called the stable condition, and the other condition is called the attractor condition.

The error in the paper [**3**] is assumption that if the infection dies out, then $(\boldsymbol{p}_t)_{t\in\mathbb{N}}$ needs to have the property of asymtotically stability. This does not necesserarly need to be true. This is because, while $\boldsymbol{x}_0$ spans $\mathbb{R}^{|V|}$, $\boldsymbol{p}_0$ is restricted to components with value in $[0,1]$. While there may be systems starting in $\boldsymbol{x}_0$ not converging, we may have the case that it converge for the restricted $\boldsymbol{p}_0$.

3. Methods for preventing epidemic outbreak

There are several methods to supress the propagation of a virus. We would describe two methods.

3.1. Throttling. Throttling is a method where we limit the maximum probability of transmittion for some or all edges. This can be done for infections in real life for example by introducing rules like using antibacterial after handshake.

3.2. Removing vertices. Removing vertices is another method for preventing outbreaks. In real life infections that would correspond to either vaccinate some persons or isolate them. Let us assume that we have the budget that we can remove k vertices from the graph. We may then ask if there is any better strategy than removing random.

An approach would be to pick the vertices with the largest sum of probabilities of the in-edges. This task have $O(|V|km)$ runtime, where m is the maximal degree of an vertex. A Another algorithm is to pick a vertex at random and go down through a successor-chain of r edges. This is an algorithm that have $O(kr)$ runtime, and is the fastest.

By Theorems 5.1 and 5.2, it suggest that a good idea would be to choose the one that causes the maximal reduction in $\lambda_{1,S}$. To find the optimal we need to find the eigenvalue $\lambda_{1,\tilde{S}}$ after removing vertex v. The fastest way of finding an approxmation of the eigenvalue of a matrix is the power iteration (see [**10**]) which takes $O(|V|^2)$ time per step, and converges at linear time l. To delete a vertex, we need to delete all its edges, which takes m time where m is maximal degree of an vertex. We need to try this to do this for all vertices, and we end up with a runtime $O(|V|^3kml)$. This means that this method is a slow method.

In the last section of Chapter 8, we have plotted the expected number of infected with different removal strategies. In that case, the method based on eigenvalues was the best method. Which of the methods that will work best for other networks is hard to say, and needs more research.

CHAPTER 6

Complexity of computing probability

In this chapter we will show that computing the probabilites for the two networks are #P-hard. Then we know there does not exists an polynomial solution. Throughout this chapter we will refer computing $P(\Psi_t(v) \in \{I, R\} | \Psi_0 = \phi)$ for SIR network as the SIR problem, and computing $P(\Upsilon_t(v) = I | \Upsilon_0 = \phi)$ for SIS network as the SIS problem.

Classifications of #P-problems has its origin in the paper of Valient [[**16**]]. Michael Shapiro and Eckbert [**13**] showed that a similiar problem called the network reliability problem is #P-hard, and M. O. Ball deduced from this that the SIR-problem is #P-hard. In the first section we will give the complete proof that the SIR problem is #P-hard, without touching network reliablity problem. Using the concepts, we show in the second section that the SIS problem is also #P-hard.

1. A proof for the #P-hardness of the SIR problem

We will in this proof first show that the V-Set Connectedness problem and the S-T Connectedness problem is in #PC, and then directly from this deduce that the SIR-problem is #P-hard. (The proof of the V-Set Connecteness problem and the S-T Connectedness problem is taken from the paper of Valiant[**16**]).

PROBLEM 6.1. *The V-Set Connectedness problem*
Given a digraph $G = (V, E)$, $s \in S$ and $V' \subset V$, the problem is how many subgraphs of G contains path from s to each vertex in V'.

THEOREM 6.1. *The V-Set Connectedness problem is #PC.*

PROOF. To show that the problem is in #P, we only need to show that a solution can be evaluated in polynomial time. This can simply be done by a Breath First Search.

The next step is to show that the problem is at least as hard as the Monotone 2SAT problem. Assume that the V-Set Connectedness problem has an polynomial solution. In the Monotone 2SAT problem, we are given $F = c_1 \wedge ... \wedge c_r$ with $c_i = y_{i1} \wedge y_{i2}$, $y_{i1}, y_{i2} \in X$ and where $X = \{x_1, ..., x_n\}$. First of all, we will add the clausule $c_r = x_n \vee \neg x_n$ to F. This is a clausule that is always true, and will not change the problem.

We will now construct a digraph $G = (V, E)$ for transforming this problem to be an instance of the V-Set Connectedness problem.

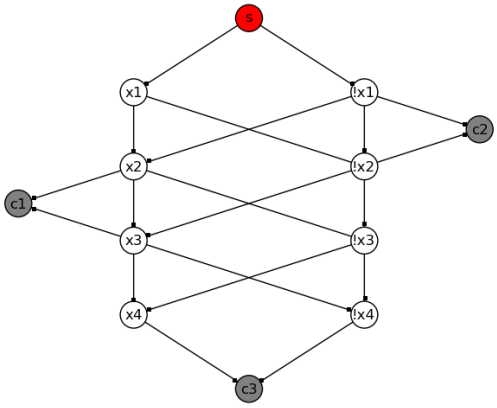

FIGURE 1. Example of construction for showing the S-Set Connectivity problem, where $F = c_1 \wedge c_2$ and $c_1 = x_2 \vee x_3$, $c_2 = \neg x_1 \vee \neg x_2$

Let

$$V = \{c_1, ..., c_{r+1}, x_1, ..., x_n, \bar{x}_1, ..., \bar{x}_n, s\}$$

where $c_1, ..., c_r, c_{r+1}$ represents the clauses, $x_1, ..., x_n$ represent the boolean variables assigned true and $\bar{x}_1, ..., \bar{x}_n$ represents the boolean variables assigned false. We let $\{c_1, ..., c_{r+1}\}$ be the set V' as described in the input for the problem of V-Set Connectedness.

Let the edges in the digraph be

$$E_1 = \{(x, c_i) | x \text{ appears in clause } c_i \text{ in F } \}$$
$$E_2 = \{(x_i, x_{i+1}), (\bar{x}_i, \bar{x}_{i+1}), (x_i, x_{i+1}), (x_i, x_{i+1}) | 1 \leq i \leq n\} \cup \{(s, x_1), (s, \bar{x}_1)\}.$$

The reason we add the clausul c_r is because the only possibility for c_r to be connected to s is that either x_i or $\bar{x}_i$ is connected to s for all i. We suppose that each edge is a random variable with two outcome: operating or failure. Edges in E_1 have probability p for operating, and edges in E_2 have probability q for operating. Let $A_{i,j}$ be the number of subset of $E_1 \cup E_2$ with i edges from E_1 and j edges from E_2 such that s is conncected to V'.

The probability r that s is connected to V' is

$$r(p, q) = \sum_{i=1}^{2r+2} \sum_{j=1}^{4n-2} A_{ij} p^i (1-p)^{2r+2-i} q^i (1-q)^{4n-2-i}$$

. By Theorem 2.9(iii), if we choose p, q and A, and $r(p, q)$ can be evaluated at polynomial time for these values, then $A_{i,j}$ can be deduced in polynomial time. Choose A, p, q such that p and q is a power of two, for example $A = 2^{4n+2r}$, $p = 2^{-2(4n+2r)} = 2^{-a}$ and $q = 2^{-3(4n+2r)^2} = 2^{-b}$. Then all the $A_{i,j}$ can be found

in polynomial time if $r(p,q)$ can be evaluated at polynomial time, which it can by the following argument.

We will construct $\tilde{G} = (\tilde{V}, \tilde{E}_1 \cup \tilde{E}_2)$. For every edge with probability for operating 2^{-k} for a k, we replace this with a path of k edges with probability $\frac{1}{2}$ for each of the edges in the apth. Let $\tilde{E}_i$ be the set of all edges from E_i where the edges are transformed to paths for $i = 1, 2$. Then

$$\begin{aligned} r(p,q) &= \sum_{i=1}^{2r+2} \sum_{j=1}^{4n-2} A_{ij} p^i (1-p)^{2r+2-i} q^i (1-q)^{4n-2-i} \\ &= \sum_{i=1}^{2r+2} \sum_{j=1}^{4n-2} A_{ij} 2^{-|\tilde{E}_1|} 2^{-|\tilde{E}_2|} \\ &= \sum_{i=1}^{2r+2} \sum_{j=1}^{4n-2} A_{ij} 2^{-a|E_1|} 2^{-b|E_2|} \\ &= 2^{-a|E_1|-b|E_2|} \sum_{i=1}^{2r+2} \sum_{j=1}^{4n-2} A_{ij}. \end{aligned}$$

$\sum_{i=1}^{2r+2} \sum_{j=1}^{4n-2} A_{ij}$ is the answer of the V-Set Connectedness problem, and therefore $r(p,q)$ can be solved in polynomial time.

We will take a closer look at the number $A_{r+1,n}$. Let us look at an arbitary subset with $r+1$ edges from E_1 and n edges from E_2 where s is connected to V'. To restrict to $r+1$ edges from E_1 does that it contains one and only one edge of type (x, c_i) for each of the clauses c_i. To restrict to n edges from E_2, deos that it can not happen that both x_i and $\bar{x}_i$ is conncted to s, but only one of them. We are sure that at least one of x_i or $\bar{x}_i$ are connected to s, or else c_{r+1} can not have be connected to s. Hence by assigning

$$x_i = \begin{cases} \text{true} & \text{if } x_i \text{ is connected to } s \\ \text{false} & \text{if } \bar{x}_i \text{ is connected to } s \end{cases}$$

we get a solution of the given 2SAT problem. Thus $A_{r+1,n}$ is the number of solutions of this problem. The Monotone 2SAT problem is therefore in #PC, and the result follows. □

PROBLEM 6.2. *S-T Connectedness*
Given digraph $G = (V, E)$ and $s, t \in V$, the problem is how many subgraphs of G contains a path from s to t.

THEOREM 6.2. *S-T Connectedness is in #PC.*

PROOF. To show that the problem lies in #P, we simply take a solution, and apply Breath-First-Search checking whether is a path from s to t.

To show that this is #P-hard, we will reduce it to the V-Set Connectedness problem. Assume that the S-T Connectedness problem has a polynomial solution. In a V-Set Connectedness problem we are given a digraph $G = (V, E)$, $s \in S$ and $V' \subset V$. Create a new digraph $\tilde{G} = (\tilde{V}, E \cup E_1)$ by letting

$$\tilde{V} = V \cup \{t\}$$

and

$$E_1 = \{(v,t)|v \in V'\}.$$

We suppose that each edge is a random variable with two outcomes: operating or failure. Each edge in E we assign the probability 2^{-1} for operating, and for each edge in E_1 we assign the probability $(1-p)$ for operating. Then probability that an edge in E_1 is not operative is p. Pick n edges from E_1. Then the probability that all n edges is not operating is p^n, and thus the probability that at least one of the n edges is operating is $1-p^n$.

Let A_i be the number of subgraphs of G where s is connected to exactly i nodes from V'. Then the probability $r(p)$ that s is connected to t is

$$r(p) = 2^{-|E|}\sum_{i=1}^{|V'|} A_i(1-p^i)$$

Then $r(p)$ is a a polynomial, and Theorem 2.8 says that the coefficients A_i can be deduced in polynomial time if the value of $r(p)$ can be computed in $|V'|+1$ points at polynomial time. The value $r(p)$ can be evaulated in polynomial time at points $p = 1-2^k$ for $k = 1, ..., |V'|+1$ in the following way.

For every $p = 1-2^k$, create a new digraph G_k where every edge in E_1 is a path of length k with probability 2^{-1} for each of the edges in the path. Denote these new edges by E_2.

For one such path of length k, there is only one subgraph, where it connects the endpoints, and $2^k - 1$ subgraphs, where it does not. Assume there are $|V'|$ paths of length k and denote this set by X_1. Pick i of the paths in X_1, and denote the set by X_2. Then the number of subsets of $X_1 - X_2$ is $2^{k(|V'|-i)}$. The number of subsets of X_2 where there is at least a path of length k is $2^{ki} - (2^k-1)^i = 2^{ki}(1-(1-2^{-k})^i)$. Hence the number of subsets of X_1 where there is at least a path of length k of the paths in X_1 is

$$2^{k(|V'|-i)}2^{ki}(1-(1-2^{-k})^i) = 2^{k|V'|}(1-(1-2^{-k})^i)$$

By inserting $p = 1-2^k$, we get

$$\begin{aligned} r(p) &= 2^{-|E|}\sum_{i=1}^{|V'|} A_i(1-(1-2^{-k})^i) \\ &= 2^{-|E|-k|V'|}\sum_{i=1}^{|V'|} A_i 2^{k|V'|}(1-(1-2^{-k})^i). \end{aligned}$$

$\sum_{i=1}^{|V'|} A_i 2^{k|V'|}(1-(1-2^{-k})^i)$ is the number of subgraphs such that s is connected to t which is the S-T Connectedness problem. Therefore $r(p)$ can be computed in polynomial time for $|V'|+1$ points and A_i can be deduced. $A_{|V'|}$ is the value we are looking for solving the V-Set Connectedness problem, and the result follows. □

Now we have done the preparation, and are now ready to prove that the SIR problem is #P-hard. We will reduce this to the S-T Connectedness problem. Assume that the SIR problem can be solved in polynomial time. In the S-T Connectedness

problem, we are given a digraph $G = (V, E)$ and $s, t \in V$. Create a network $N = (G, \mu)$, where we let $\mu(e) = p$ for all e. We define the initial state $\phi : V \to \{S, I\}$ by

$$\phi(v) = \begin{cases} I & \text{if } v = s \\ S & \text{else} \end{cases}$$

Let A_i be the number of subgraphs with i edges where s has a path to t. Then

$$P(\Upsilon_t(v) \in \{I, R\} | \Psi_0 = \phi) = \sum_{i=1}^{|E|} A_i p^i (1-p)^{|E|-i}$$

which is the SIR problem, and can be assumption be solved at polynomial time. By Theorem 2.9(ii), we can find all the coefficients A_i, by choosing the right values of A and p For instance it works for $A = \max(3, 2^{|E|})$, and $p = 2^{-(|E|+1)}$ Then we can deduce $\sum_{i=1}^{|E|} A_i$, which is the answer of the S-T Connectedness problem, and the result follows.

THEOREM 6.3. *The SIR problem is #P-hard.*

2. A proof for the #P-hardness for the SIS problem

In the third section of Chapter 4 , we created a SIR-cover network of the SIS network. Constructing the cover network is a $O((|V| + |E|)t)$ operation which is polynomial. Finding $P(\Upsilon_t(v) = I | \Upsilon_0 = \phi)$ for the SIS-network is like finding $P(\Psi_t(v, t) \in \{I, R\} | \Psi_0 = (\phi, 0))$ for the SIR-cover network. This means that the SIR problem is at least as hard as computing the SIS networks, which is #P-hard. SIS network differs greatly from SIR network since every vertex can cure itself or stay infected, and will never be removed from the network. Therefore showing the opposite way, that computing SIR-network is at least as hard as computing SIS networks may not be possible. We will instead try to show that the SIS problem is #P hard by taking inspirations from the previous section.

PROBLEM 6.3. *the V-Set SIS Connectedness*
Given a SIS network $N = (G, \mu, \rho)$ with digraph $G = (V, E)$, $s \in S$ and $V' \subset V$, with $\rho(v) = 1$ for all v. the problem is how many possible chain of successors $\phi_0, ..., \phi_t$ is such that $\phi_0^{-1}(I) = \{s\}$, and is such that it infects all vertices in V' at time t.

THEOREM 6.4. *the V-Set SIS Connectedness is in #PC.*

PROOF. To show that the problem is in #P, we show that given a solution, we can simply use a Breath-First-Search to search if there is a path from s to V'.

The next step is to reduce it to the Monotone 2SAT problem. Assume that the V-Set SIS Connectedness has a polynomial solution. In the Monotone 2SAT problem, we are given $F = c_1 \wedge ... \wedge c_r$ with $c_i = y_{i1} \wedge y_{i2}$, $y_{i1}, y_{i2} \in X$ and where $X = \{x_1, ..., x_n\}$. As earlier we will add the constrain $c_{r+1} = x_n \vee \neg x_n$ to F which will not change the problem, since c_{r+1} is always true.

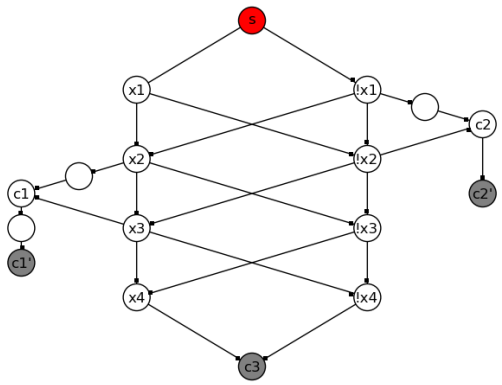

FIGURE 2. Example of construction for showing S-Set SIS Connectivity, where $F = c_1 \wedge c_2$ and $c_1 = x_2 \vee x_3$, $c_2 = \neg x_1 \vee \neg x_2$

We define the digraph in the following way. Start with the set of vertices

$$V = \{s, c_1, ..., c_r, c_{r+1}, x_1, ..., x_n, \bar{x}_1, ..., \bar{x}_n, c'_1, ..., c'_{r+1}\}$$

and the edges

$$E_1 = \{(x, c'_i) | x \text{ appears in clause } c_i \text{ in F } \}$$
$$E_2 = \{(x_i, x_{i+1}), (\bar{x}_i, \bar{x}_{i+1}), (x_i, x_{i+1}), (x_i, x_{i+1}) | 1 \leq i \leq n\} \cup \{(s, x_1), (s, \bar{x}_1)\}$$
$$E_3 = \emptyset$$

Let the function λ be denoted by

$$\lambda(y) = \begin{cases} i & \text{if } y = x_i \\ i & \text{if } y = \neg x_i \end{cases}.$$

We will do the following procedure for every constrain $c_i = y_i \vee y'_i$. Assume that y_i and y'_i is labeled such that $\lambda(y_i) \leq \lambda(y'_i)$. Then $l = \lambda(y'_i) - \lambda(y_i)$ is the difference in level between the two variables in the constrain. Create a path from y_i to c'_i with length $l + 1$, and put the the first edge in the path in E_2, and the rest of the edges in E_3. Also add an edge from y'_i to c'_i, and put the edge in V_2. Create a path from c'_i to c_i with length $n + 1 - \lambda(y'_i)$, and put the edges in E_3. We will use the digraph G consisting of vertices V and edges $E_1 \cup E_2 \cup E_3$ to solve the monotone 2-sat problem. We note that all possible paths from s to a c'_i has length $n + 2$.

Let $V' = \{c_1, c_2, ..., c_r\}$. Define the SIS network $N = (G, \mu, \rho)$, such that

$$\mu(e) = \begin{cases} p & \text{if } e \in E_1 \\ q & \text{if } e \in E_2 \\ 1 & \text{if } e \in E_3 \end{cases}$$

and $\rho(v) = 1$ for all v.

Let the initial state ϕ_0 be such that $\phi_0^{-1}(I) = \{s\}$. Let A_{ij} be the number of subgraphs of G where i and j is the number of edges from E_1 and E_2 that have ever transmitted disease and where all the vertices in V' is infected at time $n+2$. The probability μ that s is connected to all the nodes in V' at time $n+1$ is then $r(p,q)$, where

$$r(p,q) = \sum_{i=1}^{2r+2} \sum_{j=1}^{4n-2} A_{ij} p^i (1-p)^{2r+2-i} q^i (1-q)^{4n-2-i}$$

If we choose $A = 2^{4n+2r}$, $p = 2^{-2(4n+2r)} = 2^{-a}$ and $q = 2^{-3(4n+2r)^2} = 2^{-b}$, this will satisfy the conditions for Theorem 2.9(iii). This means that all the A_{ij} can be found in polynomial time, if $r(p,q)$ can be evaluated at polynomial time.

We can evaluate $r(p,q)$ at polynomial time by constructing a new digraph $\tilde{G}$. Replace the edges e in E_1 by a chain of edges of length a, and where each edge in this chain has probability $\mu(e) = 2^{-1}$ for transmission. Replace the edges in E_2 by a chain of edges of length b, and where each edge e in this chain has probability $\mu(e) = 2^{-1}$ for transmission. Replace the edges in E_3 by a chain of edges of length a, and where each edge e in this chain has probability $\mu(e) = 1$ for transmission. This procedure will do that we still maintain the property that the infection from s reach all $v \in V'$ at same timestep, but now in timestep $(n+1) \times a + b$.

Then

$$\begin{aligned} r(p,q) &= \sum_{i=1}^{2r+2} \sum_{j=1}^{4n-2} A_{ij} p^i (1-p)^{2r+2-i} q^i (1-q)^{4n-2-i} \\ &= \sum_{i=1}^{2r+2} \sum_{j=1}^{4n-2} A_{ij} A_{ij} p^i 2^{-a(n+1)} 2^{-b} \\ &= 2^{-(n+1)\times a - b} \sum_{i=1}^{2r+2} \sum_{j=1}^{4n-2} A_{ij} \end{aligned}$$

The number of subgraphs such that s has a path to all $v \in V'$ on length $(n+1)*a+b$ is the S-Set SIS Connectedness problem, and is equal to $\sum_{i=1}^{2r+2} \sum_{j=1}^{4n-2} A_{ij}$. Thus $r(p,q)$ can be solved in polynomial time.

We see $A_{r+1,n}$ is the number we are looking for. When we use $r+1$ edges from E_1, we force each constrain c_i to be infected from at least one variable in X. And by using n edges from E_2, each variable x_j will either be assigned true or false, but not both. We assign

$$x_i = \begin{cases} \text{true} & \text{if } x_i \text{ was ever infected} \\ \text{false} & \text{if } \bar{x}_i \text{ was ever infected} \end{cases}$$

and the result follows. □

We are now preperated to show that the SIS-problem is #P-hard. We will do this by reducing it to the V-SET SIS Connectedness
In this problem, we are given a digraph $G = (V, E)$ and $V' \subset V$. Create the a new

digraph $\tilde{G}$ by starting with G and add the vertex t and edges $\{(v,t)|v \in V'\}$. Let $N = (G, \mu, \rho)$ be SIS network, where

$$\mu(e) = \begin{cases} 2^{-1} & \text{if } e \in E \\ (1-p) & \text{else} \end{cases}$$

$$\rho(v) = 0 \text{ for all } v \in \tilde{V}$$

Let $A_i(t)$ denote the number of epidemics starting with infected in s and infects i vertices of V' at time t Let ϕ be a state such that

$$\phi(v) = \begin{cases} I & \text{if } v = s \\ S & \text{else} \end{cases}$$

Then the probability that t is infected at time $T+1$ is:

$$\begin{aligned} P(\Upsilon_t(v) = I | \Upsilon_0 = \phi) &= \sum_{i=0}^{|V'|} 2^{-|E|T} A_i(t)(1-p^i) \\ &= 2^{-|E|T} \sum_{i=0}^{|V'|} A_i(t)(1-p^i) \end{aligned}$$

which is the SIS problem. Since it is a polynomial of p, we can Theorem 2.8 deduce A_i by evaluating it for p at $|V'+1|$ points. The SIS problem can evaulate it at any value $p \in (0,1)$. $A_{|V'|}$ is precisely the answer of the V-SET SIS Connectedness, and the result follows.

THEOREM 6.5. *The SIS problem is #P-hard.*

CHAPTER 7

Algorithms for computing probability

Using computers for computing the probabilities for infection can be an important tool we want to investigate our hypotheses and theories. This may possibly also be used to falsify our claims, providing a simple yet powerful last verification of our theoretical work. We have taken the choice to write the algorithms in Python, using the NetworkX library for representing the graphs (See [**1**] for documentation).

The previous chapter have shown that computing the probabilities for the two networks are #P-hard. This means that for exact algorithms, it will have an exponential runtime. In the first section, we will create a algorithm, where we compute the exact probability. This will have a exponential running time. The second section, we will instead try to approximate the probabilities by Monte-Carlo simulation.

1. An exact algorithm

In the exact algorithm the idea is to find the probabilities by inspecting all the possible states. This algorithm have exponential running time, and have therefore no practical application.

1.1. Algorithm for SIR-graphs. Let N be a SIR-network.

Let $(\Psi_t)_{t\in\mathbb{N}}$ be the stochastic process with ϕ as the initial state. For the case $n = 0$, we get

$$P(\Psi_0(v) = I|\Psi_0 = \phi) = \begin{cases} 1 & \text{if } \phi(v) = I \\ 0 & \text{else} \end{cases}$$

Ψ_t is a Markov chain, and by Theorem 2.5 we have

$$P(\Psi_t = \varphi|\Psi_0 = \phi_0) = \sum_{\psi} P(\Psi_t = \varphi|\Psi_{t-1} = \phi)P(\Psi_{t-1} = \psi|\Psi_0 = \phi_0)$$

Then we may think of $G_1 = (V_1, E_1)$ as a digraph, where V_1 consists of all possible states of the network, and $e_{\phi,\psi}$ is an edge in E_1 iff ψ is a possible successor of ϕ. We may give each edge $e_{\phi,\psi}$ the probability $P(\Psi_t = \psi|\Psi_{t-1} = \phi)$.

We will now describe a variation of Breadth-First-Search on the graph G_1, starting with ϕ. Start by creating two dictionaries, one for the current state, and one for the next state. At time $t = 0$, the only state we have are ϕ with probability 1. In the beginning push $(\phi, 1)$ to the list of current states.

For each stage iterate through the current states, and find the successor states. If the successor state already exists, sum the probabilities. By proceeding in this manner, we get the successor states with its corresponding probabilities. Repeating this procedure t steps, we will have the successor states with its corresponding probabilities for every timestep.

Given a state ϕ with probability p we need to do the following to find all the successors φ with its probability $P(\Phi_t = \varphi | \Phi_{t-1} = \phi)$. Start by creating a queue, and add (ϕ, v_1, p) to the queue. While the queue is not empty, pop out the first element. This element is on the form (ψ, v_i, p), where ψ is the state, v_i is the vertex and p is the probability.

If $\phi_t(v_i) = R$, push the node (ψ, v_{i+1}, p) to the queue, since it is the only successor. If $\phi_t(v_i) = I$, create a new state ψ' where v is changed to R, and push the node (ψ', v_{i+1}, p) to the queue, since the only successor is the one turning I to R. If $\phi_t(v_i) = S$, then it can have two states. It can risk to be infected by the predecessors, or stay suspected. Equation 3.1 gives us the probability that vertex v_i gets infected. Let this probability be called q. Let ϕ' be the state where v_i becomes infected. Here we push the nodes $(\psi, v_{i+1}, p(1-q))$ and (ψ', v_{i+1}, pq) to the queue. If $i \geq |V|$ then we have inspected all the nodes, and ψ is a new state. We can add this to the dictionary of next states, with the corresponding probability p.

The return value of the algorithm is then $T \times |V|$ matrix $P = [p_{t,v}]$ where $p_{t,v} = P(\Psi_t(v) \in \{I, R\} | \Psi_0 = \phi)$. We is compute by the following equation

$$P(\Psi_t(v) \in \{I, R\} | \Psi_0 = \phi) = \sum_{\psi | \psi(v) \in \{I,R\}} P(\Psi_t = \psi | \Psi_0 = \phi)$$

This holds since by the second property of a measure function.

The resulting code is

```
1  from networkx import *
2  from numpy import zeros
3
4  S = 0
5  I = 1
6  R = 2
7
8  def find_probability(G, T, phi):
9      """
10     G is a graph in NetworkX.
11     The vertices are numbered 0 to n-1
12
13     Attributes of G:
14         G[i][j]['mu'] - every edges has a probability for transmission
15         G[i]['phi'] = I/S, the initial state.
16
17     T - number of timesteps to proceed
18
19     returns lists P where
20     P[t][n] = probability that node n has ever been infected at time t
21     """
22     N = len(G.nodes())
23     P = zeros((T+1,N))
24
25     P[0] = phi
26
```

```
27      current_states = {}
28      next_states = {}
29      current_states[tuple(phi)]  = 1
30
31      for t in range(T):
32
33          for state, p in current_states.items():
34              new_states = [state]
35              at_index = [0]
36              states_p = [p]
37
38              while(len(new_states)>0):
39                  cur_state = list(new_states.pop(0))
40                  index = at_index.pop(0)
41                  prob = states_p.pop(0)
42
43                  if index == N:
44                      #create new state
45                      new_state = tuple(cur_state)
46                      if new_state in next_states:
47                          next_states[new_state]+=prob
48                      else:
49                          next_states[new_state]=prob
50                      continue
51
52                  if cur_state[index] is R:
53                      new_states.append(tuple(cur_state))
54                      at_index.append(index+1)
55                      states_p.append(prob)
56                      continue
57
58                  if cur_state[index] is I:
59                      cur_state[index] = R
60                      new_states.append(tuple(cur_state))
61                      at_index.append(index+1)
62                      states_p.append(prob)
63                      continue
64
65                  if cur_state[index] is S:
66                      all_fail = 1.0
67                      for v in G.predecessors(index):
68                          mu = G[v][index]['mu']
69                          if state[v] is I and mu>0:
70                              all_fail *= G[v][index]['mu']
71
72
73                      #not infect
74                      new_states.append(tuple(cur_state))
75                      at_index.append(index+1)
76                      states_p.append(all_fail*prob)
77
78                      #infect
79                      if all_fail < 1:
80                          cur_state[index]=I
81                          new_states.append(tuple(cur_state))
82                          at_index.append(index+1)
83                          states_p.append((1-all_fail)*prob)
84
85          #sums up the probabilities and
86          for st, prob in next_states.items():
87              for i in range(N):
88                  if st[i] is I or st[i] is R:
89                      P[t+1][i]+=prob
90
```

```
91          # preparing the next state
92          current_states = next_states
93          next_states = {}
94
95      return P
```

1.2. Algorithm for SIS-graphs. Let $(\Upsilon_t)_{t\in\mathbb{N}}$ be the stochastic process with ϕ as the initial state. For the case $n = 0$, we get

$$P(\Upsilon_0(v) = I|\Upsilon_0 = \phi) = \begin{cases} 1 & \text{if } \phi(v) = I \\ 0 & \text{else} \end{cases}$$

Υ_t is a Markov chain, and by Theorem 2.5 we have

$$P(\Upsilon_t = \varphi|\Upsilon_0 = \phi_0) = \sum_{\psi} P(\Upsilon_t = \varphi|\Upsilon_{t-1} = \phi)P(\Upsilon_{t-1} = \psi|\Upsilon_0 = \phi_0)$$

As for SIR networks, we may think of $G_1 = (V_1, E_1)$ consisting of all possible states of the network, and $e_{\phi,\psi}$ is an edge in E_1 iff ψ is a possible successor of ϕ. We can use the same modified Breath-First-Search as for the case of SIR networks, but change out the successor part.

Let us look at how we can find the successors of a vertex ϕ with probability p. Start by creating a queue, and add (ϕ, v_1, p) to the queue. While the queue is not empty, pop out the first element. This element is on the form (ψ, v_i, p), where ψ is the state, v is the vertex and p is the probability.

If $\phi_t(v_i) = S$, then it can have two states. It can risk being infected by its predecessors or stay suspected. From Equation 3.3, we have the probability that v_i gets infected. Let this probability be called q. Let ϕ' be the state where v_i gets infected. Here we push the nodes $(\psi, v_{i+1}, p(1-q))$ and (ψ', v_{i+1}, pq) to the queue. If $\phi_t(v_i) = I$, then it may stay infected with probability $(1-\delta(v_i))$ or get cured with probability $\delta(v_i)$. Therefore we can push $(\psi, v_{i+1}, p(1-\delta(v))$ to the queue. The other alternative is that it gets cured, but risk to get infected at the same timestep. Let ψ' be the state wehere v_i is changed to S. We push the node $(\psi', v_i, p\delta(v))$ to the node. If $i > |V|$ is a number higher than number of nodes, then we have inspected all the nodes, and ψ is a new state. We can add this to the dictionary of next states, with the corresponding probability p.

The return value we want to give out is $T \times |V|$ matrix $P = [p_{t,v}]$ where $p_{t,v} = P(\Upsilon_t(v) = I|\Upsilon_0 = \phi)$. We can compute that by the following equation

$$P(\Upsilon_t(v) = I|\Upsilon_0 = \phi) = \sum_{\psi|\psi(v)=I} P(\Upsilon_t = \psi|\Upsilon_0 = \phi)$$

The resulting code is

```
1  from networkx import *
2  from numpy import zeros
3
4  S = 0
5  I = 1
6
```

```
7  def find_probability(G, T, phi):
8      """
9      G is a graph in NetworkX.
10     The vertices are numbered 0 to n-1
11
12     Attributes of G:
13         G[i][j]['mu'] - every edges has a probability for transmission
14         G.node[i]['rho'] - every node has a probability for getting cured
15         G[i]['phi'] = I/S, the initial state.
16
17     T - number of timesteps to proceed
18
19     returns list P where
20     P[t][n] = probability that node n is infected at time t
21     """
22
23     N = len(G.nodes())
24     start_state = phi
25     P = zeros((T+1,N))
26     P[0] = phi
27
28     current_states = {}
29     next_states = {}
30     current_states[tuple(phi)]  = 1
31
32     for t in range(T):
33
34         for state, p in current_states.items():
35             new_states = [state]
36             at_index = [0]
37             states_p = [p]
38
39             while(len(new_states)>0):
40                 cur_state = list(new_states.pop(0))
41                 index = at_index.pop(0)
42                 prob = states_p.pop(0)
43
44                 if index == N:
45                     #create new state
46                     new_state = tuple(cur_state)
47                     if new_state in next_states:
48                         next_states[new_state]+=prob
49                     else:
50                         next_states[new_state]=prob
51                     continue
52
53                 if cur_state[index] is I:
54                     delta = G.node[index]['rho']
55
56                     #not cured
57                     new_states.append(tuple(cur_state))
58                     at_index.append(index+1)
59                     states_p.append(prob*(1-delta))
60
61                     # cure
62                     cur_state[index]=S
63                     prob*=delta
64
65                 if cur_state[index] is S:
66                     all_fail = 1.0
67                     for v in G.predecessors(index):
68                         mu = G[v][index]['mu']
69                         if state[v] is I and mu>0:
70                             all_fail *= G[v][index]['mu']
```

```
71
72
73                      #not infect
74                      new_states.append(tuple(cur_state))
75                      at_index.append(index+1)
76                      states_p.append(all_fail*prob)
77
78                      #infect
79                      if all_fail < 1:
80                          cur_state[index]=I
81                          new_states.append(tuple(cur_state))
82                          at_index.append(index+1)
83                          states_p.append((1-all_fail)*prob)
84
85          for st, prob in next_states.items():
86              for i in range(N):
87                  if st[i] is I:
88                      P[t+1][i]+=prob
89
90          current_states = next_states
91          next_states = {}
92
93      return P
```

2. An approxmation algorithm

In this section we will create an approximation algorithm using Monte-Carlo simulation. The Monte-Carlo method is described in the second section of Chapter 3. This algorithm have in contrast to the exact algorith, a polynomial running time.

2.1. Algorithm for SIR-graphs. We are interested in the epidemic that starts in $\Psi_0 = \phi$. In Chapter 4, we argued that Ψ_t is determined by the previous state Ψ_{t-1} and the edges in play for that state. Let $e_1, ..., e_k$ be the edges that is in play at state Ψ_{t-1}. The random variable X_{e_i} has two possible outcomes 1 and 0, which can be simulated by inverse transform sampling. By these random variables we can generate the stochastic process $(\Upsilon_t)_{t=0,...,T}$ with initial state ϕ.

Let $Y_t = \chi_{I,R}(\Psi_t(v))$, where $\chi_{I,R}$ is the indicator function returning 1 if the input is either I or S. We note that

$$\begin{aligned} E[Y_t] &= 1 \times P(\Psi_t : \Psi_t(v) \in \{I, R\}) + 0 \times P(\Psi_t : \Psi_t(v) = S) \\ &= P(\Psi_t : \Psi_t(v) \in \{I, R\}) \end{aligned} \tag{7.1}$$

Let S_t be the average of n drawings of Y_t. Then by Theorem 2.1

$$P(|S_t - E[Y_t]| \geq \epsilon) \leq \frac{Var(Y_t)}{n\epsilon^2}$$

We can estimate the variance by

$$Var(Y_t) = E((Y_t - E[Y_t])^2) \leq (1+1)^2 = 4$$

This means that S_t converges to $P(\Psi_t : \Psi_t(v) \in \{I, R\})$ as n gets large. The return value we want to give out is $T \times |V|$ matrix $P = [p_{t,v}]$ where $p_{t,v} = P(\Psi_t(v) \in \{I, R\} | \Psi_0 = \phi)$. This induces the following algorithm:

```
from networkx import *
from random import *
from numpy import zeros

S = 0
I = 1
R = 2

def find_probability(G, T, M, phi):
    """
    G is a graph in NetworkX.
    The vertices are numbered 0 to n-1

    Attributes of G:
        G[i][j]['mu'] - every edges has a probability for transmission
        G.node[i]['phi'] = I/S, the initial state.

    T - number of timesteps to proceed
    M - number of trials

    returns list P where
    P[t][n] = probability that node n has ever been infected at time t
    """
    phi = tuple(phi)
    N = len(G.nodes())
    P = zeros((T+1,N))

    start_state = tuple(phi)
    P[0] = phi

    current_states = [start_state for i in range(M)]
    next_states = []

    for t in range(T):
        for state in current_states:
            state1 = [S for i in range(N)]
            for i in range(N):
                if state[i] is R:
                    state1[i] = R

                elif state[i] is I:
                    state1[i] = R
                    for v in G.successors(i):
                        mu = G[i][v]['mu']
                        if state[v] is S and random() <= mu:
                            state1[v]=I
            state1 = tuple(state1)
            next_states.append(state1)

        for i in range(N):
            for state in next_states:
                if state[i] is I or state[i] is R:
                    P[t+1][i]+=1.

            P[t+1][i]/=M

        current_states = next_states
        next_states = []

    return P
```

2.2. Algorithm for SIS-graphs. In a SIS-model things are more complicated. In additional to edges that transmit diseases, we may also have vertices that can cure themselves. We are interested in the epidemic that starts in $\Upsilon_0 = \phi$. In Chapter 4, we argued that Υ_t is determined by the previous state Υ_{t-1} and the edges and vertices in play for that state. Let $e_1, ..., e_k$ be the edges that is in play , and let $v_1, .., v_m$ be the vertices in play at state Ψ_{t-1}. The random variables X_{e_i} and X_{v_j} have two possible outcomes 1 and 0, and can both be simulated by inverse transform sampling. By these random variables we can generate the stochastic process $(\Upsilon_t)_{t=0,...,T}$ with initial state ϕ.

Let $Y_t = \chi_I(\Upsilon_t(v))$. We note that $E[Y_t] = P(\Upsilon_t : \Upsilon_t(v) = I)$ by similar argument as Equation 7.1. Let S_t be the average of n drawings of Y_t. Then by Theorem 2.1

$$P(|S_t - E[Y_t]| \geq \epsilon) \leq \frac{Var(Y_t)}{n\epsilon^2}$$

We can estimate the variance by

$$Var(Y_t) = E((Y_t - E[Y_t])^2) \leq (1+1)^2 = 4$$

This means that S_t converges to $P(\Upsilon_t : \Upsilon_t(v) = I\})$ as n gets large.

The return value we want to give out is $T \times |V|$ matrix $P = [p_{t,v}]$ where $p_{t,v} = P(\Upsilon_t(v) = I | \Upsilon_0 = \phi)$.

This induces the following code:

```
from networkx import *
from random import *
from numpy import zeros

S = 0
I = 1

def find_probability(G, T, M, phi):
    """
    G is a graph in NetworkX.
    The vertices are numbered 0 to n-1

    Attributes of G:
        G[i][j]['mu'] - every edges has a probability for transmission
        G[i]['delta'] - every node has a probability for getting cured
        G[i]['phi'] = I/S, the initial state.

    T - number of timesteps to proceed
    M - number of trials

    returns list P where
    P[t][n] = probability that node n is infected at time t
    """

    N = len(G.nodes())
    start_state = phi
    P = zeros((T+1,N))
    P[0] = phi

    current_states = [tuple(phi) for i in range(M)]
    next_states = []

    for t in range(T):
        for state in current_states:
            state1 = [S for i in range(N)]
```

```
36                for i in range(N):
37                    if state[i] is I:
38                        delta = G.node[i]['rho']
39
40                        if random() <= (1-delta):
41                            state1[i] = I
42
43                        for v in G.successors(i):
44                            mu = G[i][v]['mu']
45                            if random() <=  mu:
46                                state1[v]=I
47
48                state1 = tuple(state1)
49
50                next_states.append(state1)
51
52            for i in range(N):
53                for state in next_states:
54                    if state[i] is I:
55                        P[t+1][i]+= 1.
56                P[t+1][i]/=M
57
58            current_states = next_states
59            next_states = []
60
61        return P
```

3. Algorithm for expected value of infected

In a SIR network, it would be useful to know the total number of infected/removed vertices at a given time t. Assume that we have the $P = [p_{t,v}]$ matrix where $p_{t,v} = P(\Upsilon_t(v) \in \{I, R\})$, and that $\Psi_0 = \phi$. Then

$$\begin{aligned} E(|\Psi_t|) &= E[\sum_{v \in V} \chi_{I,R}(\Psi_t(v))] \\ &= \sum_{v \in V} E[\sum_{v \in V} \chi_{I,R}(\Psi_t(v))] \\ &= \sum_{v \in V} P(\Psi_t(v) \in \{I, R\}) \end{aligned}$$

This means that we can find the expected number of infected/removed at time t by computing $\sum_{v \in V} p_{t,v}$.

Let us instead look at a SIS-network, where it is interesting to know the total number of infected at time t. Assume that we have the $P = [p_{t,v}]$ matrix where $p_{t,v} = P(\Upsilon_t(v) = I)$ and $\Upsilon_0 = \phi$, By a similar argument we get

$$E(|\Upsilon_t|) = \sum_{v \in V} P(\Upsilon_t(v) = I)$$

and the expected number of infected at time t can be computed by the formula $\sum_{v \in V} p_{t,v}$.

CHAPTER 8

Test runs

We have throughout this master come to results for the probabilities for infection. In the previous chapter, we described two algorithms that can compute the probabilites for the network. This chapter is dedicated to run some tests using these algorithms. We will show it both for the SIS and SIR network.

The random underlying graph we will be using is generated by Erdos-Renyi algorithm with 300 vertices and $p = 0.03$. The Erdos-Renyi algorithm is described in an article by Newman [**11**]. With the networkX library, we can easily find the largest eigenvalue of the adjancency matrix, which is $\lambda_{1,A} \approx 12.87$.

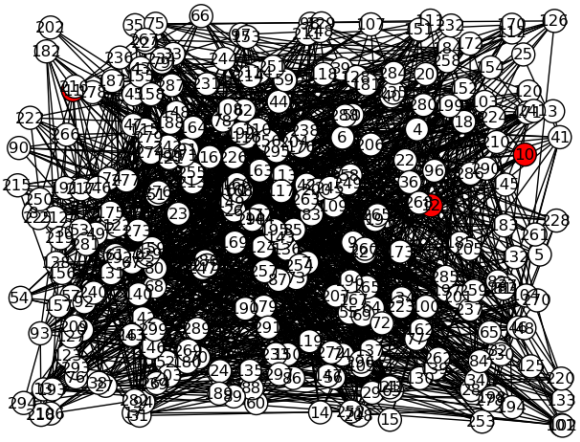

FIGURE 1. Random graph

In the first section, we do a test showing the first monotonicity property holds. In the second section, we do a test showing the second monotonicity property holds. In the third section, we do a test showing the epidemic threshold. In the forth section we do a test showing how an infection will develop with different vaccination strategies.

1. Testing the first monotonic property

Define ϕ_1, ϕ_2, ϕ_3 and ϕ_4 such that

$$\phi_1^{-1} = \{0\}$$
$$\phi_2^{-1} = \{0, 1, 2\}$$
$$\phi_3^{-1} = \{0, 1, 2, 3, 4\}$$
$$\phi_4^{-1} = \{0, 1, 2, 3, 4, 5, 6\}$$

Then

$$\phi_1 \leq \phi_2 \leq \phi_3 \leq \phi_4$$

For the case of the SIR network we will by Theorem 4.1(i)(The first monotonicity property for SIR networks) have

$$P(\Psi_t(v) \in \{I, R\} | \Psi_0 = \phi_i) \leq P(\Psi_t(v) \in \{I, R\} | \Psi_0 = \phi_{i+1})$$

Figure 2 shows the computation with the different ϕ_i, with SIR-graphs.

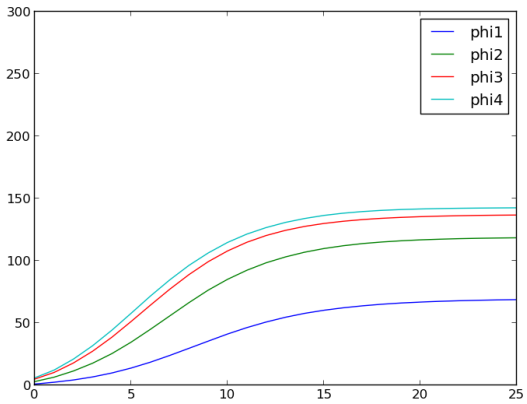

FIGURE 2. SIR network: Expected number of infected/resistant vertices at time t with different startstate ϕ_i

For the case of the SIS network we will by 4.4(i)(The first monotonicity property for SIS networks) have

$$P(\Upsilon_t(v) = I | \Upsilon_0 = \phi_i) \leq P(\Upsilon_t(v) = I | \Upsilon_0 = \phi_{i+1})$$

Figure 3 shows the computation with the different ϕ_i, with SIS-graphs.

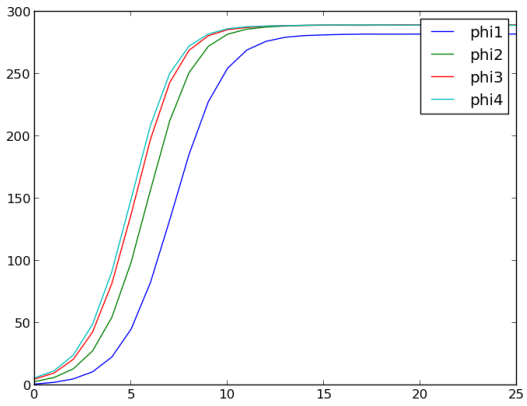

FIGURE 3. SIS network: Expected number of infected vertices at time t with different startstate ϕ_i

2. Testing the second monotonicity property

Pick 20 random edges in G an denote the set with E_1. Let $p_0 = 0, p_1 = 0.2, p_2 = 0.4, p_3 = 0.6, p_4 = 0.8, p_5 = 1$. And assume that for both the SIR and SIS network, the infection starts in state ϕ such that

$$\phi^{-1}(I) = \{10, 11, 12\}$$

We start showing it for a SIR network Let $N_i = (G, \mu_i)$ such that

$$\mu_i(e) = \begin{cases} p_i & \text{if } e \in E_1 \\ 0.150 & \text{else} \end{cases}$$

By Theorem 4.1(ii)(The second monotonicity property for SIR networks), we have

$$P(\Psi_{t,N_i}(v) \in \{I, R\}) \leq P(\Psi_{t,N_{i+1}}(v) \in \{I, R\})$$

Figure 4 shows the computation with the different ϕ_i, for the SIR network,

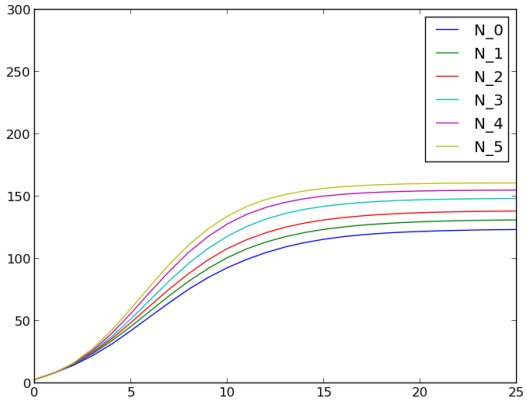

FIGURE 4. SIR network: Expected number of infected vertices at time t with different networks N_i

For the case of the SIS-network, let Let $N_i = (G, \mu_i, \rho)$ such that

$$\mu_i(e) = \begin{cases} p_i & \text{if } e \in E_1 \\ 0.08 & \text{else} \end{cases}$$

and $\rho(v) = 0.7$ for all $v \in V$.

By Theorem 4.1(ii)(The second monotonicity property for SIS networks), we have

$$P(\Upsilon_{t,N_i}(v) = I) \leq P(\Upsilon_{t,N_i}(v) = I)$$

Figure 5 shows the computation with the different ϕ_i, for the SIR network,

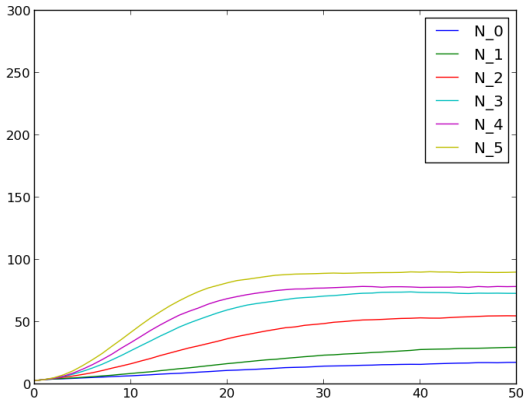

FIGURE 5. SIS network: Expected number of infected vertices at time t with different networks N_i

3. Testing epidemic threshold

The threshold is the condition determining if an epidemic outbreak will occur or not. We note that since $\lambda_{1,A} \approx 12.87$ this means $\lambda_{1,A}^{-1} \approx 0.0778$. Define the initial states ϕ_1 and ϕ_2 such that

$$\phi_1^{-1}(I) = \{11, 12, 13\}$$
$$\phi_2^{-1}(I) = V$$

We start with the SIR network. The theory says that when $\beta < \lambda_{1,A}^{-1} \approx 0.0778$, there is a high probability that number of vertices that will ever be infected is small by Theorem 5.1. Figure 6 shows the number of infected for different values of β such that $\mu(e) = \beta$.

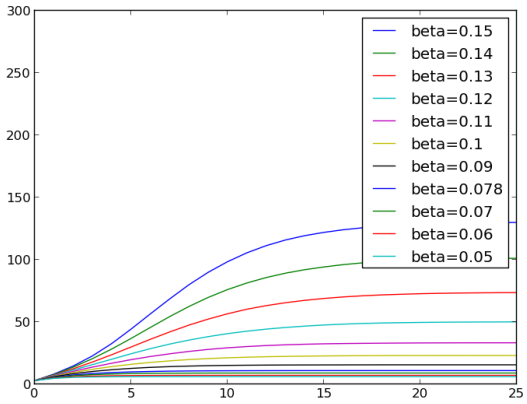

FIGURE 6. SIR network: Expected number of infected/removed vertices at time t with initial state ψ for different values of β

Now, let us take the SIS netowkr. We fixate $\delta = 0.7$ and variates β and plot the expected number of infected over time. The Theorem 5.2 says that when $\beta/\delta < \lambda_{1,A}^{-1} \approx 0.0778$, the number of infective will go to 0 when t is becomes large. Figure 7 shows how the epidemic evolves for different values of β such that $\mu(e) = \beta$.

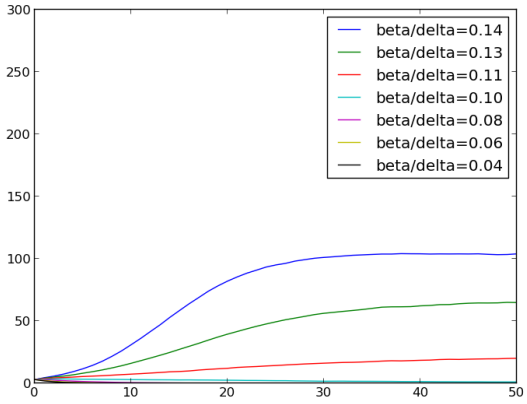

FIGURE 7. SIR network: Expected number of infected vertices at time t with initial state ψ for different values of β/δ

In the last section of Chapter 5, there is a comment telling that even when $\beta/\delta > \lambda_{1,A}^{-1}$, this doesn't mean that the infection will survive. We will use the same graph, but let φ such that $\varphi(v) = I$ for all v be the initial state. Then by the first monotonicity properties for SIS-graphs, we have that for any states ψ, then $\psi \leq \phi$ and thus

$$P(\Upsilon_t(v) = I | \Upsilon_0 = \varphi) \leq P(\Upsilon_t(v) = I | \Psi_0 = \psi)$$

Figure 8 shows that for $\beta/\delta = 0.10 > \lambda_{1,A} = 0.778$, the epidemic will still die out for all possibilities of ϕ.

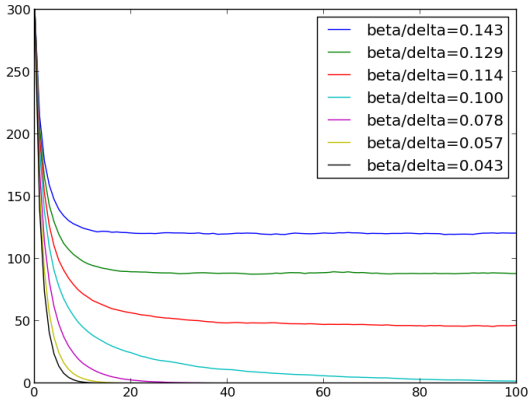

FIGURE 8. SIR network: Expected number of infected vertices at time t with initial state φ for different values of β/δ

4. Testing removing strategy

In the last section of Chapter 5 we have a small discussion of different strategies for removing vertices. Here we have plotted the different methods with the development of the epidemic after removing 10 vertices. In method 1, we are picking ten random vertices and remove them from the graph. In method 2, we first pick ten random vertices. For each of the vertex, we follow a random chain of successors of the vertex. The 8th successor of the vertex, we remove. In method 3, we find the vertex that creates the biggest drop of eigenvalue $\lambda_{1,A}$. This we repeat ten times. Figure 9 shows the strategies applied for the SIR-network. Figure 10 shows the strategies applied for the SIS-network.

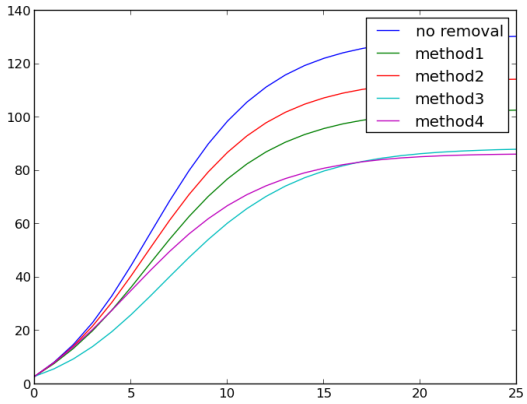

FIGURE 9. SIR network: Expected number of infected/removed vertices at time t for different removing strategy

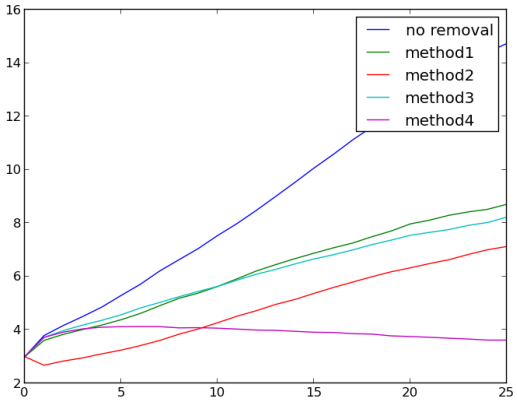

FIGURE 10. SIS network: Expected number of infected vertices at time t for different removing strategy

CHAPTER 9

Further research

We have showed that computing the probabilites for infection in SIR and SIS networks is #P-hard, and will probably not find any exact algorithm which performs better than exponential running time. If we do, this entail a major breakout in computer science, since this show that all the problems in NP and #P can be solved in polynomial time. Therefore the Monte-Carlo algorithm is the best solution for computing the probabilies of infections. The next step would be to parallel the program using Monte-Carlo algorithm such that we can find the probabilites for infection for greater networks.

During the time of this master, a great amount of time have been spend trying to find a formula for finding the formula for probability for infection for SIR networks with an underlying fan og wheel graphs. The method being used was by reducing it to smaller fan and wheel graphs. Unfortunately using this formula had a expontential run time, was complicated and not very useful, and this work have therefore been ommited in this thesis. An interesting question for further research is whether or not the computation of probabilites for such graphs also is #P-hard.

Vaccination strategies is a difficult field, and we have barely touched the surface. This is definitely a field that needs more research. We have showed monotonicity properties holds and there exists a epidemic threshold for the SIR and SIS network. We may prove these theories to the extension where we let the transmission probabilies and cureness probability vary over time. We may also look at more advanced models: The SI_kS-network is a model where a vertex can have several infective states before it becomes resistant. Then a vertex may stay suspected or goes through the following states: $S \to I_1 \to ... \to I_k \to R$. (see [**6**] for monotonicity properties of this model). The $SIRS$-network is a model where a vertex either stay suspected, or goes through the following states: $S \to I \to R \to S$. If there is a model where monotonicity properties does not hold , this means vaccination of individuals may create a greater epidemic.

Another interesting extension is what will happen if we drop the assumption that infection through edges are independent. The dependence of edges may be the case for adverticing. Instead of looking at infections and diseases, we may look at how an ad spreads from person to person. How will a message spread through friendsships, for example by buying a product. This can be illustrated by an epidemic network where the infection may be replaced owning the product. Instead of asking who we should vaccinate to lower the risk of infection, we rather in this case will ask who should we infect to create the largest epidemic. In this case we may take into account that a person will more likely buy the product, if the persons friends already have the product, and thus dependence of edges.

APPENDIX A

Code for testruns

1. Code for testrun showing first monotonic property

Code for finding the probabilites.

```
from networkx import *
import matplotlib.pyplot as plt
import time
import markovsir, markovsis
import pylab
import copy

N = 300
M = 1000

S = 0
I = 1
R = 2

G = erdos_renyi_graph(N, 0.03, seed=5334)


phi1 = [S for i in range(N)]
phi1[1] = I

phi2 = copy.deepcopy(phi1)
phi2[2] = I
phi2[3] = I

phi3 = copy.deepcopy(phi2)
phi3[4] = I
phi3[5] = I

phi4 = copy.deepcopy(phi3)
phi4[5] = I
phi4[6] = I

# SIR
G1 = DiGraph()
for x in G.nodes():
    G1.add_node(x)
for a,b in G.edges():
    G1.add_edge(a,b, {'mu': 0.15})
    G1.add_edge(b,a, {'mu': 0.15})


for name,phi in [('phi1', phi1), ('phi2', phi2), ('phi3', phi3), ('phi4',
    phi4)]:
    #construct random graph

    start = time.time()
```

```
46      P,P1 = markovsir.find_probability(G1,25, M, phi)
47      stop = time.time() - start
48      print "phi %s: time %s s" %(name, stop)
49      E = [sum(prob) for prob in P1]
50      print E
51      pylab.plot(E, label=name)
52  x1,x2,y1,y2 = plt.axis()
53  plt.axis((x1,x2,0,300))
54  pylab.legend()
55  pylab.savefig('sirmono.png')
56  #pylab.show()
57
58  # SIS
59  G1 = DiGraph()
60  for x in G.nodes():
61      G1.add_node(x,{'rho': 0.09})
62  for a,b in G.edges():
63      G1.add_edge(a,b, {'mu': 0.15})
64      G1.add_edge(b,a, {'mu': 0.15})
65
66  for name,phi in [('phi1', phi1), ('phi2', phi2), ('phi3', phi3), ('phi4',
        phi4)]:
67      #construct random graph
68
69      start = time.time()
70      P1 = markovsis.find_probability(G1,25, M, phi)
71      stop = time.time() - start
72      print "phi %s: time %s s" %(name, stop)
73      E = [sum(prob) for prob in P1]
74      print E
75      pylab.plot(E, label=name)
76  x1,x2,y1,y2 = plt.axis()
77  plt.axis((x1,x2,0,300))
78  pylab.legend()
79  pylab.savefig('sismono.png')
80  #pylab.show()
```

2. Code for testrun showing second monotonic property

Code for finding the probabilites.

```
1   from networkx import *
2   import matplotlib.pyplot as plt
3   import time
4   import markovsir, markovsis
5   import pylab
6
7   N = 300
8   M = 1000
9
10  S = 0
11  I = 1
12  R = 2
13
14  G = erdos_renyi_graph(N, 0.03, seed=5334)
15  color = ['white' for i in range(N)]
16  color[10] = 'red'
17  color[11] = 'red'
18  color[12] = 'red'
19  #draw_spring(G, node_color = color)
```

```
20 #pylab.savefig('randsis.png')
21 print adjacency_spectrum(G)
22
23
24 # SIR
25 mu = 0.150
26 for name,mu1 in [('N_0',0), ('N_1',0.2), ('N_2', 0.4), ('N_3', 0.6), ('N_4'
       , 0.8), ('N_5', 1)]:
27     #construct random graph
28     G1 = DiGraph()
29     for x in G.nodes():
30         G1.add_node(x)
31     for a,b in G.edges()[:-20]:
32         G1.add_edge(a,b, {'mu': mu})
33         G1.add_edge(b,a, {'mu': mu})
34
35     for a,b in G.edges()[-20:]:
36         G1.add_edge(a,b, {'mu': mu1})
37         G1.add_edge(b,a, {'mu': mu1})
38
39     phi = [S for i in range(N)]
40     phi[10] = I
41     phi[11] = I
42     phi[12] = I
43     start = time.time()
44     P,P1 = markovsir.find_probability(G1,25, M, phi)
45     stop = time.time() - start
46     print "Graph mu=%s: time %s s" %(mu1, stop)
47     E = [sum(prob) for prob in P1]
48     print E
49     pylab.plot(E, label=name)
50 x1,x2,y1,y2 = plt.axis()
51 plt.axis((x1,x2,0,300))
52 pylab.legend()
53 pylab.savefig('monosir2.png')
54 pylab.show()
55
56 # SIS
57 mu = 0.08
58 for name,mu1 in [('N_0',0), ('N_1',0.2), ('N_2', 0.4), ('N_3', 0.6), ('N_4'
       , 0.8), ('N_5', 1)]:
59     #construct random graph
60     G1 = DiGraph()
61     for x in G.nodes():
62         G1.add_node(x, {'rho': 0.7})
63     for a,b in G.edges()[:-20]:
64         G1.add_edge(a,b, {'mu': mu})
65         G1.add_edge(b,a, {'mu': mu})
66     for a,b in G.edges()[-20:]:
67         G1.add_edge(a,b, {'mu': mu1})
68         G1.add_edge(b,a, {'mu': mu1})
69
70
71     phi = [S for i in range(N)]
72     phi[10] = I
73     phi[11] = I
74     phi[12] = I
75     start = time.time()
76     P1 = markovsis.find_probability(G1,50, M, phi)
77     stop = time.time() - start
78     print "Graph mu=%s: time %s s" %(mu, stop)
79     E = [sum(prob) for prob in P1]
80     print E
81     pylab.plot(E, label=name)
```

```
x1,x2,y1,y2 = plt.axis()
plt.axis((x1,x2,0,N))
pylab.legend()
pylab.savefig('monosis2.png')
```

3. Code for testrun showing threshold

Code for finding the probabilites.

```
from networkx import *
import matplotlib.pyplot as plt
import time
import markovsir
import pylab

N = 300
M = 1000

S = 0
I = 1
R = 2

G = erdos_renyi_graph(N, 0.03, seed=5334)
color = ['white' for i in range(N)]
color[10] = 'red'
color[11] = 'red'
color[12] = 'red'
draw_spring(G, node_color = color)
pylab.savefig('randsis.png')
print adjacency_spectrum(G)


# SIR
for mu in [0.15, 0.14, 0.13, 0.12, 0.11, 0.1, 0.09, 0.078, 0.07, 0.06,
    0.05]:
    #construct random graph
    G1 = DiGraph()
    for x in G.nodes():
        G1.add_node(x)
    for a,b in G.edges():
        G1.add_edge(a,b, {'mu': mu})
        G1.add_edge(b,a, {'mu': mu})


    phi = [S for i in range(N)]
    phi[10] = I
    phi[11] = I
    phi[12] = I
    start = time.time()
    P,P1 = markovsir.find_probability(G1,25, M, phi)
    stop = time.time() - start
    print "Graph mu=%s: time %s s" %(mu, stop)
    E = [sum(prob) for prob in P1]
    print E
    pylab.plot(E, label="beta=%s"%mu)
x1,x2,y1,y2 = plt.axis()
plt.axis((x1,x2,0,300))
pylab.legend()
pylab.savefig('threshsir.png')
pylab.show()
```

```
51
52  # SIS
53  for mu in [0.1, 0.09, 0.08, 0.07, 0.0546, 0.04, 0.03]:
54      #construct random graph
55      G1 = DiGraph()
56      for x in G.nodes():
57          G1.add_node(x, {'rho': 0.7})
58      for a,b in G.edges():
59          G1.add_edge(a,b, {'mu': mu})
60          G1.add_edge(b,a, {'mu': mu})
61
62
63      phi = [S for i in range(N)]
64      phi[10] = I
65      phi[11] = I
66      phi[12] = I
67      start = time.time()
68      P1 = markovsis.find_probability(G1,50, M, phi)
69      stop = time.time() - start
70      print "Graph mu=%s: time %s s" %(mu, stop)
71      E = [sum(prob) for prob in P1]
72      print E
73      pylab.plot(E, label="beta/delta=%.2f"%(mu/0.7))
74  x1,x2,y1,y2 = plt.axis()
75  plt.axis((x1,x2,0,N))
76  pylab.legend()
77  pylab.savefig('threshsis.png')
78
79  #SIS 1
80  for mu in [0.1, 0.09, 0.08, 0.07, 0.0546, 0.04, 0.03]:
81      #construct random graph
82      G1 = DiGraph()
83      for x in G.nodes():
84          G1.add_node(x, {'rho': 0.7})
85      for a,b in G.edges():
86          G1.add_edge(a,b, {'mu': mu})
87          G1.add_edge(b,a, {'mu': mu})
88
89
90      phi = [I for i in range(N)]
91      start = time.time()
92      P1 = markovsis.find_probability(G1,100, M, phi)
93      stop = time.time() - start
94      print "Graph mu=%s: time %s s" %(mu, stop)
95      E = [sum(prob) for prob in P1]
96      print E
97      pylab.plot(E, label="beta/delta=%.3f"%(mu/0.7))
98  x1,x2,y1,y2 = plt.axis()
99  plt.axis((x1,x2,0,N))
100 pylab.legend()
101 pylab.savefig('threshsis1.png')
```

4. Code for testrun testing removal strategies

Code for finding the vertices removal that causes the greatest reduction of $\lambda_{1,A}$.

```
1  from networkx import *
2  import matplotlib.pyplot as plt
3  import time
4  import markovsir, markovsis
```

```
5  import pylab
6  import copy
7  from random import *
8  N = 300
9  M = 100
10
11 S = 0
12 I = 1
13 R = 2
14
15 mu = 0.150
16
17 G = erdos_renyi_graph(N, 0.03, seed=5334)
18
19 G1 = DiGraph()
20 for x in G.nodes():
21     G1.add_node(x)
22 for a,b in G.edges():
23     G1.add_edge(a,b, {'mu': mu})
24     G1.add_edge(b,a, {'mu': mu})
25
26 G4 = copy.deepcopy(G1)
27 print adjacency_spectrum(G4)[0]
28 for j in range(10):
29     print j
30     v = 0
31     spectrum = adjacency_spectrum(G4)[0]
32     for i in range(300):
33         G5 = copy.deepcopy(G4)
34         G5.remove_edges_from(G5.in_edges(i))
35         G5.remove_edges_from(G5.out_edges(i))
36         if spectrum > adjacency_spectrum(G5)[0]:
37             v = i
38             spectrum = adjacency_spectrum(G5)[0]
39
40     print "remove v=", v
41     print spectrum
42     G4.remove_edges_from(G4.in_edges(v))
43     G4.remove_edges_from(G4.out_edges(v))
```

Code for finding the probabilites.

```
1  from networkx import *
2  import matplotlib.pyplot as plt
3  import time
4  import markovsir, markovsis
5  import pylab
6  import copy
7  from random import *
8
9  N = 300
10 M = 5000
11
12 S = 0
13 I = 1
14 R = 2
15
16 G = erdos_renyi_graph(N, 0.03, seed=5334)
17 color = ['white' for i in range(N)]
18 color[10] = 'red'
19 color[11] = 'red'
```

```
20  color[12] = 'red'
21  #draw_spring(G, node_color = color)
22  #pylab.savefig('randsis.png')
23  phi = [S for i in range(N)]
24  phi[10] = I
25  phi[11] = I
26  phi[12] = I
27
28
29
30  ################
31  # SIR network #
32  ################
33
34  pylab.figure()
35  mu = 0.150
36
37
38  G1 = DiGraph()
39  for x in G.nodes():
40      G1.add_node(x)
41  for a,b in G.edges():
42      G1.add_edge(a,b, {'mu': mu})
43      G1.add_edge(b,a, {'mu': mu})
44
45  #no removal
46  P,P1 = markovsir.find_probability(G1,25, M, phi)
47  E = [sum(prob) for prob in P1]
48  pylab.plot(E, label='no removal')
49
50  G2 = copy.deepcopy(G1)
51  visited = [False for i in range(N)]
52  for i in range(10):
53      v = 0
54      while True:
55          v = randint(0,N-1)
56          if visited[v] is False:
57              visited[v] = True
58              break
59
60
61      G2.remove_edges_from(G2.in_edges(v))
62      G2.remove_edges_from(G2.out_edges(v))
63
64  # random
65  P,P1 = markovsir.find_probability(G2,25, M, phi)
66  E = [sum(prob) for prob in P1]
67  pylab.plot(E, label='method1')
68
69  visited = [False for i in range(N)]
70  G3 = copy.deepcopy(G1)
71  for i in range(10):
72      v = 0
73      while True:
74          v = randint(0,N-1)
75          if visited[v] is False:
76              visited[v] = True
77              break
78      succ = G3.successors(v)
79      if len(succ) > 0:
80          v = succ[randint(0, len(succ)-1)]
81      G3.remove_edges_from(G3.in_edges(v))
82      G3.remove_edges_from(G3.out_edges(v))
83
```

```
84  # random + chain
85  P,P1 = markovsir.find_probability(G3,25, M, phi)
86  E = [sum(prob) for prob in P1]
87  pylab.plot(E, label='method2')
88
89
90  visited = [False for i in range(N)]
91  G4 = copy.deepcopy(G1)
92  for i in range(10):
93      v = 0
94      while True:
95          v = randint(0,N-1)
96          if visited[v] is False:
97              visited[v] = True
98              break
99
100     succ = G4.successors(v)
101     for k in range(7):
102         if len(succ) > 0:
103             v = succ[randint(0, len(succ)-1)]
104             succ = G4.successors(v)
105     if len(succ) > 0:
106         v = succ[randint(0, len(succ)-1)]
107     G4.remove_edges_from(G4.in_edges(v))
108     G4.remove_edges_from(G4.out_edges(v))
109
110 # random + chain
111 P,P1 = markovsir.find_probability(G4,25, M, phi)
112 E = [sum(prob) for prob in P1]
113 pylab.plot(E, label='method3')
114
115
116
117
118 visited = [False for i in range(N)]
119 G5 = copy.deepcopy(G1)
120 for v in [152,158, 171, 219, 199, 95, 292, 31, 237, 140]:
121     G5.remove_edges_from(G5.in_edges(v))
122     G5.remove_edges_from(G5.out_edges(v))
123
124
125 P,P1 = markovsir.find_probability(G5,25, M, phi)
126 E = [sum(prob) for prob in P1]
127 pylab.plot(E, label='method4')
128
129 pylab.legend()
130 pylab.savefig('immunesir.png')
131 pylab.show()
132
133 ################
134 # SIS network #
135 ################
136
137 pylab.figure()
138 mu = 0.08
139 G1 = DiGraph()
140 for x in G.nodes():
141     G1.add_node(x, {'rho': 0.7})
142 for a,b in G.edges():
143     G1.add_edge(a,b, {'mu': mu})
144     G1.add_edge(b,a, {'mu': mu})
145
146
147
```

```
148 #no removal
149 P1 = markovsis.find_probability(G1,25, M, phi)
150 E = [sum(prob) for prob in P1]
151 pylab.plot(E, label='no removal')
152
153 visited = [False for i in range(N)]
154 G2 = copy.deepcopy(G1)
155 for i in range(10):
156     v = 0
157     while True:
158         v = randint(0,N-1)
159         if visited[v] is False:
160             visited[v] = True
161             break
162
163
164     G2.remove_edges_from(G2.in_edges(v))
165     G2.remove_edges_from(G2.out_edges(v))
166     G2.node[v]['rho'] = 1
167
168 # random
169 P1 = markovsis.find_probability(G2,25, M, phi)
170 E = [sum(prob) for prob in P1]
171 pylab.plot(E, label='method1')
172
173
174 visited = [False for i in range(N)]
175 G3 = copy.deepcopy(G1)
176 for i in range(10):
177     v = 0
178     while True:
179         v = randint(0,N-1)
180         if visited[v] is False:
181             visited[v] = True
182             break
183
184     succ = G3.successors(v)
185     if len(succ) > 0:
186         v = succ[randint(0, len(succ)-1)]
187     G3.remove_edges_from(G3.in_edges(v))
188     G3.remove_edges_from(G3.out_edges(v))
189     G3.node[v]['rho'] = 1
190
191 # random + chain
192 P1 = markovsis.find_probability(G3,25, M, phi)
193 E = [sum(prob) for prob in P1]
194 pylab.plot(E, label='method2')
195
196
197 visited = [False for i in range(N)]
198 G4 = copy.deepcopy(G1)
199 for i in range(10):
200     v = 0
201     while True:
202         v = randint(0,N-1)
203         if visited[v] is False:
204             visited[v] = True
205             break
206
207
208     succ = G4.successors(v)
209     for k in range(7):
210         if len(succ) > 0:
211             v = succ[randint(0, len(succ)-1)]
```

```
212                 succ = G4.successors(v)
213         if len(succ) > 0:
214             v = succ[randint(0, len(succ)-1)]
215         G4.remove_edges_from(G4.in_edges(v))
216         G4.remove_edges_from(G4.out_edges(v))
217         G4.node[v]['rho'] = 1
218
219     # random + chain
220     P1 = markovsis.find_probability(G4,25, M, phi)
221     E = [sum(prob) for prob in P1]
222     pylab.plot(E, label='method3')
223
224
225     G5 = copy.deepcopy(G1)
226     for v in [152,158, 171, 219, 199, 95, 292, 31, 237, 140]:
227         G5.remove_edges_from(G5.in_edges(v))
228         G5.remove_edges_from(G5.out_edges(v))
229         G5.node[v]['rho'] = 1
230
231
232     P1 = markovsis.find_probability(G5,25, M, phi)
233     E = [sum(prob) for prob in P1]
234     pylab.plot(E, label='method4')
235
236     pylab.legend()
237     pylab.savefig('immunesis.png')
238     pylab.show()
```

Bibliography

[1] Pieter Swart Aric Hagberg, Dan Schult. Networkx tutorial. `http://networkx.github.io/documentation/latest/_downloads/networkx_tutorial.pdf`.

[2] J.A. Bondy and U.S.R. Murty. *Graph theory*. Springer;, 3. edition, 2008.

[3] Deepayan Chakrabarti, Yang Wang, Chenxi Wang, Jurij Leskovec, and Christos Faloutsos. Epidemic thresholds in real networks. *ACM Transactions on Information and System Security (TISSEC)*, 10(4):1, 2008.

[4] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[5] Moez Draief, Ayaldavi Ganesh, and Laurent Massouli. Thresholds for virus spread on networks. 2008.

[6] William Floyd, Leslie Kay, and Michael Shapiro. Some elementary properties of sir networks or, can i get sick because you got vaccinated? *Bulletin of mathematical biology*, 70(3):713–727, 2008.

[7] Charles M. Grinstead and J. Laurie Snell. *Introduction to Probability*. American Mathematical Society, 2 revised edition edition, 1997.

[8] G. Ieyengard. Lecture 3 inverse transform method. `http://www.control.auc.dk/~henrik/undervisning/DES/lec03.pdf`.

[9] W.O. Kermack and A.G. McKendrick. A contribution to the mathematical theory of epidemics. 1927.

[10] David C. Lay. *Linear Algebra and Its Application*. Person Education, 3. edition, 2006.

[11] Mark EJ Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

[12] David Nualart. Stochastic processes. `http://www.mat.ub.edu/~nualart/StochProc.pdf`.

[13] Michael Shapiro and Edgar Delgado-Eckebert. Finding the probability of infection in an sir network is np-hard. *Mathematical Biosciences*, 2012.

[14] E.M. Stein and R. Shakarchi. *Real Analysis, Measure Theory, Integration, & Hilbert Spaces*, page 91. Princeton University Press, 2005.

[15] Gerald Teschl. *Topics in Real and Functional Analysis*. 2011.

[16] Leslie G. Valient. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.