Internal Report CS Bioinformatics Track 17-03 September 2017

Leiden University

Computer Science

Bioinformatics Track

Multiple Node Immunization On Complex Networks

Marios Kefalas

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Multiple Node Immunization on Complex Networks

Marios Kefalas

September 2017

Multiple Node Immunization on Complex Networks

Student:

Marios Kefalas
Student number: s1683187
E-mail : m.kefalas@umail.leidenuniv.nl
Leiden Institute of Advanced Computer Science

Supervisors:

Dr. Michael Emmerich
E-mail: m.t.m.emmerich@liacs.leidenuniv.nl
Leiden Institute of Advanced Computer Science

Prof. Dr. W.Th.F. den Hollander
E-mail: denholla@math.leidenuniv.nl
Leiden Mathematical Institute

September 2017

Contents

Acknowledgements

I would like to express my gratitude towards Dr. M. Emmerich and Prof. Dr.F. den Hollander for their views on the topics we studied, their help and the directions they suggested we follow in order to complete this thesis and most importantly their guidance and inspiration they conveyed through their classes in my two years in Leiden. I would also like to thank them for the patience they showed towards my myriad questions throughout this project.

Finally, I would like to thank my family, parents and sister, and dedicate this thesis to them, who without their love and support I would not have been able to do this Masters. Last but not least, I would like to thank, among others, my friends and fellow mathematicians and computer scientists in Leiden, Stelios Paraschiakos and Jorge (George) Vermoudas and in Greece, Georgia Zakata for their encouragement.

A paper containing results of chapter 4 and chapter 5, have been accepted to the IEEE Symposium Series on Computational Intelligence (IEEE SSCI 2017).

Abstract

Given a complex network G, like a social or computer network, which nodes should one immunize (or remove) to make it less vulnerable to virus attacks? This problem can be formulated as a subset selection problem, where the target is to select a subset of nodes to be immunized, in order to effectively prevent the spread of an epidemic. The drop of the largest eigenvalue (eigenvalue drop or eigen-drop) has been proven to be an effective measure for the impact of an immunization strategy, as it represents the network's vulnerability. It was additionally shown that the problem of selecting k out of n nodes from a network such that the eigenvalue drop is maximum belongs to the class of NP hard problems. Heuristic algorithms have been suggested to solve these problems approximately. Netshield algorithm was introduced in [8], a greedy approach that approximates the eigenvalue drop by means of a submodular function, the *shield-value*, and then maximizes the shield-value by means of a greedy approximation algorithm. In this thesis, we designed a problem specific genetic algorithm and compared it to Netshield+ – an improved variant of Netshield – and showed that on six moderate size problems from literature, their performance is competitive and often better. We also formulated a generalization of the k-Node Immunization Problem as a multiobjective problem, including the cost of immunization as a second objective. We present the first results on biobjective optimization, using multiobjective genetic algorithms as solvers. The method is demonstrated on the *USA domestic airline network* and the global city network of the *Pandemic cooperative board game*, which are enhanced by immunization cost data. In addition, first insights into the reliability of solvers and the typical shapes of Pareto fronts are obtained and discussed. Finally, we estimate through Continuous Time Markov Chain simulations the critical value for the infection rate λ on finite, square lattices in $\mathbb{Z}^2$ with percolation.

Chapter 1

Introduction

1.1 Problem Motivation

Imagine having a large cluster of servers that process millions of data or even more, regarding civilians' sensitive, personal information and imagine that this cluster suffers a cyber-attack that could distribute a phishing virus over the network. It is decisive that government agencies and policy makers know beforehand which servers to close down in order to suspend the magnitude of the attack and save as much information as possible. To take it even a step further, it is useful for law-enforcement agencies to have intelligence, given a social network of criminals or terrorists, on which individuals to neutralize so as to maximally scatter the network and also for health organizations to determine which citizens are more susceptible in contacting a disease and disseminating it further, resulting in a pandemic.

Immunization of a complex network, whether it is a social network or a computer network, is a crucial step in combating attacks, such as virus attacks (both digital and natural), rumours and other attacks over networks. In view of this, knowing in advance which nodes to immunize (or equivalently remove/quarantine) in order to counter the influence of the attack and stop it, is a key question in targeted immunization strategies, which aim in applying what is known as "herd-immunity" [1],[2].

1.2 Approach

The immunization of complex networks can be thought of as a subset selection problem, where the aim is to select a limited number of nodes to be immunized (removed or quarantined) in order to halt and stop the spread of an epidemic. There has been much attention on this matter, such as

[3],[4], [5], where various methods of node selection are discussed, like random immunization or acquaintance immunization. In [6], [7], this matter is approached from a more algebraic perspective; that of the largest eigenvalue of the adjacency matrix. In [8], Chen et al. argue that the eigenvalue drop or eigen-drop, which is the drop of the largest eigenvalue of the graph is a powerful measure for the impact of the immunization strategy, under the SIS epidemic model and proved that the problem of selecting k out of n nodes from a network, such that eigenvalue drop is maximum, belongs to the class of NP hard problems. In this view, they present Nethield and Netshield+, approximation algorithms that greedily solve this matter by means of a submodular function, the shield-value.

Due to the complexity class of this problem we believe that heuristic algorithms and particularly a class of metaheuristic algorithms, namely genetic algorithms, can prove to be a valuable asset in maximizing the eigen-drop and even outperform the Netshield algorithms in certain scenarios. Thus, in this project we follow this direction by designing problem specific genetic algorithms and comparing them with Netshield+, the efficient variant of Netshield. We make the comparison on six moderate-sized problems from literature; namely:

- Karate: Social network of friendships between 34 members of a karate club at a US university in the 1970s [9]. See figure A.1.
- Dolphins: Is a social network consisting of an undirected network of frequent associations between 62 dolphins in a community living off Doubtful Sound, New Zealand [10].
- US Flights: List of the most important Airports in the United States connecting one to another based on the existence of connecting flights (edge) from one port to the other ports. See figure A.3. Data taken from `kateto.net/network-visualization`.
- Pandemic: A cooperative board game with the goal to fight the outbreak of the virus. We used the graph that connects cities in the world as an example data set [11]. A picture of the Pandemic board is seen in Figure A.4.
- Conference Day 1: Social interaction of members of a conference on first day. Taken from here [12]. See figure A.5.
- Conference Day 3: From the same data set as above, but for the third day. See figure A.6.

We further formulate the node immunization problem as a multiobjective problem, by defining and including a cost function as the second objective. The method is demonstrated on the *USA domestic airline network* and the global city network of the *Pandemic cooperative board game*, which are augmented by immunization cost data. First insights into the reliability of solvers and the typical shapes of Pareto fronts are obtained.

We deal, finally, with a more theoretical aspect; that of the contact process on square lattices. We present work that has been done on the extinction time of an epidemic on graphs and we study the extinction time of an infection on various finite lattices on $\mathbb{Z}^2$ with percolation, in terms of steps under simulations of a Continuous Time Markov Chain (CTMC) model. We present the results demonstrated on $10 \times 10, 15 \times 15, 20 \times 20$ and 25×25 square lattices and we discuss open problems and future work.

1.3 Research Questions

The research questions discussed above, can thus be summarized as following:

1. Given an integer $k \in \mathbb{Z}_{>0}$. Can genetic algorithms effectively determine which node subsets achieve the highest eigenvalue drop and outperform Netshield+?

2. Defining a cost for immunizing each node, how can we determine a Pareto front by minimizing the cost and maximizing the eigenvalue drop, that can be useful for policy makers, by the use of a multiobjective genetic algorithm (EMOA)?

3. Estimating the *critical value* for the infection rate λ on finite, square lattices in $\mathbb{Z}^2$, on which percolation has happened.

   estimating the critical value for the infection rate λ on finite, square lattices in $\mathbb{Z}^2$ with percolation.

1.4 Overview

The rest of the thesis is organized as follows: In Chapter 2, we formally define the research questions 1 and 2, stated previously. In Chapter 3, we present Netshield and Netshield+ from [8]. In Chapter 4 we present the single-objective genetic algorithms and their results for research question 1 and in Chapter 5 the multiobjective genetic algorithms and the resulting Pareto

fronts for research question 2. We end Part 1 with Chapter 6 discussing our results and presenting future work. In Part 2, we begin by presenting the Contact Process in Chapter 7 and in Chapter 8 we discuss Percolation Theory. Finally in Chapter 9 we present our approach and results to research question 3 and we conclude with the discussion and future work in Chapter 10.

Part 1

Chapter 2

The Node Immunization Problem

In this Chapter we will briefly discuss the epidemic models SI, SIS, SIR. We will then formally define a network (or graph) and the eigenvalue drop before finally presenting rigorously the k-Node Immunization Problem, as well as the Multiobjective Optimization Problem.

2.1 Epidemiology

Epidemiology is the field of study of infectious disease spread across a population. The formation and spread of infectious diseases is a complex phenomenon with many interacting factors and agents. As a result, mathematical models have been designed to illustrate the establishment and spread of pathogens and to facilitate their study. Epidemiological models have applications also in the analysis of computer networks, for instance to study the transmission of messages and computer viruses through the internet and in social networks and to study the way a rumour is spread in Twitter or Facebook, for example. The foundations for such an approach, were set by Kermack and McKendrick in the early 1900s [13]. In the rest of this thesis when referring to an epidemiological model, we will mean a model of a virus spreading in a network.

These models are known as *compartmental models* in epidemiology, and serve as a basic mathematical framework for understanding and studying the complex nature of these systems. In the simplest scenario, the population can be classified into two states only: Infected (I) and Susceptible (S), if the probability of becoming infected is strictly positive. However, to make the scenario even more realistic a third label, R, for the population is included

which represents entities that are immune/recovered/removed.

With this terminology, we can define the following models:

- SI model: A population entity can get infected after being in a susceptible state. Once an entity is infected it keeps being infected and it can infect other entities. Such dynamics occur also in the spread of messages and information across computer networks.
- SIS model: In this model entities that are infected can recover and return to the susceptible state. After this though, they might get infected again. This resembles a flu-like spread.
- SIR model: Here, an infected entity can be removed from the network some time after its infection. Reasons for this could be that the infected entity got immunized, or the infected entity got isolated from the population, or the disease was lethal. This model reasonably resembles dynamics of infectious diseases which are transmitted between humans, and where recovery confers lasting resistance, such as measles, mumps and rubella.

There also many more classification schemes or compartments that will not be discussed here, as they are a field of study on their own. We point the interested reader however to [6], [7]. In the following we will deal with the SIS epidemic model.

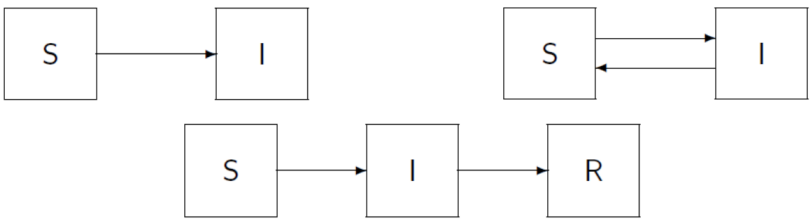

Figure 2.1: Three common models in epidemiology, In the SI model, nodes stay infected, once they got infected, In the SIS model, infected nodes can return in a susceptible state, and in the SIR model nodes are immunized after having recovered and can no longer infect neighboring nodes.

The study of these models is done through deterministic approaches, for example through the use of ordinary differential equations, but can also be viewed by a stochastic simulation framework which is more realistic. We will present the latter when simulating the contact process on finite lattices, later in Chapter 7.

2.2 Definitions

Before we proceed into the definition of the eigenvalue drop, we briefly give the definition of a network (graph) in its abstract, mathematical form.

Definition 1 *A network (also called a graph) is an ordered pair $G = (V, E)$ consisting of a set of vertices $V = \{v_1, \ldots, v_n\}$ (also called nodes or sites) and a set of edges $E \subseteq V \times V$ (also called links or bonds) connecting pairs of vertices [14].*

In essence, a graph is a structure amounting to a set of objects, in which some pairs of the objects are in some sense "related" [15]. Vertices and edges can also have weights, as well as directions . However, in this project we will not be using weights, except for the multiobjective approach in Section 5, where the weight of a node will be the cost of its immunization. We will also deal only with undirected and connected graphs. The latter means, that the there are no two nodes that cannot be reached by some path between them.

Graphs can be coded efficiently into what we call an adjacency matrix, the elements of which are either 1 if a link between two nodes is realized and 0 otherwise. The matrix representation of graphs is rather important since it allows for applying various algebraic techniques, as well as it is one way of coding them in computers.

Throughout this project we will be using the adjacency matrix, since it gives rise to the next definition; that of the the eigenvalue drop.

Definition 2 *Given a network G, let S be the subset of nodes chosen to be removed. Then G' is a subgraph of G with the nodes in S removed and their adjacent edges. The eigenvalue drop $\Delta\lambda$ is defined as the difference between the maximum eigenvalue of the adjacency matrix of G, λ_1 and the maximum eigenvalue of the adjacency matrix of G', λ_1'. That is $\Delta\lambda = \lambda_1 - \lambda_1'$.*

We would like to justify here the use of the eigenvalue drop as a measure. As mentioned also in the introduction, in [6] and [7] the largest eigenvalue λ is a good measure of how vulnerable the graph is. λ is closely related to the epidemic threshold τ of a network under an SIS epidemic model, via the formula $\tau = 1/\lambda$, since its threshold will be small. Thus, for larger λ an epidemic is more likely to be sustained. The way to overcome this is to lower the vulnerability of the network by minimizing λ and thus maximizing the epidemic threshold, which is equivalent to maximizing the difference ($\Delta\lambda$) between the largest eigenvalue of the initial network and the largest eigenvalue of the perturbed network.

We should also mention here, that because the adjacency matrix of a graph is symmetric, with non-negative entries, all of its eigenvalues are real numbers and there exists a non-negative eigenvalue λ which has maximum absolute value among all eigenvalues. This eigenvalue has also a non-negative real eigenvector (Perron–Frobenius theorem) [16]. Furthermore, the removal of nodes and their edges from a graph, is equivalent to deleting the respective rows and columns from its adjacency matrix. Then by the Poincare Separation Theorem (or Cauchy Interlacing Theorem) [16], [17], [18], we have that the largest eigenvalue of the sub-matrix will be at most equal to the largest eigenvalue of the original matrix, of the connected graph. Thus, $\Delta\lambda \geq 0$.

Finally, we will call *eigen-scores* the elements of the eigenvector corresponding to the largest eigenvalue of the graph. These components play a special role when designing the genetic algorithms.

In the next section we will formally define the single objective Node Immunization Problem, as well as the multiobjective optimization approach.

2.3 The Node Immunization Problem

We begin by defining the single objective optimization approach, where the aim is to maximize the eigenvalue drop, to decide which nodes to remove.

Problem 1 *Given a network $G = (V, E)$ with n nodes and $k \in \{1, 2, ..., n\} \subset \mathbb{N}$, the (single objective) k-Node Immunization Problem is the problem of determining a set $S \subseteq V : |S| = k$, to be removed from a network, such that the eigenvalue drop is maximal.*

Chen et al. proved in [8] that the k-Node Immunization Problem with the largest eigenvalue is NP hard. Due to this, heuristic algorithms are proposed in [8], namely Netshield and Netshield+, which are approximation algorithms. The key aspect of these algorithms is that they do not operate directly on the eigenvalue drop, but use an approximation of it. The approximation of the largest eigenvalue is a submodular function and therefor allows for the construction of greedy heuristics with guaranteed performance bounds, as we will discuss in Chapter 3.

Next we will define the multiobjective Node Immunization Problem, by introducing a cost function as the second objective to be optimized.

Problem 2 *Given a network $G = (V, E)$ with n nodes, let c_i be the cost of immunization of each node $i \in V$. The (multiobjective) Node Immunization*

Problem is the problem of determining a set $S \subseteq V$, to be removed from the network, such that the eigenvalue drop is maximal and the cost of immunization is minimal.

Note that in Problem 1, we know a priori the number k of nodes to be immunized. This is not the case, however, for Problem 2, which is why it reasonably resembles real world scenarios. Imagine for example if policy makers must close down airports to halt the spread of an infection. It might be wiser in terms of cost and eigenvalue drop to close down 10 "small" airports rather than 1 "big" one. In essence Problem 2, tries to determine the possible solutions in which there exists a tradeoff between two objectives to be optimized, by establishing an efficient set and the resulting Pareto front.

Chapter 3

Netshield Algorithm

In this Chapter we will present NetShield and Netshield+ (see Algorithms 1 and 2) from [8]. Some of the ideas of these algorithm will be useful in the design of the problem specific genetic algorithm. Moreover the Netshield+ algorithm, a more efficient version of the Netshield algorithm, will serve as a baseline algorithm in the benchmarking.

Netshield and Netshield+ are heuristic algorithms, designed to solve the k-Node Immunization Problem. These algorithms given $k \in \mathbb{Z}_{>0}$ and a graph $G = (V, E)$, greedily select nodes to determine the set $S \subseteq V : |S| = k$, which is to be removed from V. However, they do not directly operate on the eigenvalue drop, but rather use an approximation of it which is submodular and therefore lends itself for constructing an approximation algorithm. This submodular approximation is called *Shield-value*, $Sv(\cdot)$, and is define as follows:

$$Sv(S) = \sum_{i \in S} 2\lambda(u_i)^2 - \sum_{i,j \in S} a_{ij} u_i u_j$$

where S is the set of the nodes to be removed from V. Here u_i denotes the i-th element of the eigenvector that corresponds to the largest eigenvalue of the graph G. The Shield-value rewards dissimilarity between nodes, that is small a_{ij}, and nodes that have a high eigen-score.

It also proved in [8] that the Shield-value is a good approximation of the eigenvalue drop, when the first (largest in magnitude) eigenvalue λ_1 of the graph G is simple and also that $\delta \geq 2\sqrt{2kd}$, where $\delta = \lambda_1 - \lambda_2$ (eigen-gap) and d is the maximum degree of G. Given also the fact that $\lambda \leq d$, we get $\delta \leq d$ and we end up with the expression $k \leq d/8$. This constraint explicitly says that in order to get a good approximation of the eigenvalue drop with Shield-value, the cardinality of subset S cardinality should satisfy the inequality, which does not always hold when the maximum degree of the graph G is relatively small. To circumvent this issue and further

balance the optimization quality and the computational cost, [8] proposes Netshield+. As opposed to the Netshield algorithm however, the Netshield+ algorithm, removes nodes in batches of b-nodes each. After each batch the largest eigenvalue and the corresponding eigen-scores are recomputed. This way the algorithm yields more accurate results, but due to multiple eigenvalue computations the computation time increases, but is still linear with respect to the size of the input graph.

Algorithm 1 Netshield

Input: adjacency matrix A and integer k
Output: a set S with k nodes

1: compute the first eigenvalue λ_1 of A and u the corresponding eigenvector $u(j)(j = 1, ..., n)$
2: initialize S to be empty
3: **for** j=1:n **do**
4: $v(j) = (2 \cdot \lambda - A(j, j)) \cdot u(j)^2$
5: **end for**
6: **for** iter $= 1 : k$ **do**
7: let $B = A(:, S)$
8: let b = B$\cdot u(S)$
9: **for** j = 1:n **do**
10: **if** $j \in S$ **then**
11: let score(j) = -1
12: **else**
13: let score(j) $= v(j) - 2 \cdot b(j) \cdot u(j)$
14: **end if**
15: **end for**
16: let i $= argmax_j score(j)$, add i to set S
17: **end for**
18: return S

Netshield works as follows: In step 1 the first (largest) eigenvalue λ is computed, as well as the corresponding eigenvector u and in step 2 we initialize an empty set S which is the node subset. In step 4 the algorithm goes through all n nodes and computes the Shield-value score of each individual node. Afterwards in each iteration of steps 6-17 the algorithm **greedily** selects one more node and adds it to set S (step 16) according to the score in step 13. The intermediate steps 10-12 are used to exclude the nodes that are already in S.

Algorithm 2 Netshield+

Input: adjacency matrix A and integer k and integer b
Output: a set S with k nodes

compute the number of iterations needed $t = *k/b$
initialize S to be empty
for j=1:t **do**
    initialize S' to be empty
    $S' = Netshield(A, b)$
    $S = S \cup S'$
    update A by deleting the corresponding rows/columns indicated by the nodes in S'
end for
if $k > tb$ **then**
    $S' = Netshield(A, k - tb)$
    $S = S \cup S'$
end if
return S

In the next Chapters we present our results on the Node Immunization Problem. In Chapter 4 we discuss the single objective k-Node Immunizaion Problem and in Chapter 5 we present the multiobjective Node Immunization Problem.

Chapter 4

The k-Node Immunization Problem

This Chapter is the first part of our contribution. We will deal here with Problem 1, by designing problem specific genetic algorithms for single objective node selection and we will be comparing our results with those of Netshield+. The comparison will be made on six moderate-sized problems from literature (see page 7).

4.1 Motivation

As we mentioned previously, in [8] Chen et al. proved that the k-Node Immunization Problem is NP hard. In particular they proved that the problem is NP complete, and we know that these problems require time that is super-polynomial in the input size. For this reason it is meaningful to use heuristics. They allow for faster computation, but do not guarantee optimality. It is a necessary trade off. This is in contrary to approximation algorithms that involve a mathematical proof certifying the quality of their returned solutions in the worst case scenario. Netshield and Netshield+ are such algorithms.

We believe that for Problem 1, genetic algorithms could be a good candidate class of algorithms. Genetic algorithms are a particular instance of meta-heuristics, inspired by natural selection. They belong to the class of Evolutionary Algorithms. Evolutionary Algorithms are an **optimization** method based on the biological analogy of *survival of the fittest*. Genetic algorithms, typically are applied on discrete search spaces as opposed to Evaluation Strategies, which encode continuous vectors. In this project we designed genetic algorithms to deal with Problem 1.

In the next section we will briefly describe genetic algorithms to make

the reader acquainted with the notions discussed.

4.2 Genetic Algorithms

In this section we will briefly present Genetic Algorithms.

Genetic Algorithms (GAs) are a class of meta-heuristic algorithms which belong in the broader class of meta-heuristics called Evolutionary Algorithms (EAs). The latter, are a class of direct, probabilistic search and optimization algorithms, which have GAs as one of their main representatives.

Developed by Holland, a computer scientist and psychologist at the University of Michigan, in 1975 [19], in the form we know and use them, GAs intended to simulate biological systems to use natural selection (Darwin's survival of the fittest) to solve practical applications. His goal was broader, to develop adaptive systems that communicate with their environment and evolve [20].

Genetic algorithms have several key-components. These are problem representation, crossover or recombination, mutation and selection. In more detail, representation is crucial as it specifies in an encoded form the solutions to a particular problem. The choice of representation can affect the rest of the algorithm operators such as mutation. The crossover operator, which is one of the main aspects of a GA, allows for exploitation of the search space, by combining parts of the parent representation into the children, making it possible to exchange information between individuals and combine favourable, for the problem, characteristics. Crossover, thus, can be thought of as a basic reproductive procedure we know. However, as it also occurs in nature, mutations take place. Mutation operators are necessary for exploring the search space of a problem at hand. It allows for changes in the offspring that would not be possible by simply applying the recombination/crossover operators. Finally, the selection operator, determines which of the parents and/or offspring will play the role of the new parent population and so on. There are two popular schemes such as $(\mu + \lambda)$ and (μ, λ). The former means that we select as our new parent population the best (with respect to our problem) individuals from the union of children and parents, while the latter specifies that we select as our new parent population the μ best offspring out of the λ offspring generated. For more details we point the reader to [20].

In the next section, we will present the outline of our GAs and discuss the problem specific parameters.

4.3 GAs and Problem Specific Parameters

We will start by our first attempt which is a simple (1+1)-GA, noted as GA_0. Below we present its outline. G is the graph, maxEval is the integer indicating the maximum number of iterations or generations and k is the integer determining how many nodes to remove.

Algorithm 3 GA_0

Input: the graph G, maxEval, k
Output: a set S with k nodes

1: determine the adjacency matrix A of G
2: compute the first eigenvalue λ_1 of A
3: $t \leftarrow t$
4: $P(t) \leftarrow$ initialize population of a single individual randomly
5: $f(P(t)) \leftarrow$ evaluate the fitness value of the individual
6: **while** t $\leq$ maxEval **do**
7: $P'(t) \leftarrow$ mutate the individual with mutation probability p_m
8: $f(P'(t)) \leftarrow$ evaluate the fitness value of the mutated individual
9: $t \leftarrow t + 1$
10: $P(t) \leftarrow$select the best individual between the two as the next parent population
11: **end while**
12: **return** solution S with the highest eigenvalue drop

Function f here is the actual eigenvalue drop and not an approximation of it. GA_0 is a simple genetic algorithm, with no problem specific parameters, which operates as a basis for the other GAs we will discuss next.

Remark: Before we proceed it is necessary to discuss the solution representation and evaluation we used in GA_0, as well as how a solution is interpreted. These remarks also hold for all the other GAs we will present next.

Let's begin with the latter. Given a graph $G = (V, E)$ with $|V| = n$, we represent a solution s as a permutated sequence of integers in $\{1, 2, .., n\}$, where each integer represents a node of the graph G. That is index 1 represents node 1, index 2 represents node 2 and so on. For example $s = (1, 3, 5, 2, 6, 7, 9, 4, 8, 10)$, if $n = 10$, is a solution. This representation is not usual, but a problem specific representation for subset selection as it has also been used in other contexts too. See [21].

Given $k \in \{1, 2, ..., n\} \subset \mathbb{N}$, the first k components of the solution s are the nodes which need to be removed, in order to get the maximum eigenvalue

drop. A solution s is thus evaluated on the first k elements of s. This is implicit in the pseudo code above. That is $f(s) = \Delta\lambda(s[1:k])$, where we calculate the eigenvalue drop if we remove the nodes with indexes the first k elements of solution s.

Finally, the mutation is done as following for this representation. Given a solution s we mutate the solution k-times by selecting each time with probability p_m an element from $s[1:k]$ and with probability p_m an element from $s[k+1:n]$ and then interchanging them. We chose $p_m = 1/n$, as proposed by Bäck in [20].

We continue now, presenting the problem-specific GAs we designed.

Algorithm 4 $(\mu + \mu)$ - GA

Input: the graph G,maxEval, k, p_m
Output: a set S with k nodes

determine the adjacency matrix A of G
compute the first eigenvalue λ_1 of A; let u be the corresponding eigenvector and $u(j), j \in \{1, .., n\}$ the eigen-scores
$t \leftarrow t$
$P(t) \leftarrow$ initialize the population randomly with μ individuals
$f(P(t)) \leftarrow$ evaluate the fitness value of each individual in the population
while !(termination_criteria_met) **do**
    for $i \leq \mu$ **do**
        $p_1 \leftarrow$ select parent 1
        $p_2 \leftarrow$ select parent 2
        $c \leftarrow$ recombine parent 1 and parent 2 to create offspring c
        $c' \leftarrow$ mutate offspring c with probability p_m
        $f(c') \leftarrow$ evaluate the fitness value of the mutated offspring
    end for
    $t \leftarrow t + 1$
    $P(t) \leftarrow$ select the best μ individuals from the union of the μ offspring and μ parents, as the new parent population
end while
return solution S with the highest eigenvalue drop

The pseudo code of Algorithm 4 represents the general outline of our problem specific GAs. In more detail, the Shield-value formula defined in Chapter 3

$$Sv(S) = \sum_{i \in S} 2\lambda(u_i)^2 - \sum_{i,j \in S} a_{ij} u_i u_j$$

has a higher value if the nodes in the set $S \subseteq V$, where V is the set of nodes of graph G, have high eigen-scores and the nodes are dissimilar between each other [8]. We took the former into account to make a problem specific mutation operator for the GAs. In particular, we gave a larger mass/probability for the indexes of the nodes with the k highest eigen-scores to be included in the first k elements of each solution s in the population, after mutation, which is done precisely as mentioned before. This way, our GAs are more focused on the part of the search space that is more likely to be relevant for solving the problem. We also feel that because we combine the greedy approach of Shield value with the actual eigenvalue drop function, there is an advantage in determining the nodes which yield the largest eigenvalue drop. The different mutation rates we used for the k components of u with the highest eigen-score were $p_m = 2/n, 3/, 6/n, 1$ and for all other nodes the mutation probability was set to $1/n$. For completeness reasons we also included the $p_m = 1/n$ in the set above. The choices for p_m were arbitrary and based upon the idea that the larger probabilities should be proportional to the basic $1/n$. Regarding the parent selection scheme, we used the classic scaled proportional selection.

For recombination we did the following procedure: Since the representation is not binary we could not use a usual recombination operator. Let $G = (V, E)$ be a graph, with $|V| = n$ and let $k \in \mathbb{Z}_{>0}$. Let also p_1, p_2 be the two parents selected by scaled proportional selection. We calculated the union $T = p_1[1 : k] \cup p_2[1 : k]$. If $|T| > k$ we randomly discard nodes so as to have $|T| = k$. Then the offspring will have in its first k positions, the elements of T and the remaining $n - k$ components of the offspring will be filled by the complement of the set T in $\{1, 2, ..., n\}$, to avoid any duplicates. Afterwards, a randomness is introduced by the mutation operator.

Moreover, the evaluation of the solutions is done as mentioned previously for Algorithm 1. In addition we should also mention that we used a static population size of $\mu = 50$ and a recombination probability of $p_c = 0.75$ [20].

Finally, we used an additional termination criterion to the maximum number of generations. That is, if the best fitness value remained stagnant for more number of generations than $k(n - k)$, the algorithm should end its search. We adopted this, keeping in mind that in a solution s the possible number of mutations is proportional to $k(n - k)$.

In the next section we will describe our experiments and present our results.

4.4 Experiments and Results

We will start this section by describing the experiments we carried out before moving to the presentation of the results. We will note as GA_0 the single parent GA, presented in Algorithm 3 and as GA_1, GA_2, GA_3, GA_4, GA_5 the GA presented in Algorithm 4 with $p_m = 1/n, 2/n, 3/n, 6/n, 1$ respectively.

We ran each GA on every network, 20 times, with maximum number of generations maxEval = 30000. We did this for $k = 1, 2, 3, 5, 10$. The choices for k were arbitrary and were based mostly on the idea that our networks are relatively small, thus the number of nodes to remove should not be large. Furthermore, coding of the algorithms was done in the *RStudio* environment[1] with *igraph* package[2]. The experiments we executed on the following machines of the LIACS Data Science Lab:

- Latinum: 16 Intel Xeon E5-2630v3 CPUs @ 2.40GHz (32 threads) 1.5TB RAM
- Duranium: 20 Intel Xeon E5-2650v3 CPUs @ 2.30GHz (40 threads) 128GB RAM
- Tritanium: 20 Intel Xeon E5-2650v3 CPUs @ 2.30GHz (40 threads) 1TB RAM

We would like to point out here that we used instead of function evaluations, number of generations in the GA's as a termination criterion. In the $1 + 1$ of course these two notions are equivalent. However, this is not the case for the population-based GA's. The reason for this is that these GA's can be parallelized, using at least μ processors.

We will start now by presenting the results in boxplots. Each boxplot presents the Eigen_drop against the Class of algorithms, that is GA_0, GA_1, GA_2, GA_3, GA_4, GA_5 and Netshield_plus representing Netshield+. In each figure starting from upper left to lower right, we have $k = 1, 2, 3, 5, 10$.

Next, in Table 4.1 we give the median results for all Networks and for all algorithms.

[1]Version 1.0.136 – © 2009-2016 RStudio, Inc.
[2]Version 1.0.1, `http://igraph.org/r/`

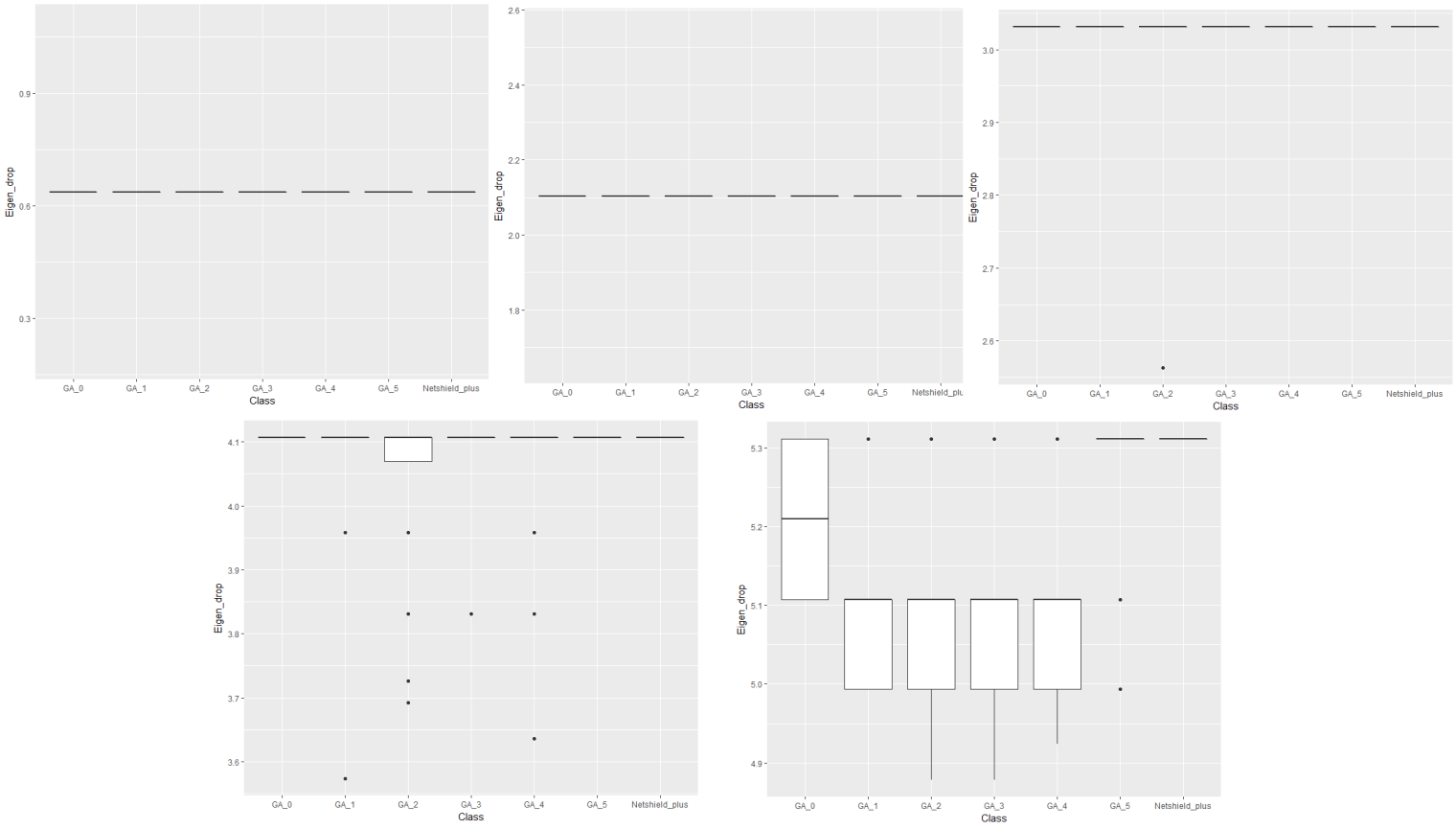

Figure 4.1: Results for the Karate Network.

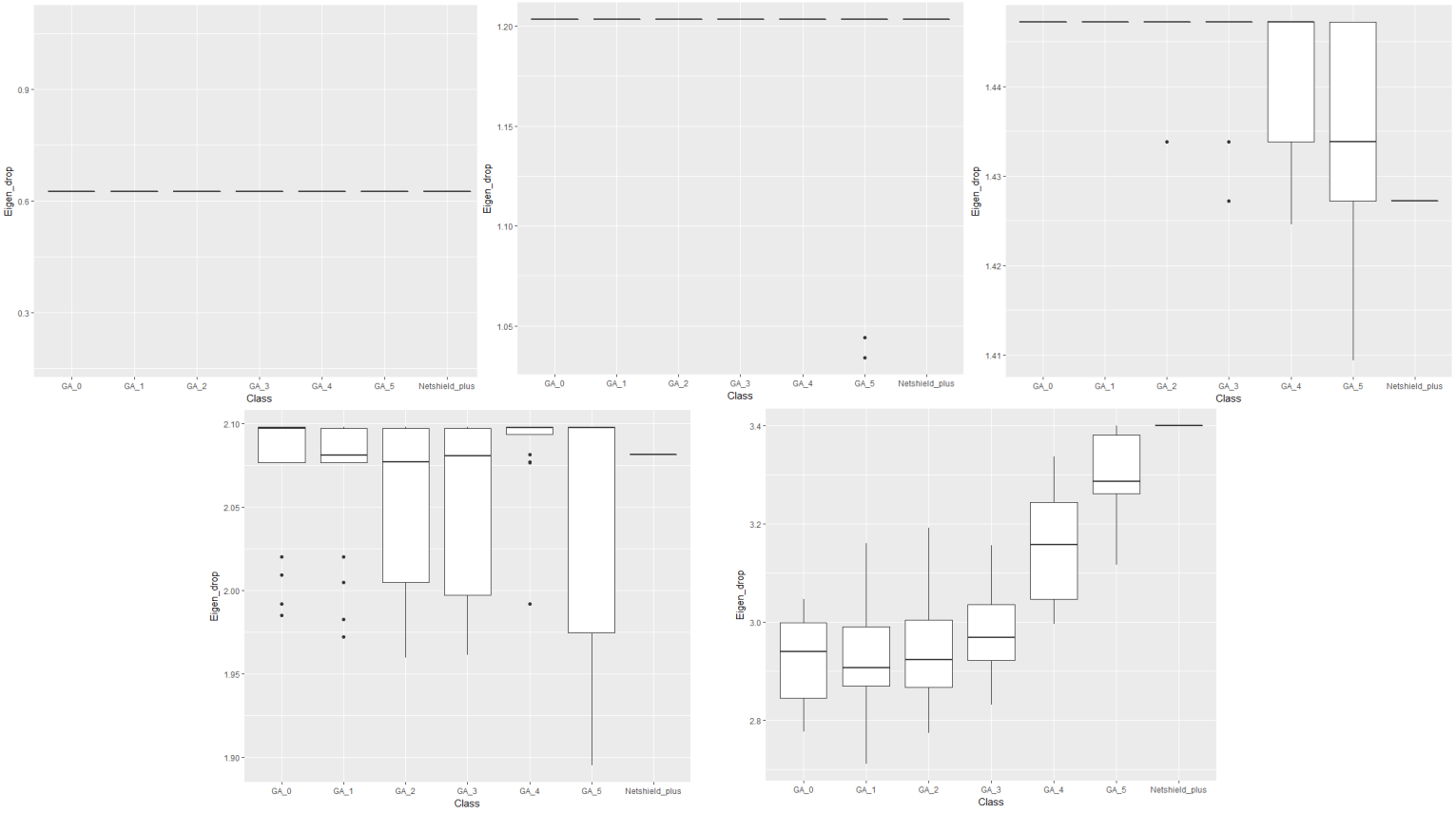

Figure 4.2: Results for the Dolphins Network.

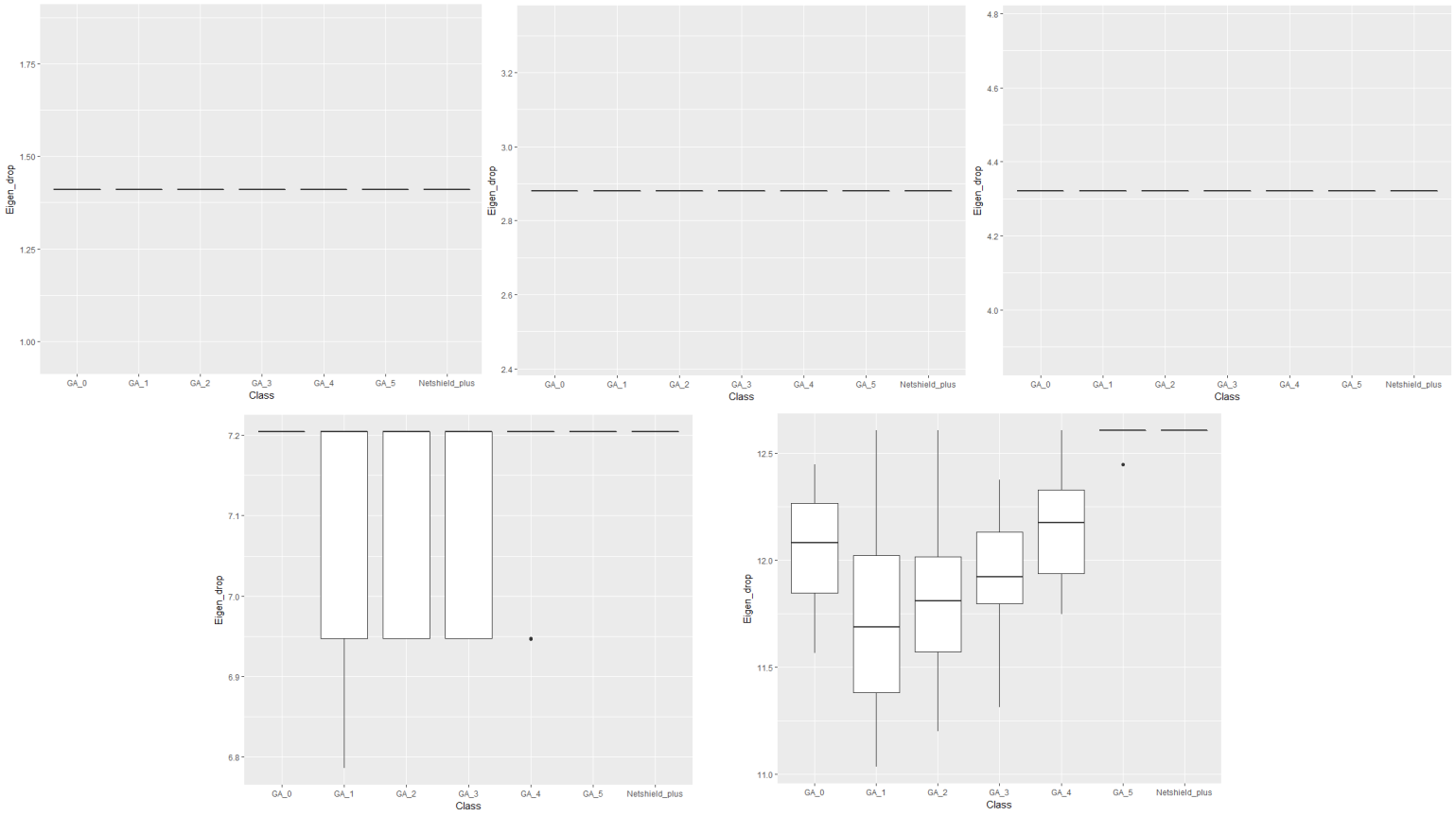

Figure 4.3: Results for the USA Network.

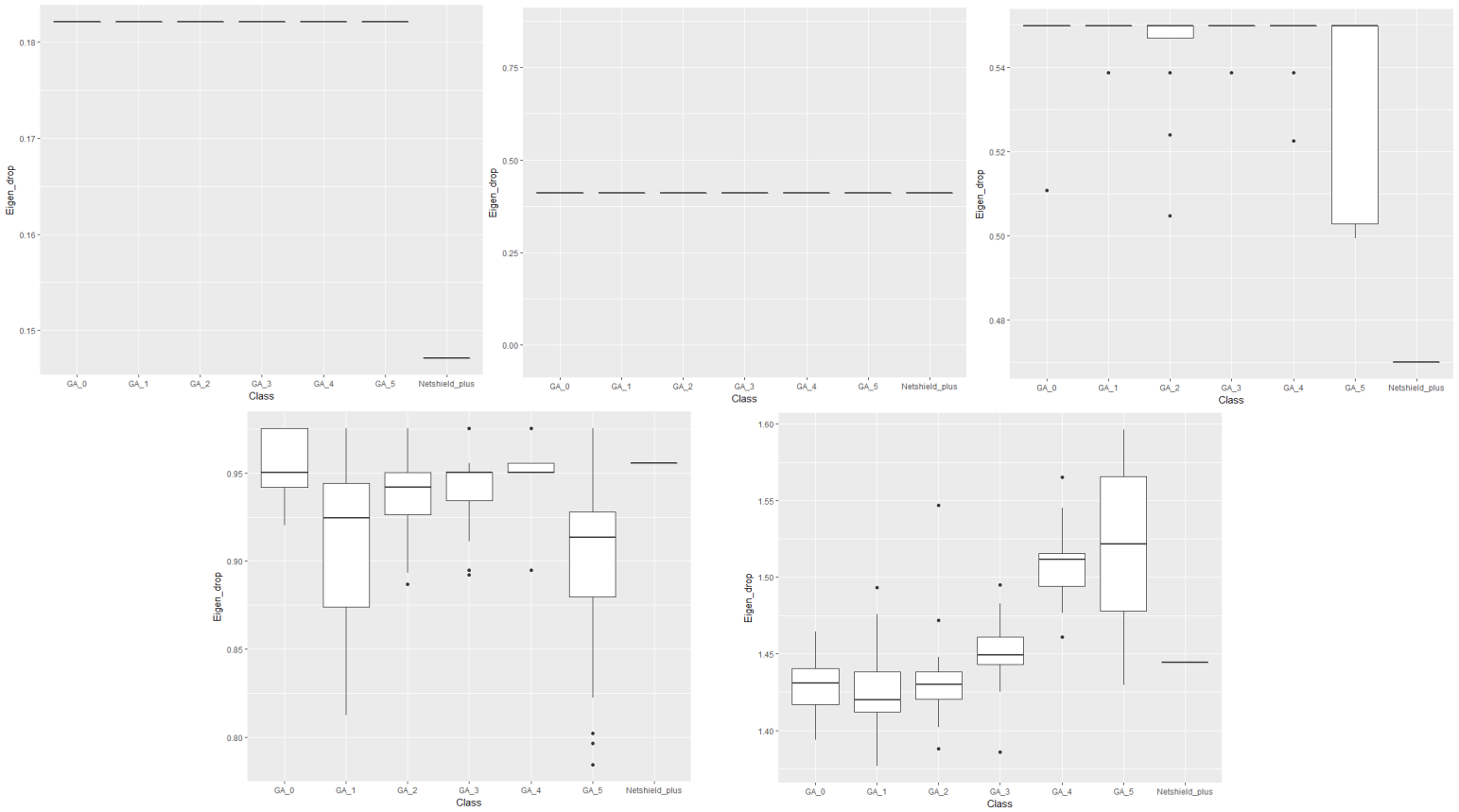

Figure 4.4: Results for the Pandemic board game Network.

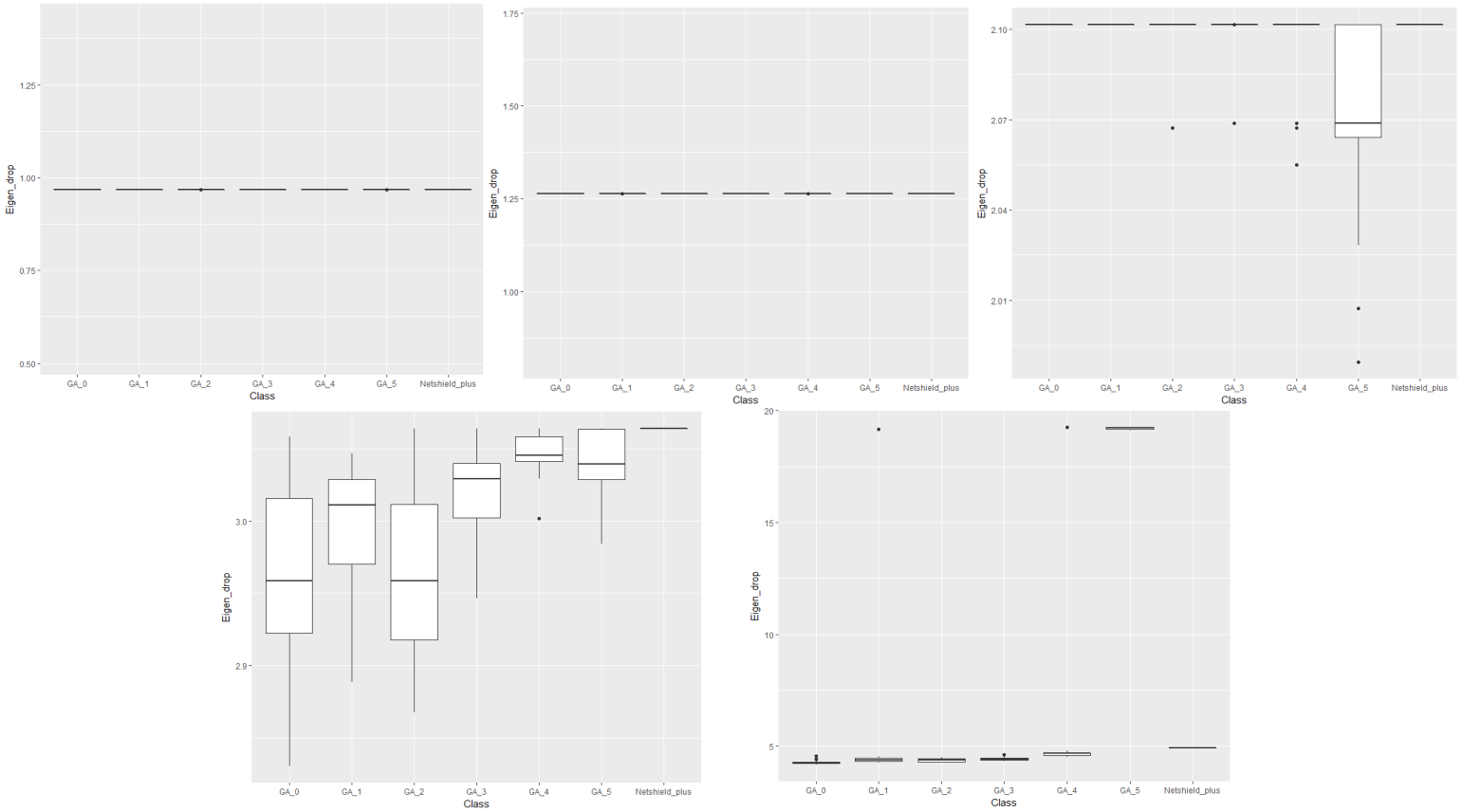

Figure 4.5: Results for the Conference day 1 Network.

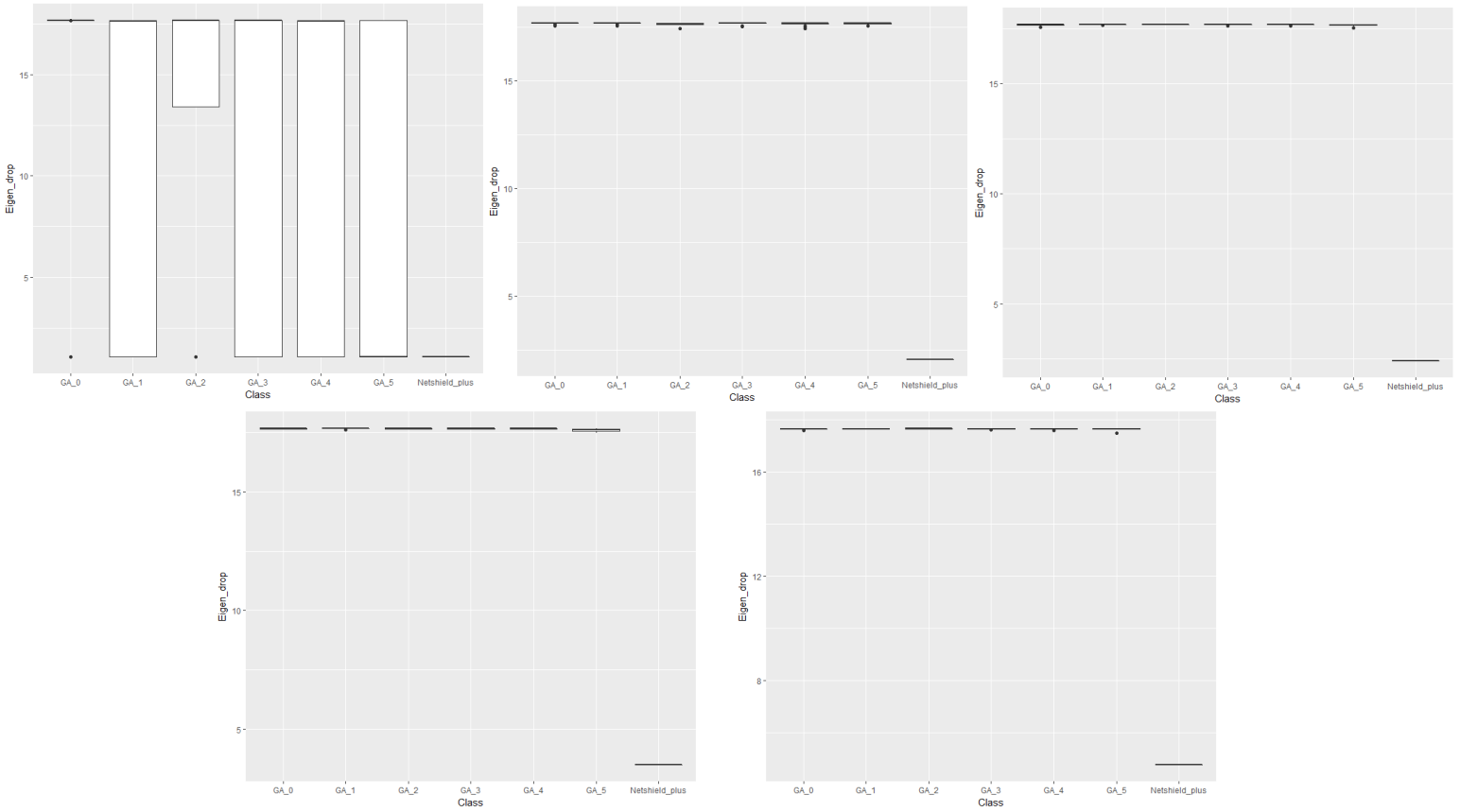

Figure 4.6: Results for the Conference day 3 Network.

Table 4.1: Median results for all Networks

	Networks	GA_0	GA_1	GA_2	GA_3	GA_4	GA_5	NetshieldPlus
k = 1	karate	0.637663	0.637663	0.637663	0.637663	0.637663	0.637663	0.637663
	Dolphins	0.625875	0.625875	0.625875	0.625875	0.625875	0.625875	0.625875
	USA	1.410999	1.410999	1.410999	1.410999	1.410999	1.410999	1.410999
	Pandemic	0.182128	0.182128	0.182128	0.182128	0.182128	0.182128	0.147066
	Conf. day 1	0.968081	0.968081	0.968081	0.968081	0.968081	0.968081	0.968081
	Conf. day 3	17.67357	17.62927	17.67347	17.67260	17.65165	1.059852	1.059852
k = 2	karate	2.103674	2.103674	2.103674	2.103674	2.103674	2.103674	2.103674
	Dolphins	1.203444	1.203444	1.203444	1.203444	1.203444	1.203444	1.203444
	USA	2.880762	2.880762	2.880762	2.880762	2.880762	2.880762	2.880762
	Pandemic	0.411588	0.411588	0.411588	0.411588	0.411588	0.411588	0.411588
	Conf. day 1	1.264015	1.264015	1.264015	1.264015	1.264015	1.264015	1.264015
	Conf. day 3	17.6732	17.67276	17.6566	17.67241	17.6681	17.67013	2.058068
k = 3	karate	3.031487	3.031487	3.031487	3.031487	3.031487	3.031487	3.031487
	Dolphins	1.447190	1.447190	1.447190	1.447190	1.447190	1.433805	1.427201
	USA	4.321868	4.321868	4.321868	4.321868	4.321868	4.321868	4.321868
	Pandemic	0.549761	0.549761	0.549761	0.549761	0.549761	0.549761	0.470017
	Conf. day 1	2.101622	2.101622	2.101622	2.101622	2.101622	2.068796	2.101622
	Conf. day 3	17.67019	17.67149	17.67142	17.67166	17.67112	17.65112	2.412332
k = 5	karate	4.106751	4.106751	4.106751	4.106751	4.106751	4.106751	4.106751
	Dolphins	2.097431	2.08117	2.076892	2.080667	2.097751	2.097751	2.081674
	USA	7.204260	7.204260	7.204260	7.204260	7.204260	7.204260	7.204260
	Pandemic	0.950231	0.924342	0.941926	0.9502309	0.950231	0.91331	0.955593
	Conf. day 1	2.958561	3.010937	2.95832	3.028930	3.045471	3.039115	3.063814
	Conf. day 3	17.65924	17.67044	17.67136	17.66936	17.66959	17.60980	3.854225
k = 10	karate	5.209574	5.107664	5.107664	5.107664	5.107664	5.311484	5.311484
	Dolphins	2.940289	2.907666	2.923008	2.968493	3.157546	3.286219	3.399719
	USA	12.08268	11.68963	11.80947	11.92225	12.17715	12.60777	12.60777
	Pandemic	1.431002	1.420122	1.429910	1.449002	1.511400	1.521515	1.444183
	Conf. day 1	4.252082	4.385349	4.383103	4.420728	4.669685	19.23729	4.912077
	Conf. day 3	17.64637	17.65904	17.66447	17.65775	17.65886	17.65800	5.648326

4.5 Conclusion

The study presented here shows that genetic algorithms often perform better, if not significantly better, in solving the k-Node Immunization Problem. Netshield+ is a fast, greedy, approximation algorithm that produces in many

cases good results. Based on our findings depicted above, we believe that an efficient strategy to solve this problem could be, if time permits, using not only Netshield+ but also the problem specific genetic algorithm to make it more probable that the best solution for the eigenvalue drop objective is found.

In order to achieve good results, problem specific tuning turned out to be very useful. The idea we used and which works well is to use eigen-score values in order to adjust the mutation probabilities. This way the search is directed in areas of the search space that is more likely to be relevant for solving the problem. We should also emphasize here that the supplementary use of a problem specific genetic algorithm has the advantage of calculating the actual eigen-drop, rather than an approximation of it. This can be useful for moderate sized networks. However, in large networks the computational cost increases, since the algorithm eigendecomposes larger adjacency matrices.

Chapter 5

The Multiobjective Node Immunization Problem

In this Chapter, we discuss Problem 2, a generalization of the k-node immunization problem, which we defined in Chapter 2 (paragraph 3). The motivation behind the study of this problem is that it resembles closely real world situations, where the objectives can be conflicting. In actual world scenarios it is likely that multiple nodes need to be controlled or immunized, however the number of nodes is not known a priori. What is known, though, is the cost for the immunization of each node.

We studied a biobjective node immunization problem with the eigenvalue drop as the first objective and the cumulated costs of immunization of each node, as the second. The latter is defined as follows: Let c_i be the cost of each node $i \in V$, where V is the set of nodes of graph G. The cost of immunization of set a $S \subseteq V$ is then

$$C(S) = \sum_{i \in S} c_i$$

which clearly is to be minimized, while the eigenvalue drop is to be maximized. The problem is formulated as:

$$f_1(S) := \Delta\lambda(S) \to \max$$
$$f_2(S) := C(S) \to \min$$
$$S \subseteq V$$

We are interested in the efficient set of this problem, that is the set: $\mathcal{S}_E = \{S \subseteq V = \{1, ..., n\} | \ \nexists S' \subseteq V = \{1, \ldots, n\} : f_1(S') \geq f_1(S) \wedge f_2(S') < f_2(S) \vee f_1(S') > f_1(S) \wedge f_2(S') \leq f_2(S)\}$ and the Pareto front $\{(f_1(S), f_2(S))^T | \, S \in \mathcal{S}_E\}$.

In the next paragraph, we present the Experiments and the Results.

5.1 Experiments and Results

We will start this paragraph by describing the experiments we carried out before moving to the presentation of the results. We used two multiobjective genetic algorithms (EMOAs) as solvers. Namely the the nondominated sorting genetic algorithm NSGA-II [22], belonging in the class of Pareto based EMOAs and the the S-metric selection algorithm, SMS-EMOA [23], belonging in the class of indicator based EMOAs.

The implementations of SMS-EMOA and NSGA-II, were done in the *RStudio* environment, featured by Bossek's `ecr` package[3]. The representation of a subset S of the nodes V of given graph G is chosen to be a bit vector b in $\mathbb{B}^n$, where $b_i = 1$ means the node is selected to be removed/quarantined and $b_i = 0$ means the node is not selected, for $i = 1, \ldots, n$. As recombination operator, one point crossover is used. For all bits we used $p_m = 1/n$ as the mutation probability. The reason for this mutation rate is that, in contrast to the single objective genetic algorithms we discussed, here we do not know a priori the number of nodes to remove/quarantine. That is, we do not specify a subset cardinality. As a consequence the algorithm should not try to explore a particular direction of the search space (bias introduced from the mutation operator), but rather present to the decision makers a complete picture of their possible choices. For example, quarantining 10 less-important (in terms of eigen-score) airports could be more beneficial in terms of cost, than quarantining 1 important (in terms of eigen-score) airport.

As mentioned briefly in the introduction, the *USA domestic airline network* (see figure A.3), and the global city network of the *Pandemic cooperative board game* (see figure A.4), which are augmented by immunization cost data, served as examples for computing the efficient sets and the resulting Pareto fronts. In case of the Pandemic network the size of the cities was used as a cost, assuming that it is more difficult to immunize larger cities (see table A.3). In case of the US flights network the size of the airport (number of visits) was taken into account (see table A.2).

We would like to underline here that even though we aimed for real-world, problem settings, more modeling would be needed, such as social interactions, geographic environment, and various other factors, in order to plan an effective real-world immunization. In this study, we simply focus on the network aspects of the problem and we feel that it could be used as a basis for further studies on this or similar topics.

[3] `https://github.com/jakobbossek/ecr2` and `https://cran.r-project.org/web/packages/ecr/ecr.pdf`

Regarding the methodology of the experiments, each algorithm for the multiobjective optimization was run a total of 5 times, producing 5 Pareto front approximations. Moreover, we allowed for a maximum of 10000 function evaluations and we used as a parent selector for both classes of algorithms `selSimple` which randomly selects parents for recombination, the `recCrossover` operator which applies the one-point crossover recombinator and finally we used, as mentioned previously, the `mutBitFlip` operator as the bit-flip mutation operator, which flips each bit with a given probability.

Results for Pandemic are shown in Figure 5.1 and Figure 5.2. Results for USA Flight network are shown in Figure 5.3 and Figure 5.4.

Below are 5 Pareto fronts for the Pandemic board game network based on NSGA-II algorithm. In every figure we have included a sixth plot in the bottom right, combining all plots to serve as an indicator of the variance of the results.

5.2 Conclusion

Looking at the results, we observe that the NSGA-II algorithm obtained, overall, better results and displayed a more robust performance than the indicator based SMS-EMOA on this problem. It is also visible that the Pareto fronts looks near linear, especially for the USA Flights network. This might be explained by the fact that big nodes (larger cities or, respectively, airports) are at the same time expensive to immunize as well as crucial for immunization. We can also observe for the USA Flights network a knee-like region can be identified, which is a preferable by decision makers solution [24].

In Figure 5.5 we can see both solvers for both networks. In black + we represent the NSGA-II algorithm and in red $\triangle$ the SMS-EMOA algorithm.

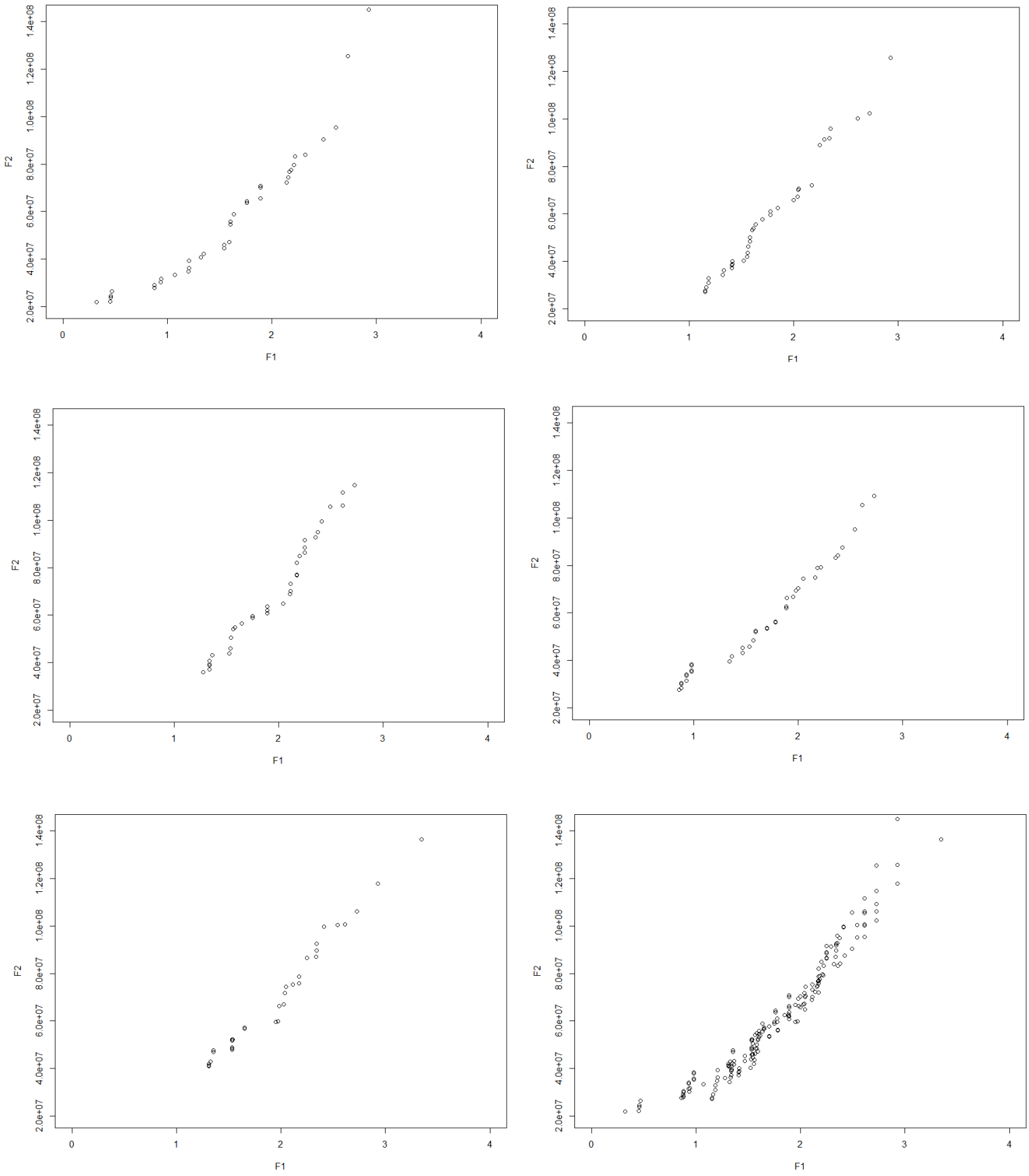

Figure 5.1: 5 Pareto fronts for the Pandemic board game network based on NSGA-II algorithm.

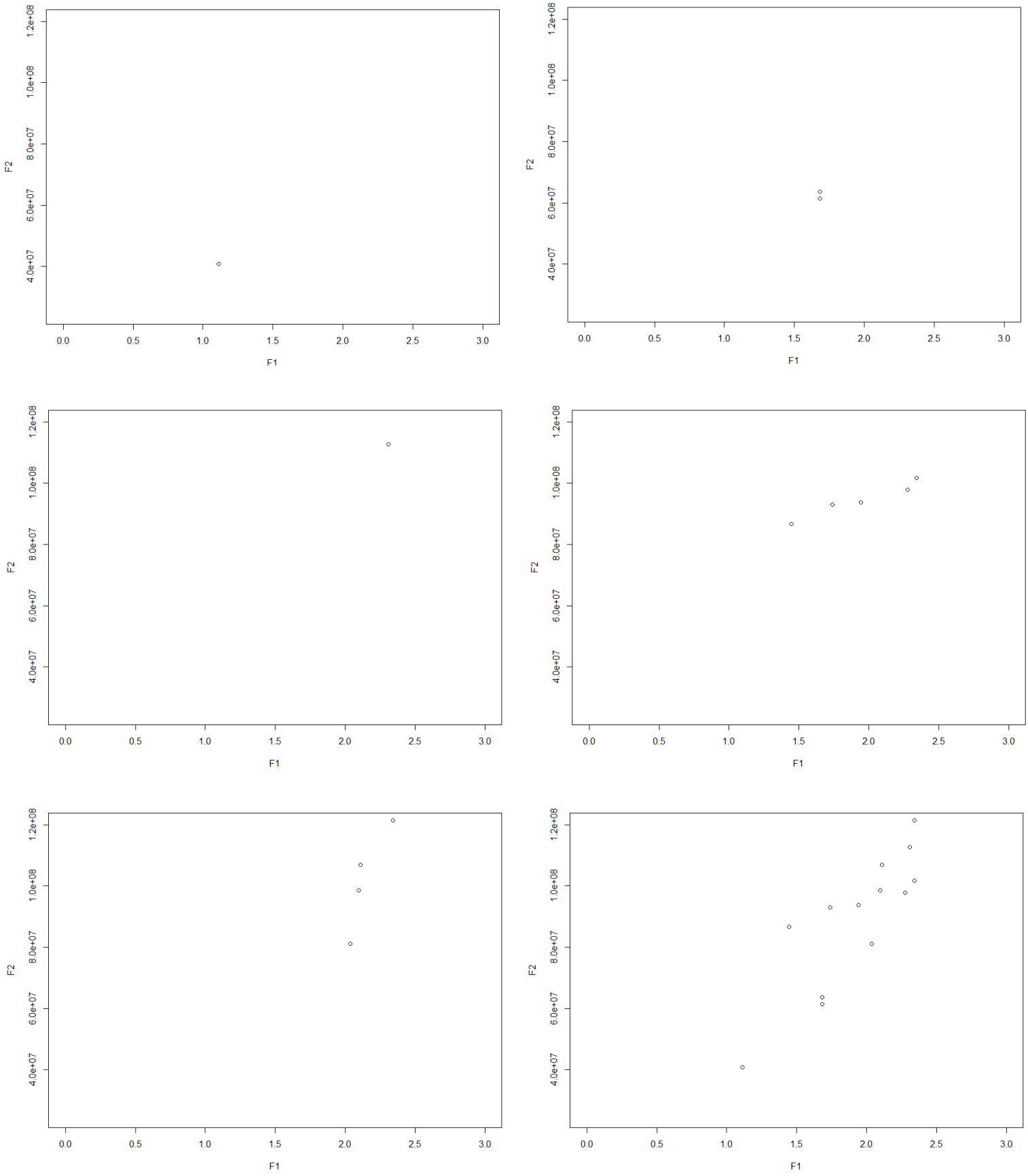

Figure 5.2: 5 Pareto fronts for the Pandemic board game network based on SMS-EMOA algorithm.

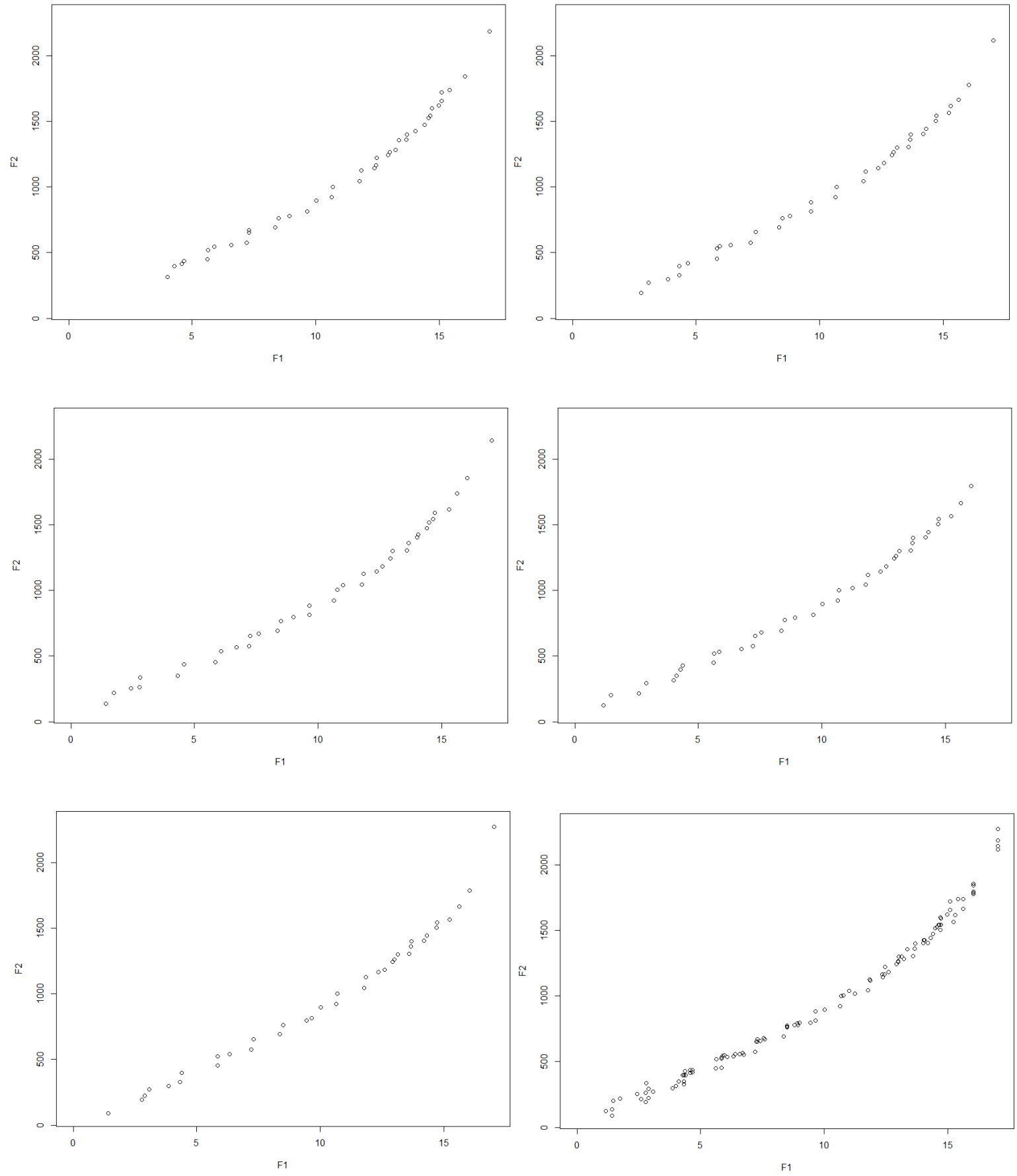

Figure 5.3: 5 Pareto fronts for the USA Flight network based on NSGA-II algorithm.

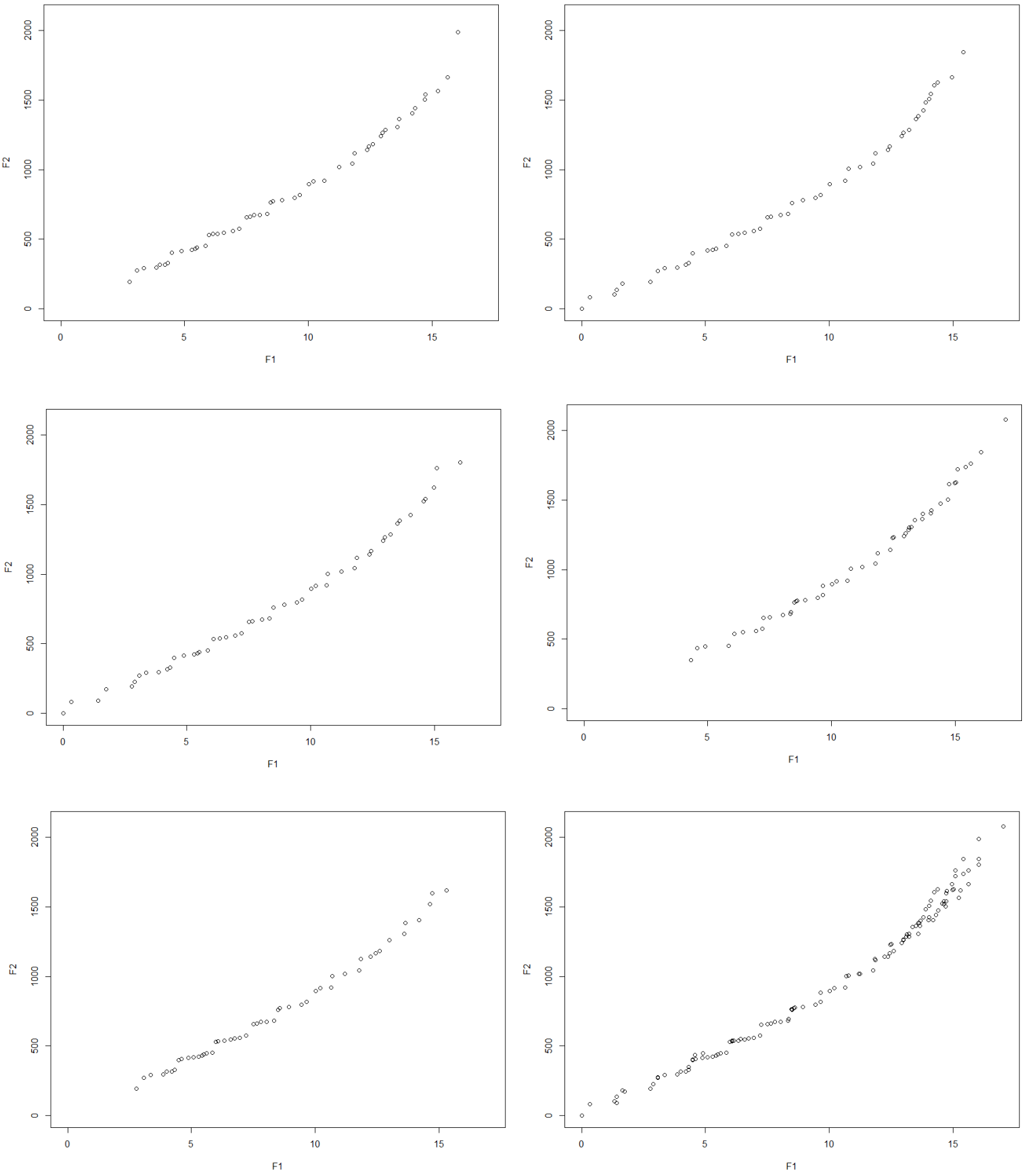

Figure 5.4: 5 Pareto fronts for the USA Flight network based on SMS-EMOA algorithm.

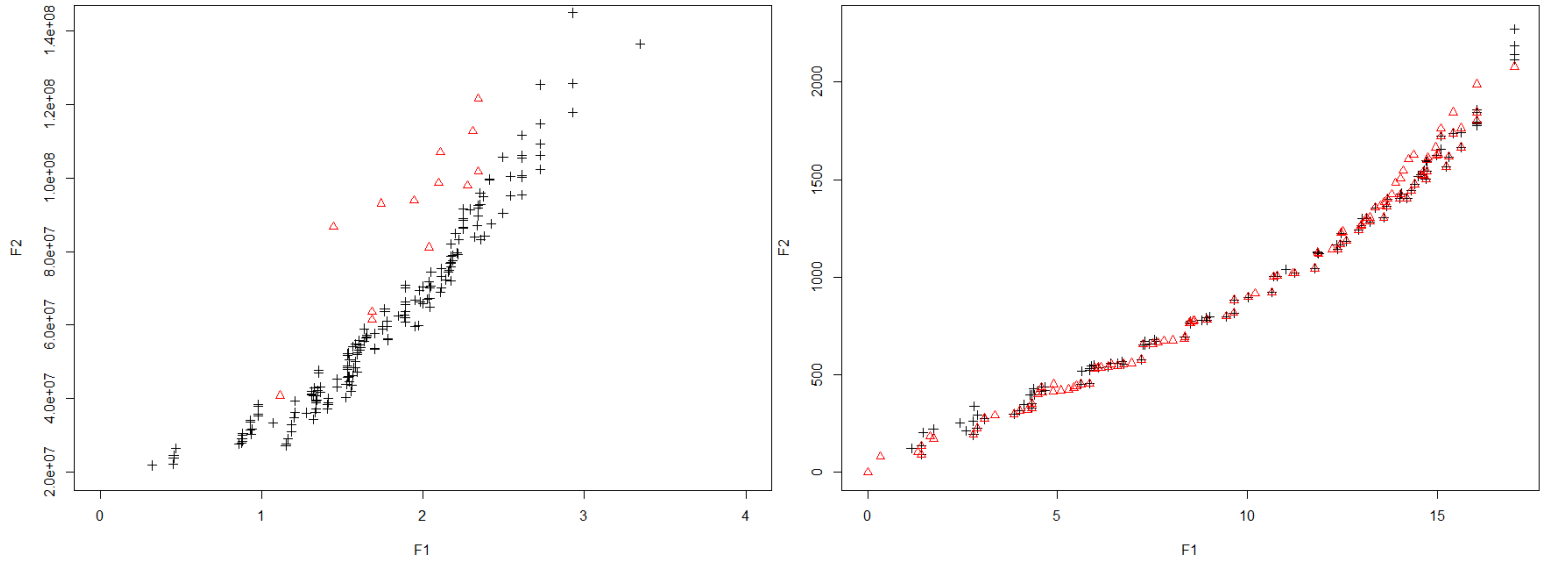

Figure 5.5: Pandemic board game network (left) and USA Flight network (right).

Chapter 6

Discussion and Future Work

In this project we studied the NP hard k-Node Immunization Problem as well as a multiobjective generalization of it, where k is not known a priori. We exemplified that the use of genetic algorithms often returns better, sometimes even significantly more robust results, in solving the k-Node Immunization Problem. We compared our results to Netshield+, a fast, greedy heuristic algorithm which belongs in the class of approximation algorithms, that produced in many cases good results. Based on our findings and results, we recommended the following strategy: if time is available, one should not only use Netshield+, but also a more problem driven genetic algorithm to make it more probable that the best solution for the eigenvalue drop objective is not overlooked. In essence, we believe that a problem specific genetic algorithm for such a complex problem, should pose as a complementary solution, but is some cases it yields a much better solution than Netshield+.

In this study, in order to achieve good results, we designed problem specific adaptations, which turned out to be very useful. An idea that we saw works well, is to use eigen-score values in order to adjust the mutation probabilities. This way the search is more focused on the part of the search space that is more likely to be relevant for solving the problem. We should also emphasize here that the supplementary use of a problem specific genetic algorithm has the advantage of calculating the actual eigen-drop, rather than an approximation of it, like Netshield+. This can be useful for moderate sized networks. However, in large networks the computational cost increases, since the algorithm eigendecomposes larger adjacency matrices.

Finally, first results were also obtained on a multiobjective formulation of the node immunization problem. We discuss the approach where the total cost of immunization is one objective and the drop of eigenvalue is a second objective. Two different state-of-the-art metaheuristics, namely NSGA-II and SMS-EMOA are applied to solve this problem and the results and show

robust performance.

For future work we recommend the study of larger networks, through problem driven adaptations to allow the support of genetic algorithm, which by nature, converge slower to an optimal solution. We consider as a promising route to accomplish this is, to hybridize GA with Netshield+, for instance by using the latter in constructing initial solutions. Moreover, the development of problem specific crossover operators could be also beneficial, since genetic algorithms benefit from the recombination operator. Finally, it would be interesting and useful, to also test disconnected networks to model the spread of air-born diseases. These diseases, which can be transmitted through the air, can be modeled on disconnected networks and thus allow for a study on node selection on these type of networks. In addition, we suggest to model a multiobjective optimization problem by augmenting more data, such as social interactions and expert's knowledge, as well as optimizing more objectives to tackle even more realistic scenarios, for example node immunization importance, node immunization cost and degree immunization for a node.

Part 2

In this part we will deal with the 3rd research question, that is, estimating the critical value for the infection rate λ on finite, square lattices in $\mathbb{Z}^2$ with percolation.
We will start by presenting the Contact Process and the work that has been done in the literature, before moving on to Percolation Theory and to the presentation of our results.

Chapter 7

The Contact Process

The Contact Process (CP) is an example of a model of an *interacting particle system*, which are continuous-time Markov jump processes describing the collective behavior of stochastically interacting components or agents [25]. It is a subfield of Probability Theory that studies models that arise in e.g. statistical physics, biology, economics and epidemiology to name a few [27]. The CP is usually interpreted as a model for the spread of an infection. We will deal with this process on networks (or graphs), particularly, on two-dimensional lattices, which belong in the category of regular networks.

The standard model of the CP on an undirected graph $G = (V, E)$, given an infection rate $\lambda \in (0, +\infty)$, is a continuous-time, discrete-space Markov process $(\eta)_{t\geq 0} \in \{0,1\}^V$, with generator $\mathfrak{L}$ given by

$$(\mathfrak{L}f)(\eta) = \sum_{x\in V} \Big(\eta(x) + (1-\eta(x))\lambda \sum_{y\sim x} \eta(y)\Big)(f(\eta^x) - f(\eta)),$$

where $f : \{0,1\}^V \to \mathbb{R}$ is a bounded function and $x \sim y$ means that there exists an edge between vertices x, y, and $\eta^x(z) = \eta(z)$ if $z \neq x$ and $\eta^x(z) = 1-\eta(z)$ if $z = x$. Here, $\eta_t(x) = 1$ indicates that at time t vertex x is infected and $\eta_t(x) = 0$ indicates that at time t vertex x is healthy.

Informally, the above says that if at time $t \geq 0$, the vertices in $A \subseteq V$ are infected, then as time progresses each uninfected vertex $x \in V \setminus A$ becomes infected at an exponential rate equal to λ times the number of currently infected neighbors, and each infected vertex in A becomes a healthy (uninfected) vertex at an exponential rate equal to 1, independently of the

status of their neighbors. The CP, as we defined it, is also sometimes referred to as the susceptible-infected-susceptible (SIS) epidemic model [26], [27], [28].

The CP was first introduced by Harris in [29], as a continuous-time Markov process $(\eta_t)_{t\geq 0} \in \{0,1\}^{\mathbb{Z}^d}$, where $\mathbb{Z}^d$ is the d-dimensional integer lattice. The CP has been studied on graphs other than $\mathbb{Z}^d$. To our knowledge the first such work was done by Pemantle on infinite trees [30] and on certain non-homogeneous classes of graphs. Chatterjee and Durrett [31] and Berger, Borgs, Chayes and Saberi [32] have considered the CP on two different models of power-law random graphs. Furthermore, the CP has been studied in other contexts, such as high energy physics, where it was introduced by Grassberger and de la Torre [33] and it was shown to be equivalent to the reggeon spin model, a discretization of reggeon field theory [27].

The behavior of the CP depends on the parameter λ, the infection rate of the disease. In the next paragraph we will discuss this dependence.

7.1 Critical Infection Threshold

It is natural to consider what happens to the process as λ increases. It is obvious that the infection will spread faster and also that it will take a longer time for the infection to die out, that is, reach the absorbing state where every individual is healthy. The interesting question, however, is whether there is a critical value, a threshold, for λ at which the CP exhibits a phase transition. In order to present the results we must first define certain notions.

We define $\underline{0}, \underline{1}$ to denote the configuration $\eta \equiv 0, \eta \equiv 1$. We also define $P_t^\eta(\cdot) = P(\eta_t \in \cdot | \eta_0 = \eta)$, the probability distribution of η_t at time t, given the initial configuration η.
We have that, $P_t^{\underline{0}}$ is non-decreasing as function of t and $P_t^{\underline{1}}$ is non-increasing as a function of t. As a result, the limits of these probability distributions exist, which implies the existence of a *critical infection threshold* $\lambda_c \in [0, \infty]$ such that if $\lambda \leq \lambda_c$, then the limit of $P_t^{\underline{1}}$ is $\underline{0}$, meaning that the infection will die out eventually, while if $\lambda \geq \lambda_c$, then the limit is not $\underline{0}$, meaning that the infection survives forever [34], [35]. Setting $p(\lambda)$ to denote the density of the infections for the limit of $P_t^{\underline{1}}$ as $t \to +\infty$, we have that $\lambda_c = \inf\{\lambda \geq 0 | p(\lambda) > 0\} = \sup\{\lambda \geq 0 | p(\lambda) = 0\}$ (see Figure 7.1).

Finally, $p(\lambda)$ as a function of λ is non-decreasing and continuous. Furthermore, we say that the process survives if the infection persists with positive

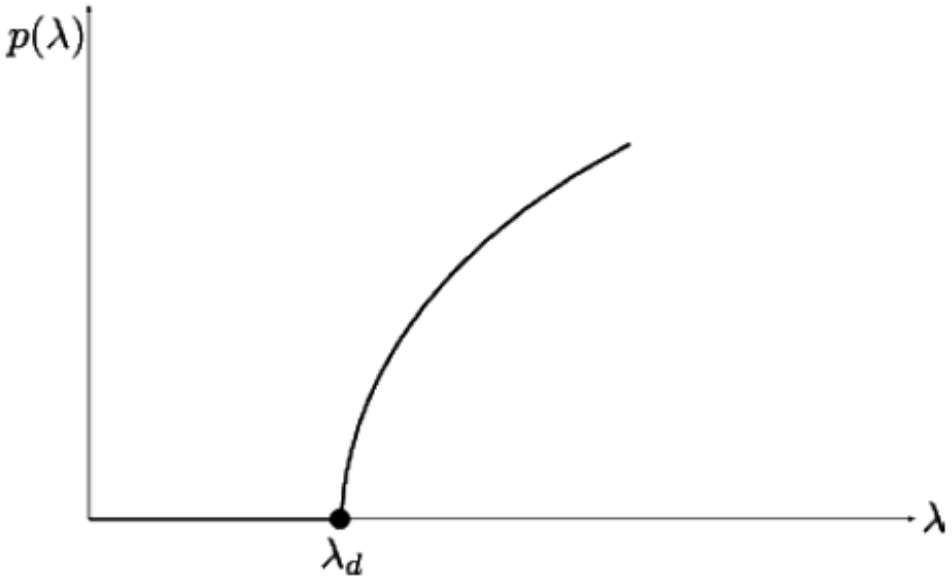

Figure 7.1: Qualitative plot of the density function. [34].

probability; otherwise we say that it dies out. Finally, if N_1, N_2 are two networks for which $N_2 \subseteq N_1$, then $\lambda_c(N_1) \leq \lambda_c(N_2)$ ("a network with more connections has a lower threshold").

We continue by distinguishing between finite and infinite graphs. We start with the latter.

7.1.1 The CP on Infinite Graphs

There have been studies of the CP on infinite lattices, including triangular grids and regular trees. There are two types of survival: weak and strong. The former means that every site gets infected finitely many times (with probability one), while the latter means that it gets infected infinitely many times (with positive probability). We define $\lambda_{c1}, \lambda_{c2}$ as the critical values for the CP surviving weakly and surviving strongly, respectively [44].

On the d-dimensional lattice $\mathbb{Z}^d$ it is known that $\lambda_{c1} = \lambda_{c2} = \lambda_c \equiv \lambda_d$, where d denotes the dimension of the lattice. Furthermore, it can be shown through simulations that $\lambda_{d=1} \approx 1.6494$, and rigorously that $1 < \lambda_1(\mathbb{Z}) < 2$. Also, due to the monotonicity property mentioned previously,

$1 \leq 2d\lambda_d(\mathbb{Z}^d) \leq 2\lambda_1(\mathbb{Z}) < 2$. In addition, we know that for the triangular grid $6\lambda_c(\mathbb{T}) = 1.548$ [34], [35]. Finally, for homogeneous trees $\mathbb{T}_d$ ($d \geq 3$) we have $\lambda_{c1} < \lambda_{c2}$ [44].

7.1.2 The CP on Finite Graphs

On finite graphs the infection becomes extinct with probability 1. The important question is how long it takes for the infection to become extinct. Equivalently: How long will it take for the dynamics on the network to reach the configuration $\underline{0}$ starting from the configuration $\underline{1}$? For this, we define $\tau \equiv \tau_{\underline{0}} = \inf\{t \geq 0 : \xi_t^{\underline{1}} = \underline{0}\}$.

We begin by discussing the results on finite sub-graphs of $\mathbb{Z}^d$. Let $\Lambda_N = [0, N)^d \cap \mathbb{Z}^d, N \in \mathbb{N}$, be the N-block in $\mathbb{Z}^d$. If the infection starts with the configuration $\underline{1}$, then for $\lambda < \lambda_c(\mathbb{Z}^d)$[37]

$$\lim_{N\to\infty} \frac{\tau}{\log(|\Lambda_N|)} = c_1(\lambda) \in (0, \infty).$$

As a result,

$$\lim_{N\to\infty} \frac{E(\tau)}{\log(|\Lambda_N|)} = c_1(\lambda) \in (0, \infty).$$

On the other hand, for $\lambda > \lambda_c(\mathbb{Z}^d)$[38] we have

$$\lim_{N\to\infty} \frac{\log(E(\tau))}{|\Lambda_N|} = c_2(\lambda) \in (0, \infty).$$

Thus, in the sub-critical phase the time to extinction is logarithmic in the volume of the lattice (i.e., very slowly increasing with the volume), while in the super-critical phase it is exponential (i.e., very rapidly increasing with the volume).

In the super-critical phase, the order of magnitude of the extinction time is exponential in the number of vertices of the graph. The process exhibits metastability, meaning that it persists for a long time in a state that resembles an equilibrium, called quasi-equilibrium, and then quickly moves to its true equilibrium, ($\underline{0}$ in this case).

In $d = 1$ and for $\lambda = \lambda_c$, it is known that

$$\lim_{N\to\infty} \frac{\tau}{|\Lambda_N|} = \infty \text{ and } \lim_{N\to\infty} \frac{\tau}{|\Lambda_N|^4} = 0$$

in probability [39].

We continue by discussing results on finite sub-graphs of the d-regular tree $\mathbb{T}^d$. Fix $d \geq 2$ and let $\mathbb{T}_h^d$ be the finite sub-graph of $\mathbb{T}^d$, consisting of all the generations from the root to generation h. Like before, let the CP on $\mathbb{T}_h^d$ start from configuration $\underline{1}$. Then the following results stand:

• If $\lambda < \lambda_{c2}$, then there exist constants $k_1, k_2 > 0$, such that

$$P(k_1 h \leq \tau \leq k_2 h) \to 1$$

as $h \to \infty$.

• If $\lambda > \lambda_{c2}$, then $\forall\ \sigma < 1\ \exists\ k_1, k_2 > 0$ such that

$$P(\tau > k_1 e^{k_2 (\sigma d)^h}) \to 1$$

as $h \to \infty$.

The last result tells us that τ is at least as large as a stretched exponential function of the number of vertices $(d+1)^h$ [40].

We end this paragraph with results on general, finite graphs. Even though there are successful case studies for the extinction time, these depend on the structure of the graphs under consideration and sometimes their relation to some infinite (possibly even random) graph. The following results are context-free i.e., hold for arbitrary (general) sequences of graphs. In addition, there is large literature on the extinction time of the CP on finite graphs. This is sub-divided into two categories: papers that study situations where the extinction time is "large" (i.e., exponential in the number of vertices of the graph) and papers that focus on situations where the infection disappears quickly. However, no rigorous results exist for finite graphs that are not regular.

The following facts have been established [41]: For $n \in \mathbb{N}$ and $d > 0$, let $\Lambda(n, d)$ denote the set of all trees with n vertices and degree bounded by d, and let $G(n, d)$ be the set of graphs having a spanning tree in $\Lambda(n, d)$. Then

• For any $d \in \mathbb{N}$ and $\lambda < \lambda_{c1}(\mathbb{T}^d)$ there exists a $C > 0$ such that, for any graph G with degree bounded by d and $|G| \geq 2$,

$$E(\tau) \leq C \log(|G|).$$

• For any $\lambda > \lambda_c(\mathbb{Z})$ and any $\epsilon > 0$, there exists a constant $c(\epsilon)$ such that, for any *connected* graph G with $|G| \geq 2$,

$$E(\tau) \geq e^{c(\epsilon)\frac{|G|}{\log(|G|)^{1+\epsilon}}},$$

and for any non-empty $A \subseteq G$,

$$P(\eta^A_{\exp\{c(\epsilon)\frac{|G|}{\log(|G|)^{1+\epsilon}}\}} \neq \underline{0}) > c(\epsilon).$$

• For any $\lambda > \lambda_c(\mathbb{Z})$ and any sequence of graphs $(G_n)_{n\in\mathbb{N}}$ with $|G_n| \to \infty$ as $n \to \infty$, $\frac{\tau}{E(\tau)} \to \exp(1)$ in distribution as $n \to \infty$.

The previous results hold for general, finite and connected general graphs. It is interesting to notice that if the infection rate λ is greater than the critical infection rate of the one-dimensional process ($\lambda_c(\mathbb{Z})$) the average extinction time grows faster than $\exp(|G|/(\log|G|)^k)$. Furthermore, this result allows to safely say that: with positive probability we know that starting from any non-empty set of infected vertices, then at time $t = \exp\{c(\epsilon)\frac{|G|}{\log(|G|)^{1+\epsilon}}\}$ the infection has not died out. Finally, the previous results show that the extinction time divided by its expectation converges in distribution to the exp(1) (the unitary exponential distribution) as the number of vertices tends to infinity.

In the next chapter we will briefly discuss Percolation Theory and present the notions that we will read in our chapter on simulations and results.

Chapter 8

Percolation Theory

Percolation Theory is the study of connectivity in large networks. It describes the behaviour of random clusters in these networks. To understand the notion

better, we must think through an example: Imagine that a liquid is poured on top of a porous material, such as a sponge. The question one could ask is: Will the liquid be able to make its way inside the material, from hole to hole, and finally reach the bottom and exit? This practical question is modelled mathematically as a three-dimensional network of $n \times n \times n$ vertices, usually called "sites", in which there may exist edges , usually called "bonds", between two neighboring vertices (allowing the liquid to go through) with probability p, or not with probability 1–p. The probability of an edge existing or not is assumed to be independent of the existence of any other edges. The retained edges are called open and the removed ones closed. Usually, we are interested in the behavior for large n [34]. This problem, called bond or edge percolation, was introduced in mathematics by Broadbent and Hammersley [42], and has been studied intensively by mathematicians and physicists since that time.

There exists also what is called site percolation. By letting a vertex be occupied with probability p or removed with probability $1-p$ (in which case all edges incident to the vertex are removed as well), one again obtains a random graph. In the following we will deal only with bond percolation, and we will call this model *ordinary percolation*.

Since it is easier to examine infinite networks, the question that arises naturally is the following: are there infinite clusters? For this we will define a given vertex as the origin 0 and let $C_p(0)$ denote the cluster containing the origin in the random graph. We define also the percolation function $\theta(p) := \mathbb{P}(|C_p(0)| = \infty)$, denoting the probability that the origin is connected to infinity. By Kolmogorov's zero–one law, for any given p, the probability that an infinite cluster exists is either zero or one. Since this probability is a non-decreasing function of p there must be a critical p below which the probability is always 0 and above which the probability is always 1. We call this critical value p_c and it is defined as $p_c = \sup\{p \in (0,1) : \theta(p) = 0\}$. It is known that $p_c \in (0,1)$ and that $p \to \theta(p)$ is continuous $\forall\ p \neq p_c$ and strictly increasing on $(p_c, 1)$. Continuity is also expected to hold at $p = p_c$. However, this has been proved for the square lattice with $d = 2$ and for $\mathbb{Z}^d$ with $d \geq 11$. Thus, at $p = p_c$ a phase transition occurs:

- $p < p_c$: all clusters are finite
- $p > p_c$: there are infinite clusters

In the supercritical phase it turns out that there is a unique infinite cluster, with probability 1 [43].
For a more thorough reference on percolation theory we point to [44]. In the

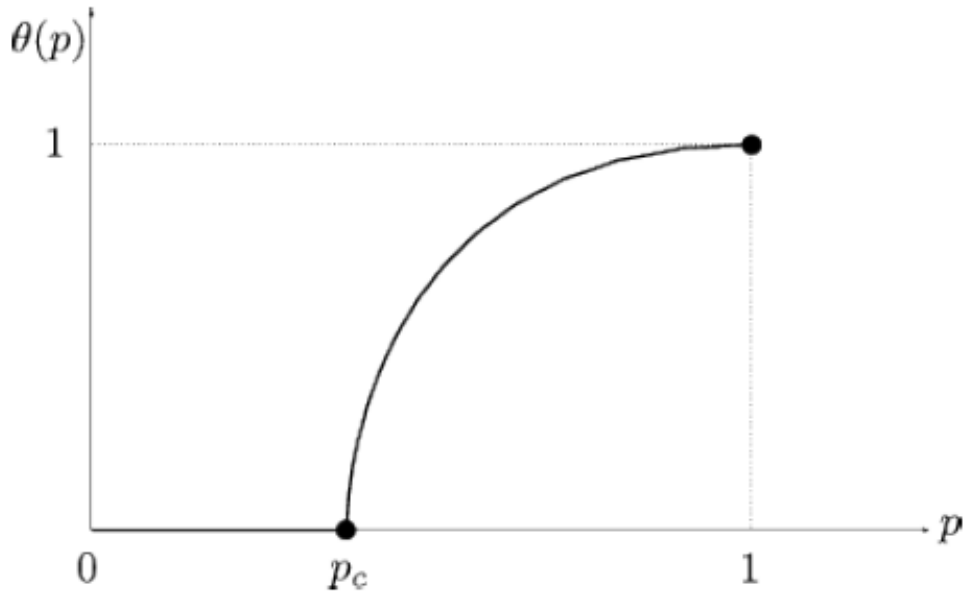

Figure 8.1: Qualitative plot of the percolation function [34].

next chapter, we will discuss our simulations and results.

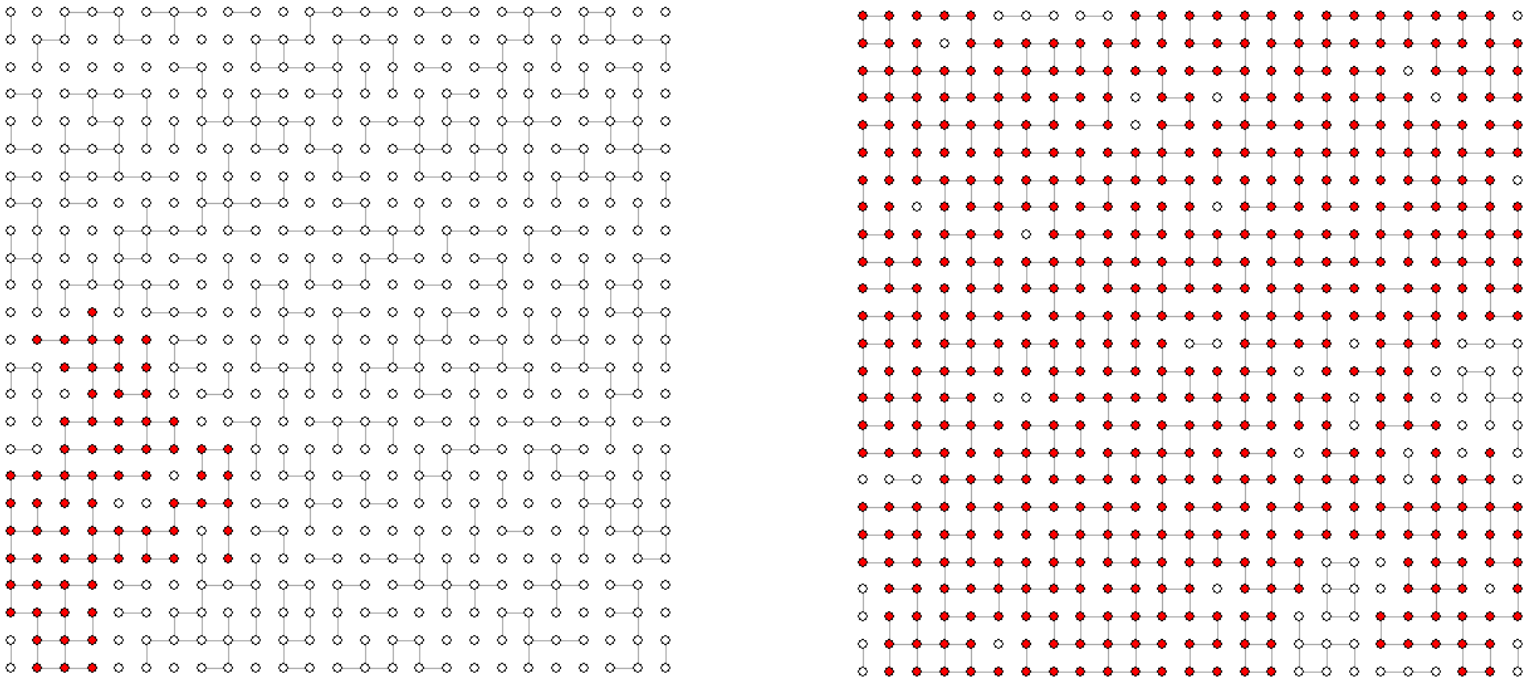

Figure 8.2: Simulation of ordinary percolation on a 25×25 block in $\mathbb{Z}^2$, for $p = 0.4$ (left) and $p = 0.6$ (right). The largest cluster is colored in red. Note that the red cluster spans the area from top to bottom and left to right for $p = 0.6$, but not for $p = 0.4$.

Chapter 9

Simulations and Results

In this chapter we will present the simulations we carried out to address *research question 3*, as well as the results of these simulations. We will start by discussing the problem before moving on to our simulations and the parameters we chose.

9.1 The Problem

The CP on finite graphs eventually dies out and all nodes become healthy. This is because $(\eta_t)_{t\geq 0}$ is a Markov chain on a finite state space. As discussed previously the interesting question is how long this will take?

These conditions in [41] mentioned in paragraph 7.1.2 are not sharp, in the sense that the two critical values do not match, but have a gap. For this reason we used a Continuous Time Markov Chain (CTMC) to simulate the spread and, eventually, the death of the infection on finite square lattices, to seek for sharper bounds. Furthermore, we introduced randomness by putting percolation on the lattices and determining a connection between the percolation probability and the critical infection thresholds on the lattice with edges randomly removed.

9.2 Approach and Results

In this paragraph we will discuss our approach and present our results.

As mentioned previously, we would like to determine sharper bounds for the critical values for the infection rate of which the time to extinction changes drastically. We focused on finite, square lattices with percolation, and we modeled the spread of the disease using a Continuous Time Markov

Chain (CTMC) model. For simplicity, we did not use the actual time to extinction, but instead the number of the Markov Chain steps. We used as an upper bound (i.e., termination criteria) 2000 steps. After that we terminate the simulation. We used the following algorithm:

1. Successively pick $p \in \{0.5, 0.6, 0.7, 0.8, 0.9, 1\}$ to percolate the lattice, using bond percolation with probability p.

2. If the percolated lattice has a giant component (i.e., a cluster spanning from left to right and top to bottom), then continue to step 3, otherwise repeat from step 1.

3. Chose a λ and infect all sites of the lattice. Start the CTMC with this λ and repeat 10 times.

4. If the system becomes healthy (i.e., all nodes have state 0) for this value of λ prior to 2000 steps for *all* 10 runs, then we classify the system as being is in the subcritical, phase and we go to step 3 to choose the next value of λ.

5. We continue like this until we have found at least 3 successive values of λ where the infection has not ceased spreading prior 2000 steps, for all 10 runs.

6. We choose as the critical value for λ the *first* λ for which for all 10 runs the infection did not cease spreading up to 2000 runs (i.e., over this value of λ the system is declared to go in the supercritical phase).

7. We go to step 1 again.

The CTMC rates we used to simulate the CP $(\eta_t)_{t\geq 0}$ are chosen as in [27]:

$$\begin{aligned} \lambda \sum_{y\sim x} \eta(y), \qquad & \text{if } \eta(x) = 0, \\ 1, \qquad & \text{if } \eta(x) = 1, \end{aligned}$$

for $\lambda \in (0, +\infty)$. In words, infected vertices become healthy at rate 1 and healthy vertices become infected at rate λ times the number of infected neighbors. Of course, we cannot simulate an infinite network, and so there will be an upper bound on the size of the clusters. Namely, all clusters have sizes (volume) up to n. This is called a finite size effect. Nevertheless, we used as percolation probability only the values $p \in \{0.5, 0.6, 0.7, 0.8, 0.9, 1\}$, due to the fact that we did not come across a giant component for $p < 0.5$. This did not come as a surprise: although there are finite size effects, for the infinite

lattice there is a critical percolation probability $p_c = 0.5$, for bond percolation: we open (retain) bonds with probability p and we close (discard) with probability $1 - p$. Furthermore, for each p we tested the spreading of the disease for $\lambda \in \{0, 0.2, 0.3, 0.4, 0.5, 0.6, 0.8, 0.9, 1, 1.2, 1.3, 1.4\}$. Even though $\lambda = 0$ is not interesting, we still included it for completeness reasons.

We would like to mention that no analytical results exist (to our knowledge) regarding the behavior of such experiments on finite-sized systems. Thus, our study is to be considered as a first approach.

Next we present our results. We simulated the infection spread on $10 \times 10, 15 \times 15, 20 \times 20$ and 25×25 square lattices (see Figures A.7-A.30 for the percolated lattices of different sizes under different percolation probabilities, used in this study).

Our results in Figure 9.1[4] represent the critical values of λ over which the number of steps to extinction transcends 2000 steps, for each of the different lattice blocks. From the plot what can be immediately seen as a general trend is a drop in the critical thresholds for each lattice as the block gets larger. Specifically, smaller lattices have a higher threshold curve, whereas as the block increases this curve descends. Moreover, it is visible that in the $15 \times 15, 20 \times 20$ and 25×25 lattices the curves are nearly identical at $p = 0.7 - 0.8$. It is also interesting that for each lattice size the critical value line is non-increasing. In words this means that as the lattice tends towards being 4-regular, its "tolerance" or "vulnerability" (with respect to infection rates) decreases. This implies that, as the percolation probability increases, the critical infection rate λ decreases. In addition the critical values for λ for each lattice size under $p = 1$ have a difference between them equal to 0.1 in magnitude. What is more, it is interesting that this critical value lines are all convex. One could interpret this curves as Pareto fronts, where one objective is the percolation probability and the other is the infection parameter λ.

An explanation we believe reflects the observations is that the larger the network (in the number of vertices) and the more connected it is (in the number of edges), the less resilient the network becomes to infections, and more iterations from the stochastic simulation algorithm are needed in order for all vertices to become healthy. This reflects our intuition when we take into account the CP rates we used for the simulation. The more nodes there are, the more infected vertices we begin with in our simulations, and the more connected the network is, the more susceptible nodes become infected. Thus, it is harder for the system to reach its absorbing state (i.e., all nodes are healthy) quicker, even for fairly small values of λ.

[4]In Appendix A, page 97 to page 108, we present the results of our simulations.

Finally, we would like to comment on the following open question. As marked on the plot in Figure 9.1 we see that as $p \to p_c = 0.5$ then λ_c converges to a point *strictly* below the critical value for the one-dimensional process $\lambda_c(\mathbb{Z}) \approx 1.6494$ (dashed black line), and this can be seen for all the different blocks. It is believed that this result holds for the infinite lattice in which percolation has occurred, as $p \downarrow p_c = 0.5$. Indeed, for any $p > p_c$ there exists an infinite cluster with probability 1, and as a result a one-dimensional lattice will be present in it. Thus, the (infinite) network will be more vulnerable in sustaining an infection than $\mathbb{Z}$, which is implied by the more general fact that if N_1, N_2 are two networks for which $N_2 \subseteq N_1$, then $\lambda_c(N_1) \leq \lambda_c(N_2)$.

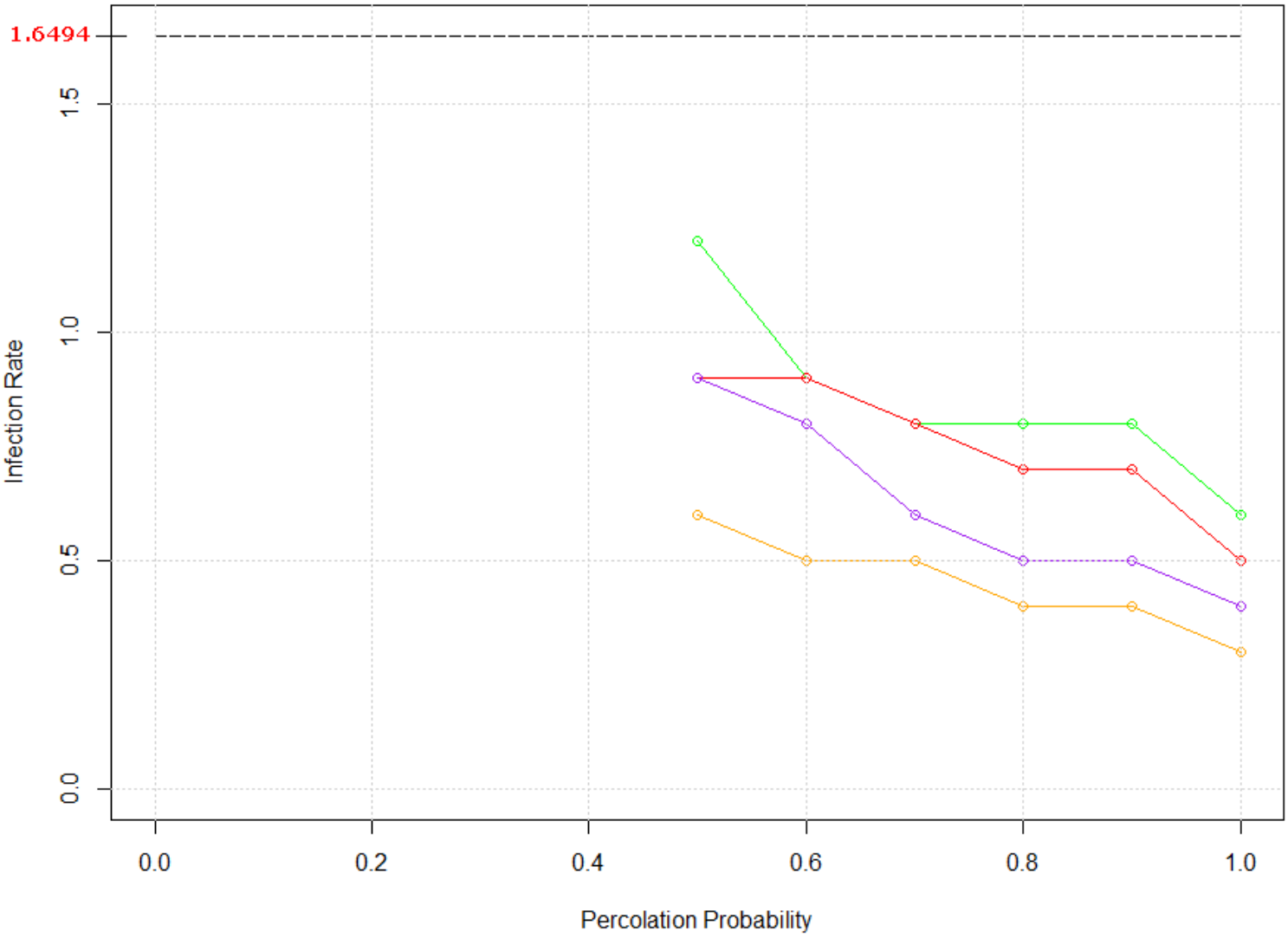
1.6494
1.5
1.0
0.5
0.0
Infection Rate
0.0
0.2
0.4
0.6
0.8
1.0
Percolation Probability

Figure 9.1:

Chapter 10

Discussion and Future Work

Following upon the work presented in [41], we simulated, using a continuous time Markov chain model, called the the contact process on random graphs generated from finite, square lattices with percolation. Our goal was to obtain bands on the, critical thresholds for the infection parameter λ. We used as a measure of criticality the number of steps the stochastic simulation must do in order to end up in the absorbing state where all individuals are healthy, starting from a fully infected network. We demonstrated the infection spread on $10 \times 10, 15 \times 15, 20 \times 20$ and 25×25 square lattices. Our results indicate that the larger the network is (in the number of vertices) and the more connected it is (in the number of edges), the harder it is for system to reach the absorbing state, for at least fairly small values of λ, in the sense that the simulation needs more steps to reach that state. We used as a criticality cutoff point 2000 steps. If the system did not reach the absorbing state prior to 2000 steps, then we terminated the simulation and classified it as being in the supercritical phase.

While this study is merely a first approach to the problem, we believe it can be used as a methodology for future studies in this direction, where empirical results are needed. We propose for future work the study of other networks, such as the Erdős-Rényi model, the configuration model and the preferential attachment model of Barabási and Albert. What is more, we suggest a study towards an immunization strategy in the following manner: Determine the critical values of the infection parameter λ by removing a number k of key-nodes (such as largest degree vertices), taking into account the extinction time of the infection. In this view, we propose the use of the largest eigenvalue drop (discussed in Part 1) to determine which nodes are the most significant and estimate the critical values of λ for a given number k, indicating the number of significant nodes to be immunized.

Bibliography

[1] Anderson RM, May RM. Immunization and herd-immunity. Lancet 335 (8690), 641–645, 1990.

[2] Thedchanamoorthy, G., Piraveenan, M., Uddin, S. et al. Influence of vaccination strategies and topology on the herd immunity of complex networks. Social Network Analysis and Mining, 4 (1), 1-16, 2014.

[3] R. Cohen, S. Havlin and D. ben Avraham. Efficient immunization strategies for computer networks and populations. Phys. Rev. Lett.,92 (24), 247901, 2003.

[4] Romualdo Pastor-Satorras and Alessandro Vespignani Phys. Rev. E 65, 036104 – Published 8 February 2002.

[5] R.M. Anderson and R.M. May. Infectious Diseases in Humans. Oxford University Press, Oxford, 1992.

[6] D. Chakrabarti, Y. Wang, C. Wang, J. Leskovec, and C. Faloutsos. Epidemic thresholds in real networks. ACM Transactions on Information and System Security, 10(4), 1, 2007.

[7] B. A. Prakash, D. Chakrabarti, N. C. Valler, M. Faloutsos, and C. Faloutsos. Threshold conditions for arbitrary cascade models on arbitrary networks, Knowledge and Information Systems, 33 (3), 549–575, 2012.

[8] Chen Chen, Hanghang Tong, B. Aditya Prakash, Charalampos E. Tsourakakis, Tina Eliassi-Rad, Christos Faloutsos, and Duen Horng Chau. Node Immunization on Large Graphs: Theory and Algorithms. IEEE Transactions on Knowledge and Data Engineering, 28 (1), January 2016.

[9] Wayne W Zachary. An information flow model for conflict and fission in small groups. Journal of anthropological research, 33 (4):452–473, 1977.

[10] David Lusseau, Karsten Schneider, Oliver J Boisseau, Patti Haase, Elisabeth Slooten, and Steve M Dawson. The bottlenose dolphin community of doubtful sound features a large proportion of longlasting associations. Behavioral Ecology and Sociobiology, 54 (4):396– 405, 2003.

[11] M Leacock. Pandemic.[board game]. Z-Man Games: Mahopac, NY, 2008.

[12] `http://www.sociopatterns.org/datasets/infectious-sociopatterns`. Retrieved May 2017.

[13] May, Robert M.; Anderson, Roy M. (1991). Infectious diseases of humans: dynamics and control. Oxford [Oxfordshire], Oxford University Press. ISBN 0-19-854040-X.

[14] Michael Emmerich, Diego Garlaschelli, Frank den Hollander, Complex Networks, Leiden University (as part of the lecture notes for the course Complex Networks), December 17 2016.

[15] `https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)`. Retrieved August 2017.

[16] `http://www.cs.elte.hu/~lovasz/eigenvals-x.pdf`. Retrieved August 2017.

[17] `https://en.wikipedia.org/wiki/Poincar%C3%A9_separation_theorem`. Retrieved on 10 of August 2017.

[18] M. Seetharama Gowda and J. Tao, The Cauchy interlacing theorem in simple Euclidean Jordan algebras and some consequences, Linear and Multilinear Algebra Vol. 00, No. 00, Month 200x, 1–23, received 12 August 2008.

[19] J. H. Holland. Adaptation in natural and artificial systems. The University of Michigan Press, Ann Arbor, MI, 1975.

[20] Bäck, Thomas Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms, Oxford university press, 1996.

[21] Yang, Jihoon and Honavar, Vasant. Feature subset selection using a genetic algorithm. IEEE Intelligent Systems and their Applications, 13 (2), 44-49, 1998.

[22] Deb, Kalyanmoy and Pratap, Amrit and Agarwal, Sameer and Meyarivan. TAMT, A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on evolutionary computation, 6 (2), 182-197, 2002.

[23] Beume, Nicola and Naujoks, Boris and Emmerich, Michael,SMS-EMOA: Multiobjective selection based on dominated hypervolume, European Journal of Operational Research, 181 (3), 1653-1669, 2007, Elsevier.

[24] Kalyanmoy Deb, Shivam Gupta,Understanding Knee Points in Bicriteria Problems and TheirImplications as Preferred Solution Principles Kanpur Genetic Algorithms Laboratory (KanGAL) Indian Institute of Technology Kanpur, KanGAL Report Number 2010005, July 5, 2010.

[25] `https://en.wikipedia.org/wiki/Interacting_particle_system`. Retrieved August 2017.

[26] Jonathon Peterson. The contact process on the complete graph with random vertex-dependent infection rates. Stochastic Processes and their Applications, 121 (3), 609-629, 2011.

[27] T. M. Liggett. Interacting Particle Systems. Springer, Berlin, 1985.

[28] T. M. Liggett. Stochastic Interacting Systems: Contact, Voter and Exclusion Processes. Springer, Berlin, 1999.

[29] T.E.Harris. Contact Interaction on a Lattice. The Annals of Probability, 2 (6), 969-988, 1974.

[30] Robin Pemantle. The contact process on trees. The Annals of Probability, 20(4):2089–2116, 1992.

[31] Shirshendu Chatterjee and Rick Durrett. Contact processes on random graphs with power law degree distributions have critical value 0. The Annals of Probability, 37 (6), 2332–2356, 2009.

[32] Noam Berger, Christian Borgs, Jennifer T. Chayes, and Amin Saberi, On the spread of viruses on the internet. In Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, New York, 2005, 301–310 (electronic).

[33] Grassberger, P.; de la Torre, A. Reggeon field theory (Schogl's first model) on a lattice: Monte Carlo calculations of critical behaviour. Annals of Physics, 122 (2), 373-396.

[34] Frank den Hollander, Diego Garlaschelli , Michael Emmerich. Complex Networks. Faculty of Science, Leiden University, The Netherlands, lecture notes, 2017.

[35] Frank den Hollander. Epidemics on Networks. Studium Generale: Complex Networks, Challenges and Perspectives. Leiden, 22/02 - 27/03, 2017.

[36] C. Bezuidenhout and G. R. Grimmett. The critical contact process dies out. The Annals of Probability, 18 (4), 1462-1482, 1990.

[37] R. Durrett and X. F. Liu. The contact process on a finite set. The Annals of Probability, 16 (3), 1158-1173, 1988.

[38] R. Durrett and R. H. Schonmann. The contact process on a finite set II. The Annals of Probability, 16 (4), 1570-1583, 1988.

[39] R. Durrett, R. H. Schonmann and N. Tanaka. The contact process on a finite set III. The critical case. The Annals of Probability, 17 (4), 1303-1321, 1989.

[40] A. M. Stacey. The contact process on finite homogeneous trees. Probability Theory and Related Fields, 121 (4), 551–576, 2001.

[41] Schapira, B. and Valesin, D. Extinction time for the contact process on general graphs. Probability Theory and Related Fields, 1-29, 2015.

[42] Broadbent, Simon and Hammersley, John, Percolation processes I. Crystals and mazes. Proceedings of the Cambridge Philosophical Society, 1957, 53, 629–641.

[43] Burton, R.M., Keane, M. Density and uniqueness in percolation. Communications in Mathematical Physics, 121, 501–505, 1989.

[44] G.R. Grimmett. Percolation. Springer, Berlin, 1989.

Appendix A

Networks

In table A.1 below we summarize the characteristics of our Networks.

Table A.1:

Networks	nodes	edges
karate	34	78
Dolphins	62	159
USA	27	207
Pandemic	48	93
Conf. day 1	190	703
Conf. day 3	148	517

Remark: We should mention here the following: For the networks Conf. day 1 and Conf. day 3, we have used only their largest connected component in our study. The reason for this has been that Netshield/Netshield+ only deal with connected graphs and thus, it would be only fair to do our comparisons on such graphs.

We continue by visually presenting the graphs we used.

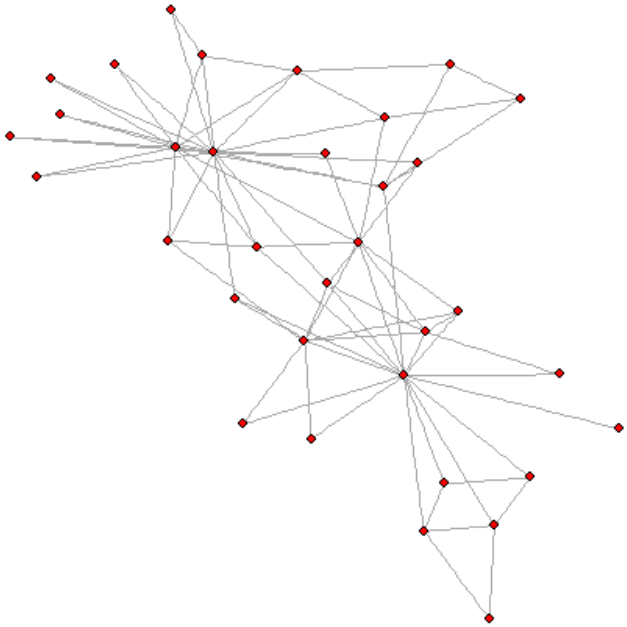

Figure A.1: Network of Karate Club.

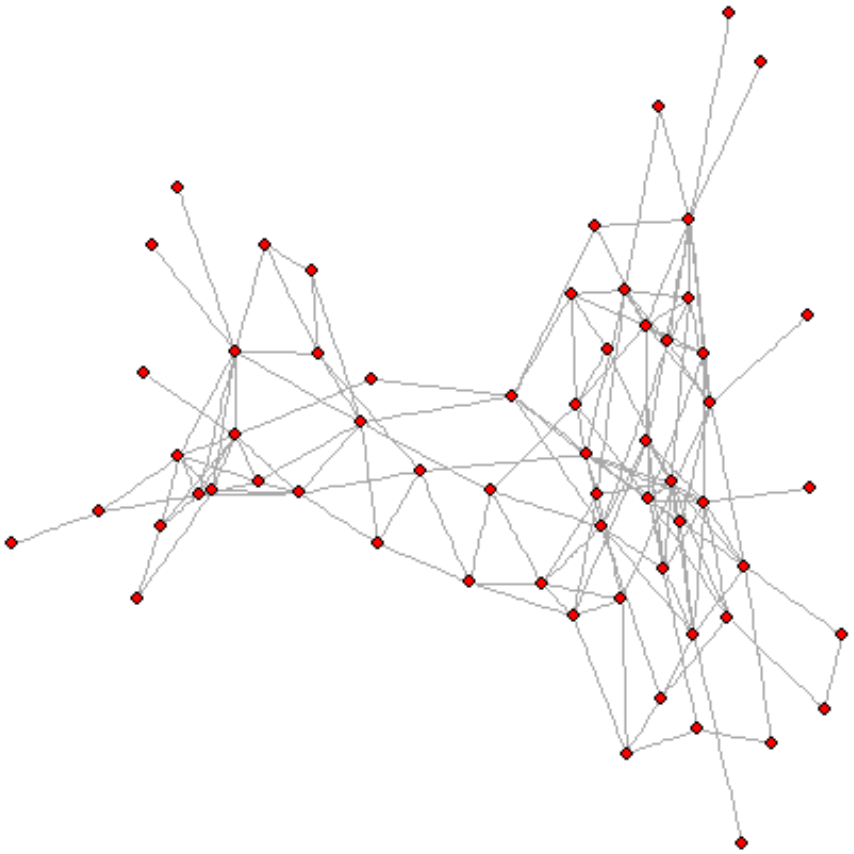

Figure A.2: Network of Dolphins.

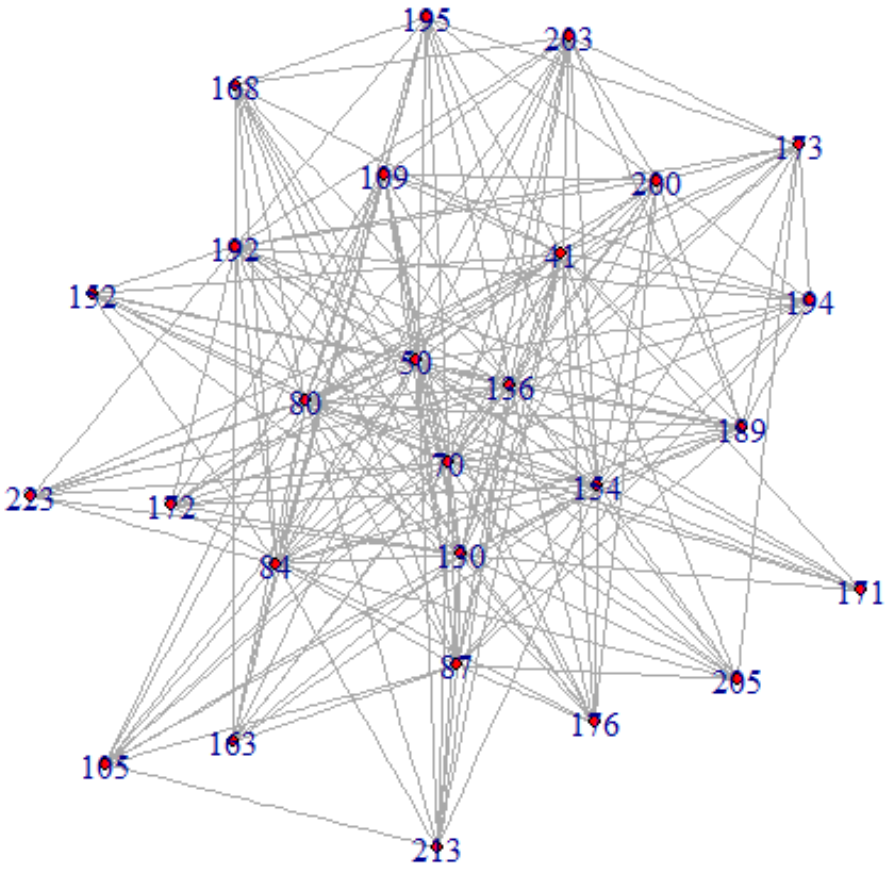

Figure A.3: Network of USA flights.
See also next page for the label meaning.

Label	*Airport*	*Visits*
41	$Cincinnati/northernKentucky$	117
50	$DetroitMetropolitanWayneCounty$	126
70	$GeorgeBushIntercontinental$	90
80	$Hartsfield-jacksonAtlantaInternational$	102
84	$HopkinsInternational$	123
87	$IndianapolisInternational$	120
105	$KansasCityInternationalAirport$	117
109	$LaGuardia$	123
130	$MemphisInternational$	105
136	$Minneapolis-St.PaulIntl$	135
152	$NashvilleInternational$	108
154	$NewarkLibertyInternational$	123
163	$OrlandoInternational$	84
168	$PhiladelphiaInternational$	120
171	$PittsburghInternational$	120
172	$PortColumbusIntl$	120
173	$PortlandInternational$	138
176	$Raleigh-durhamInternationalAirport$	108
191	$RonaldReaganWashingtonNationalAirport$	117
192	$SaltLakeCityInternational$	123
194	$SanDiegoInternationalAirport$	99
195	$SanFranciscoInternational$	114
200	$Seattle-TacomaInternational$	141
203	$SkyHarborIntl$	99
205	$SouthwestFloridaReg$	81
213	$TampaInternational$	84
223	$WashingtonDullesInternational$	117

Table A.2: Cost values for Pandemic network (proportional to city size).

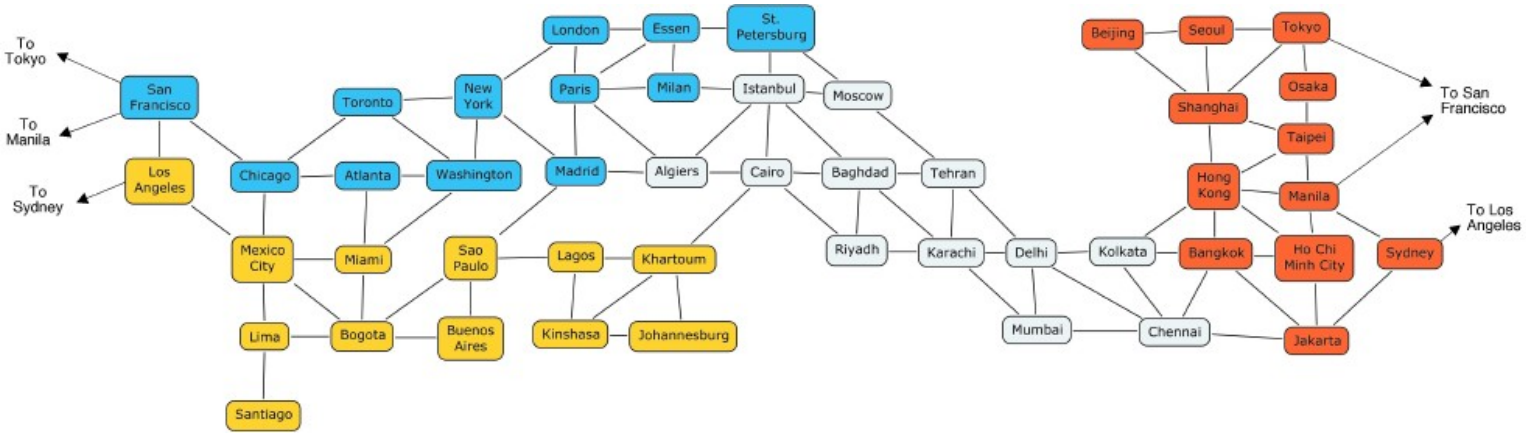

Figure A.4: Network of the Pandemic board game.

ID	*City*	*Population*	*ID*	*City*	*Population*
1	*SanFrancisco*	723724	25	*Beijing*	7602069
2	*Chicago*	2830144	26	*Seoul*	9860000
3	*Montreal*	3280123	27	*Tokyo*	8372440
4	*NewYork*	8124427	28	*Shanghai*	15017783
5	*Washington*	548359	29	*HongKong*	7347000
6	*Atlanta*	424096	30	*Taipei*	2491662
7	*Madrid*	3146804	31	*Osaka*	2590815
8	*London*	7489022	32	*Bangkok*	4935988
9	*Paris*	2141839	33	*HoChiMinhCity*	3496586
10	*Essen*	596204	34	*Manila*	10546511
11	*Milan*	1316218	35	*Jakarta*	8556798
12	*St.Petersburg*	4991000	36	*Sydney*	4444513
13	*Algiers*	2029936	37	*Khartoum*	2090001
14	*Istanbul*	10034830	38	*Johannesburg*	2091491
15	*Moscow*	10472629	39	*Kinshasa*	9464000
16	*Cairo*	7836243	40	*Lagos*	9020089
17	*Baghdad*	5753612	41	*SaoPaulo*	10059502
18	*Tehran*	7160094	42	*BuenosAires*	11595183
19	*Delhi*	11215130	43	*Santiago*	4893495
20	*Karachi*	11969284	44	*Lima*	7857121
21	*Riyadh*	4328067	45	*Bogota*	7235084
22	*Mumbai*	18410000	46	*MexicoCity*	8659409
23	*Chennai*	7088000	47	*LosAngeles*	3911500
24	*Kolkata*	4497000	48	*Miami*	386740

Table A.3: Cost values for Pandemic network (proportional to city size). Data taken from `kateto.net/network-visualization`

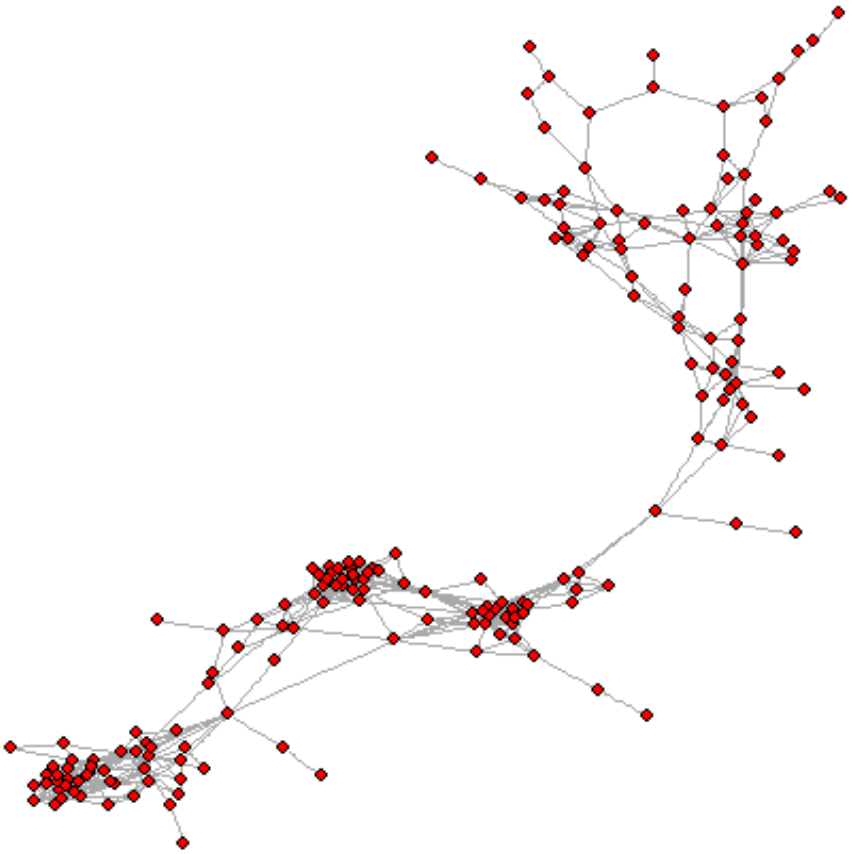

Figure A.5: Network of Conference Day 1.

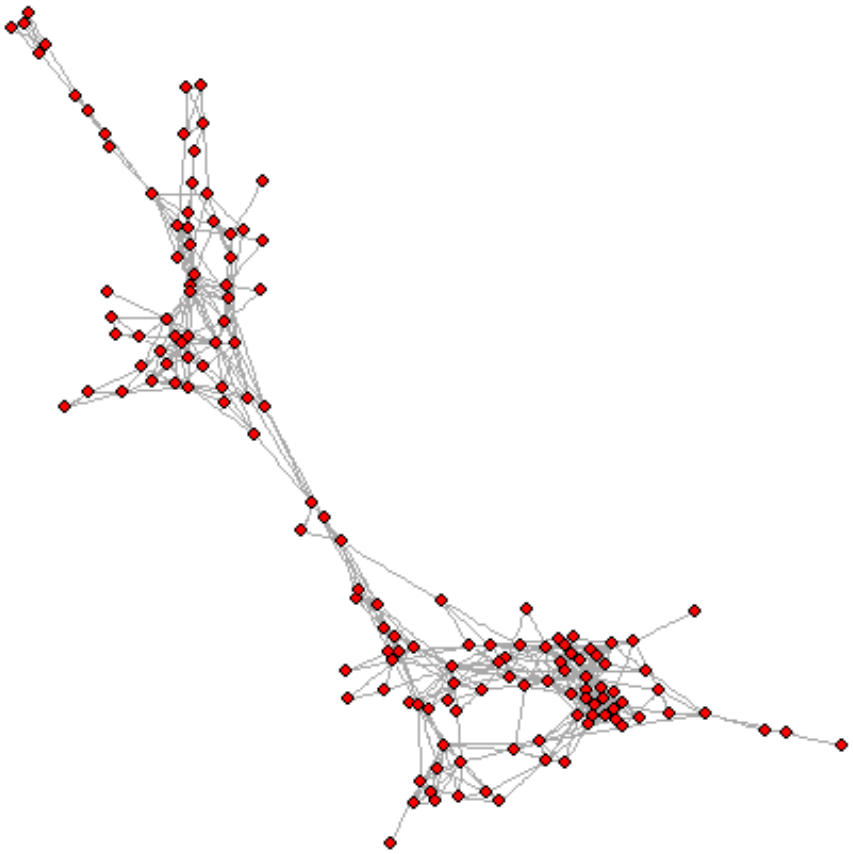

Figure A.6: Network of Conference Day 3.

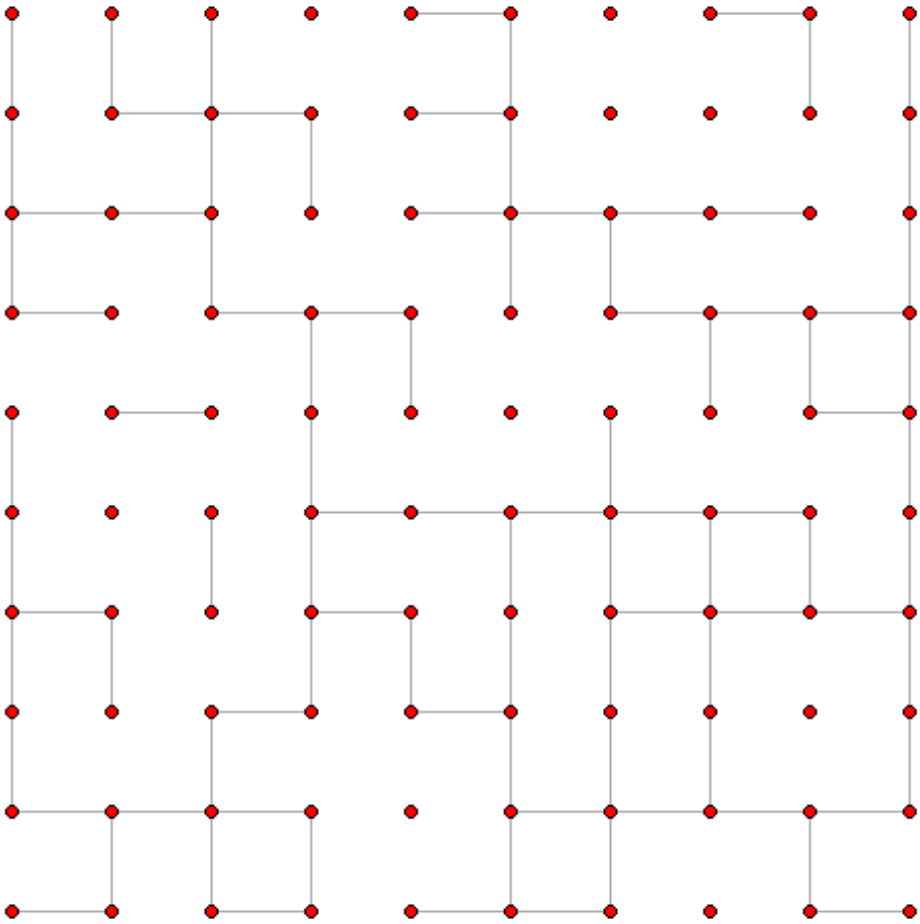

Figure A.7: 10×10 lattice with percolation probability $p = 0.5$

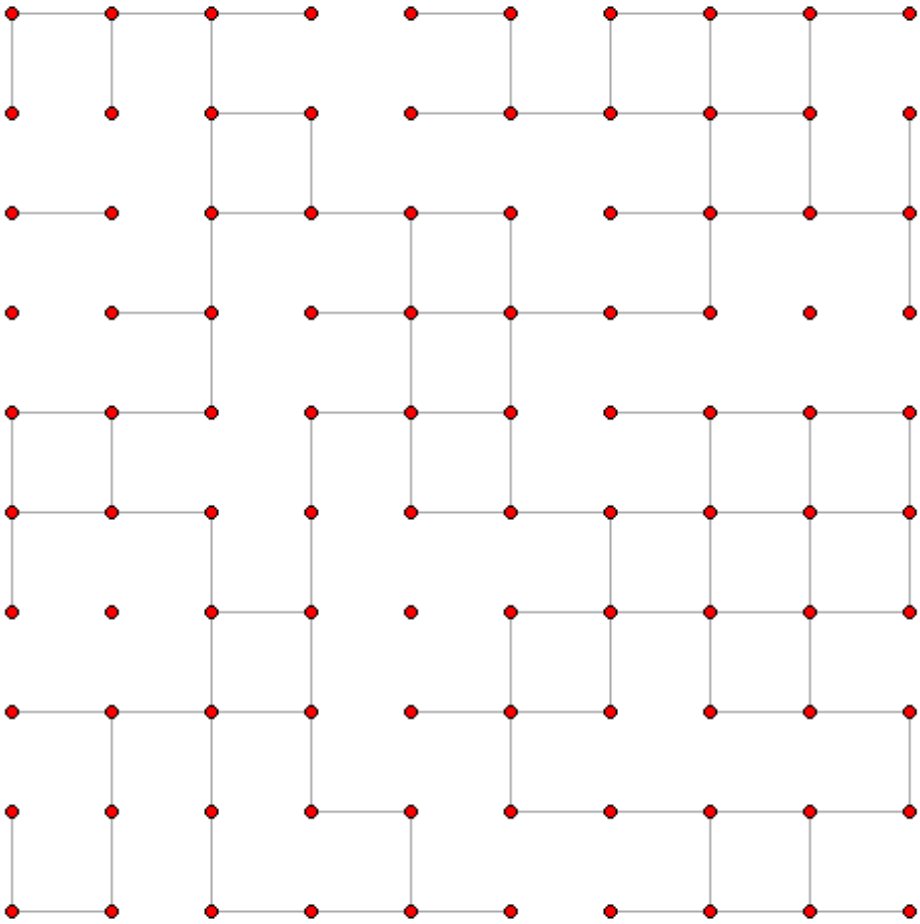

Figure A.8: 10×10 lattice with percolation probability $p = 0.6$

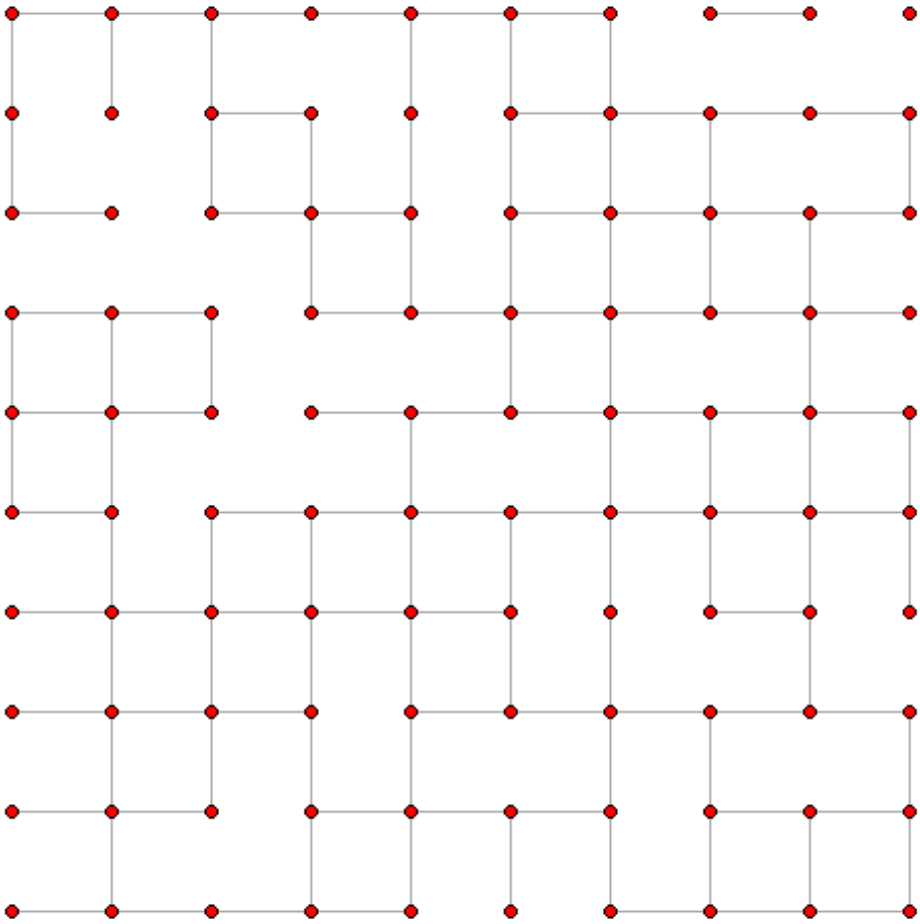

Figure A.9: 10×10 lattice with percolation probability $p = 0.7$

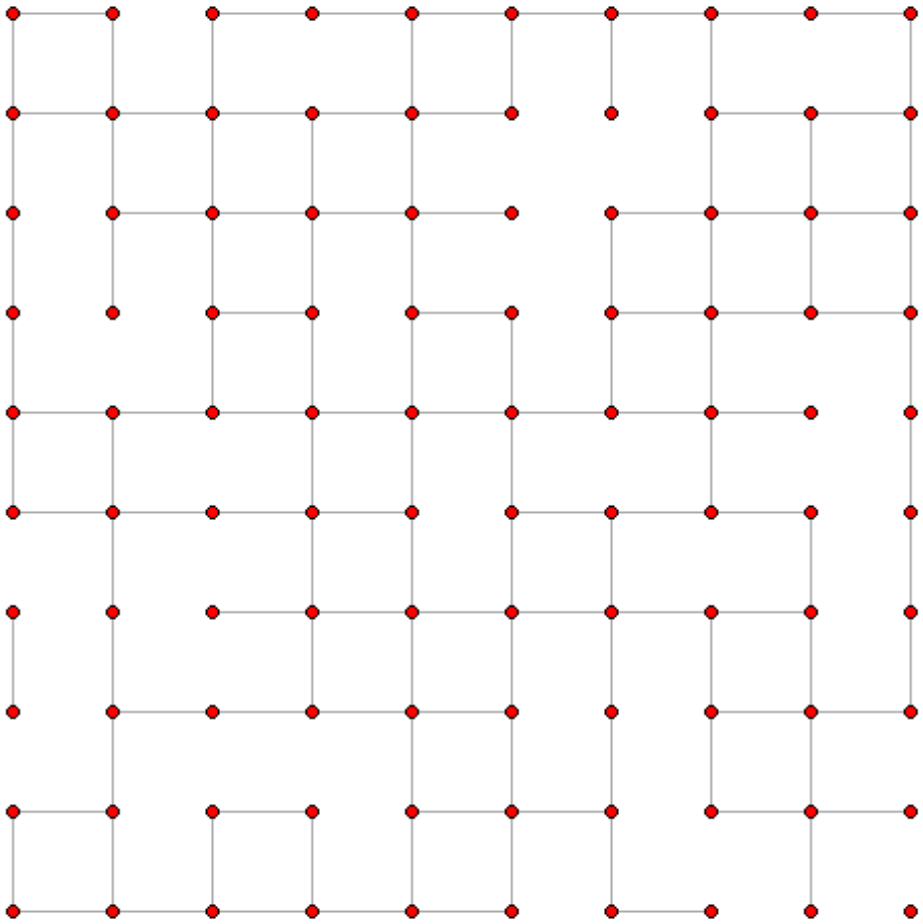

Figure A.10: 10×10 lattice with percolation probability $p = 0.8$

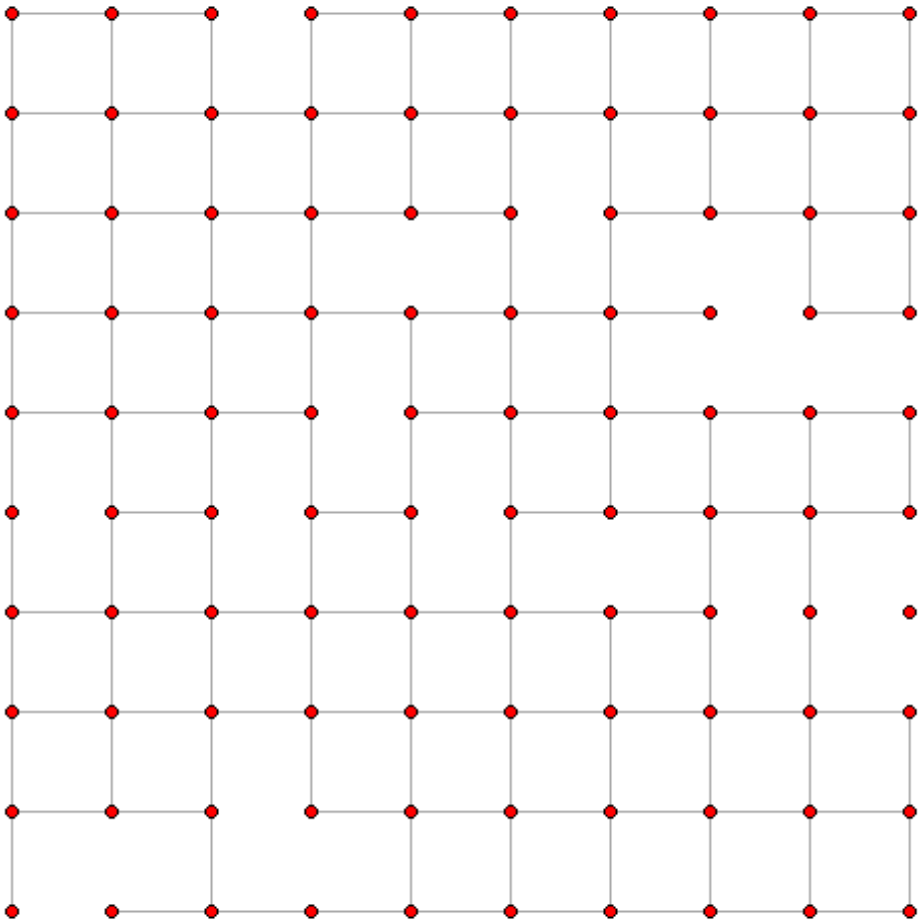

Figure A.11: 10×10 lattice with percolation probability $p = 0.9$

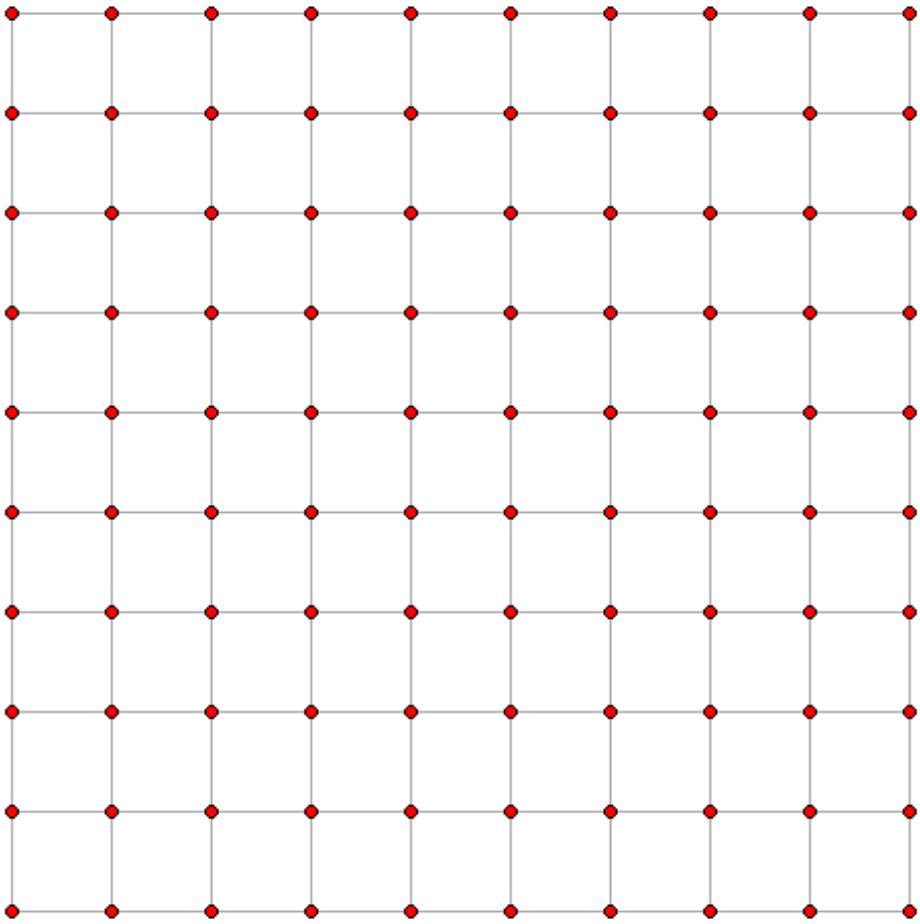

Figure A.12: 10×10 lattice with percolation probability $p = 1$

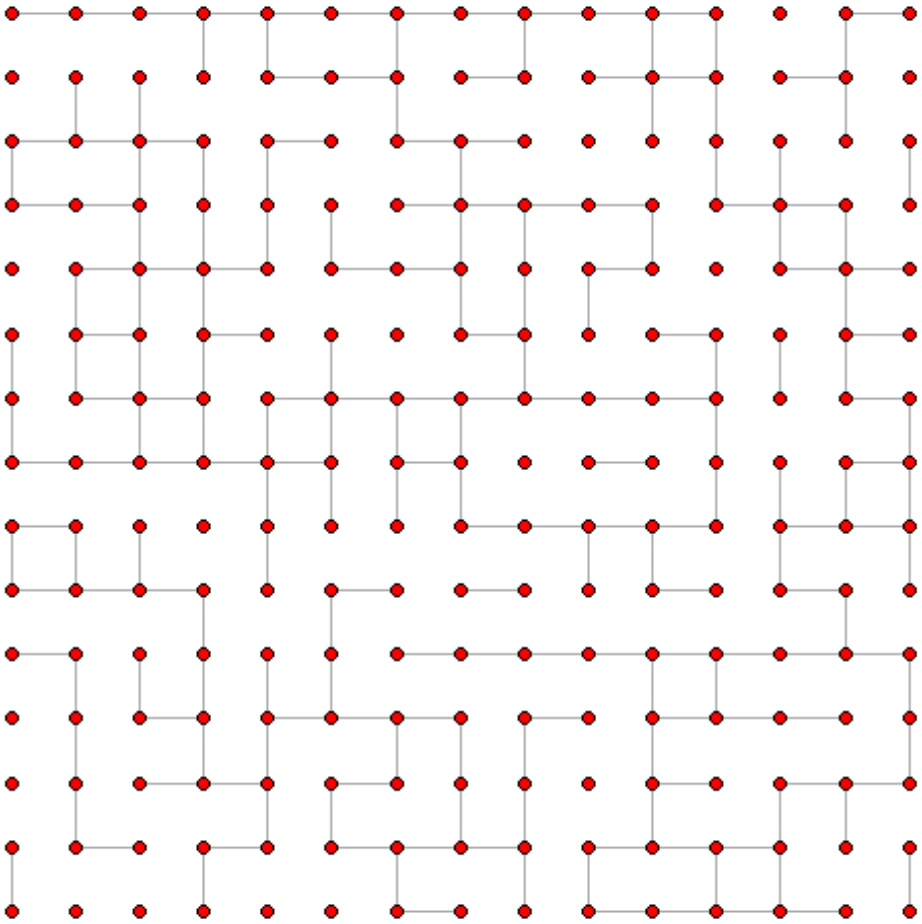

Figure A.13: 15×15 lattice with percolation probability $p = 0.5$

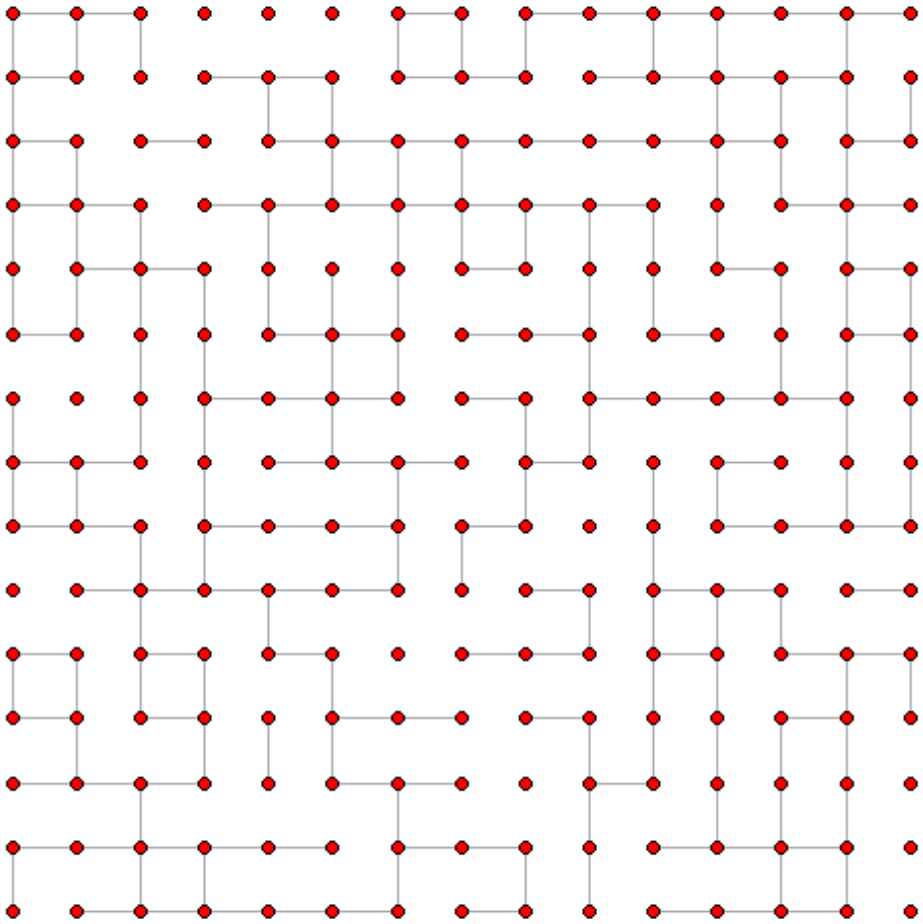

Figure A.14: 15×15 lattice with percolation probability $p = 0.6$

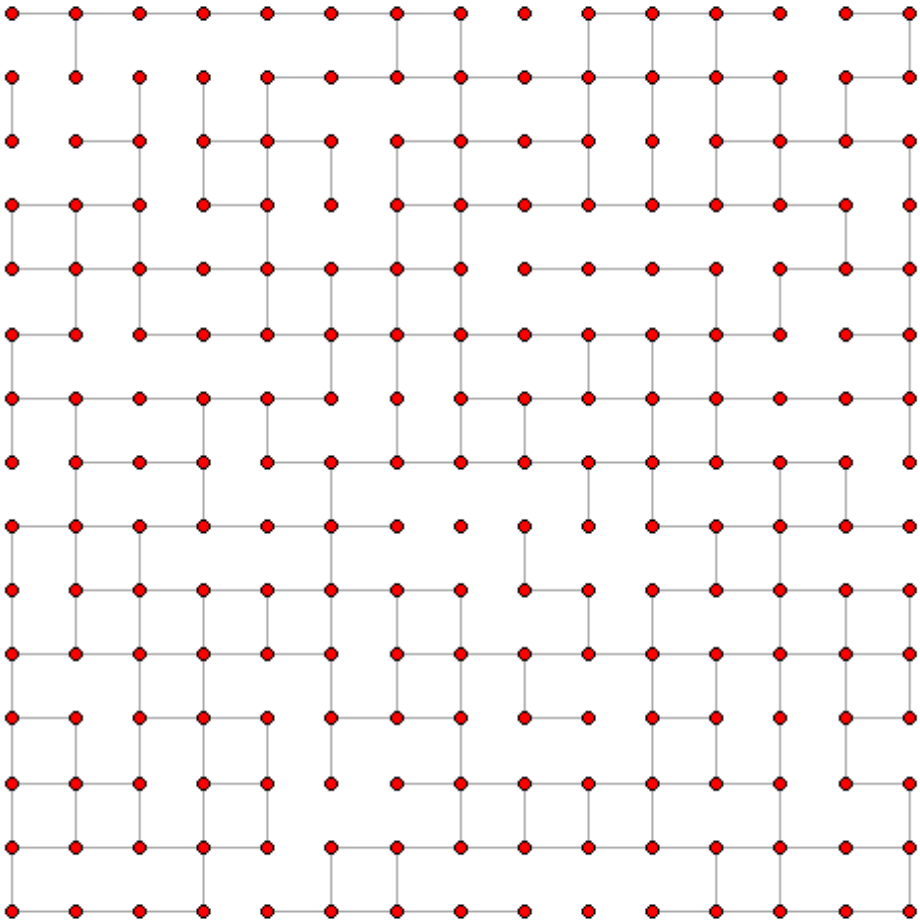

Figure A.15: 15×15 lattice with percolation probability $p = 0.7$

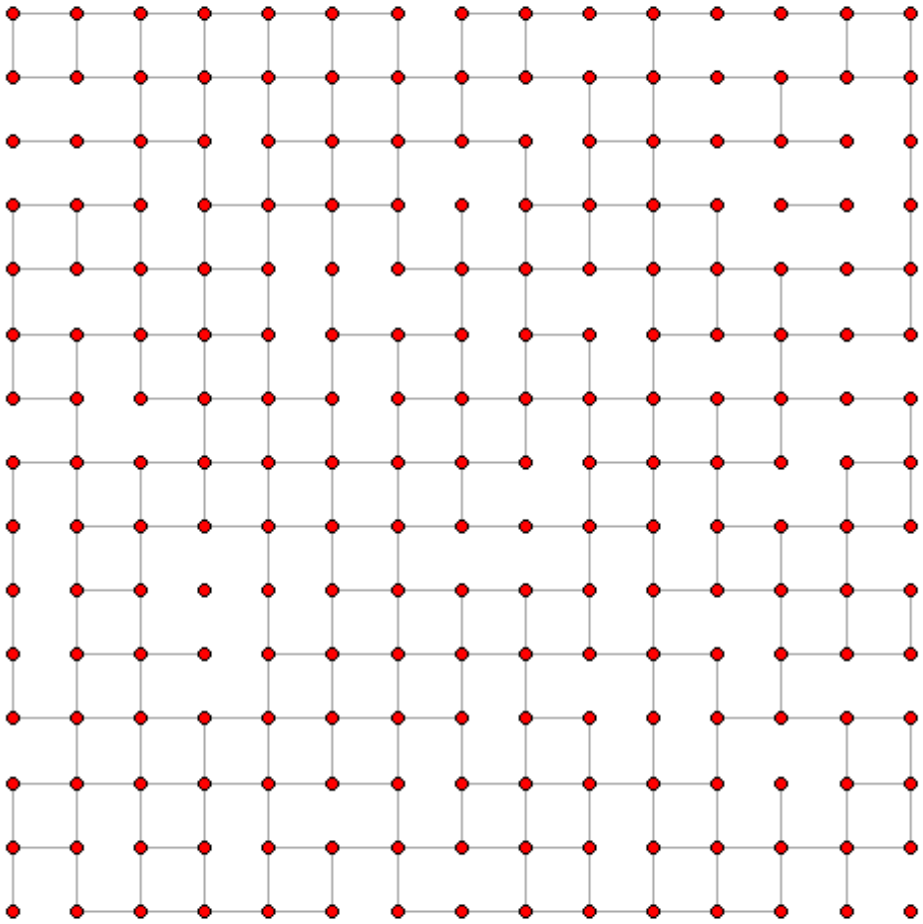

Figure A.16: 15×15 lattice with percolation probability $p = 0.8$

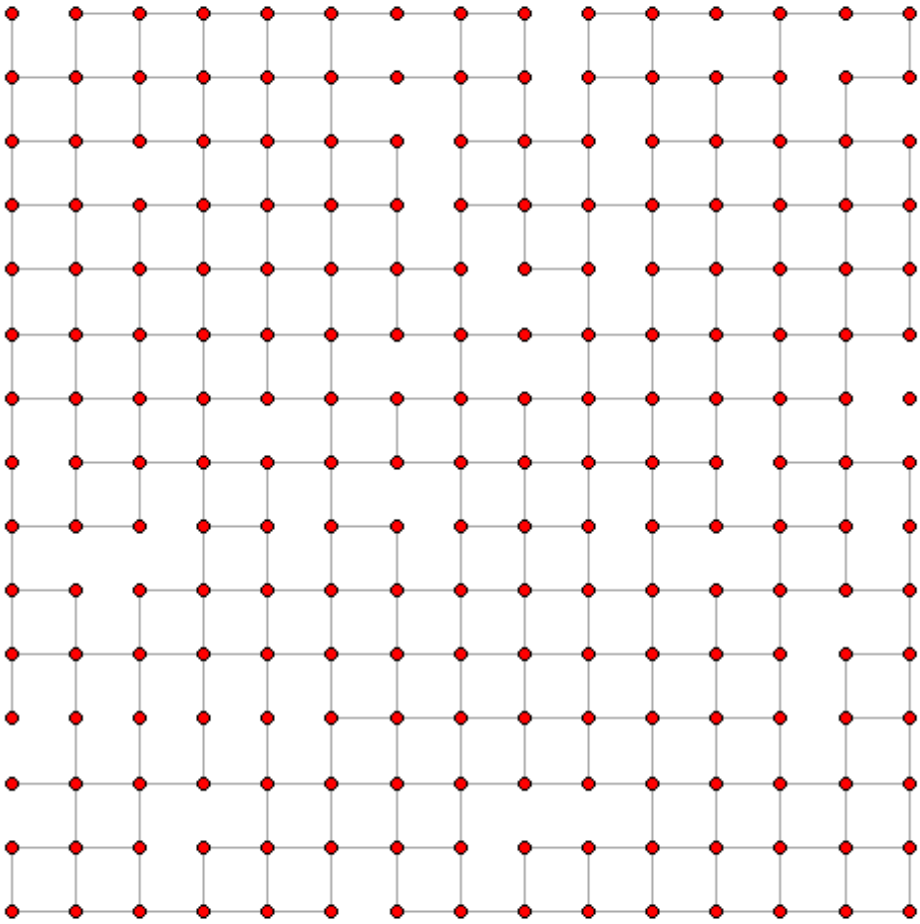

Figure A.17: 15×15 lattice with percolation probability $p = 0.9$

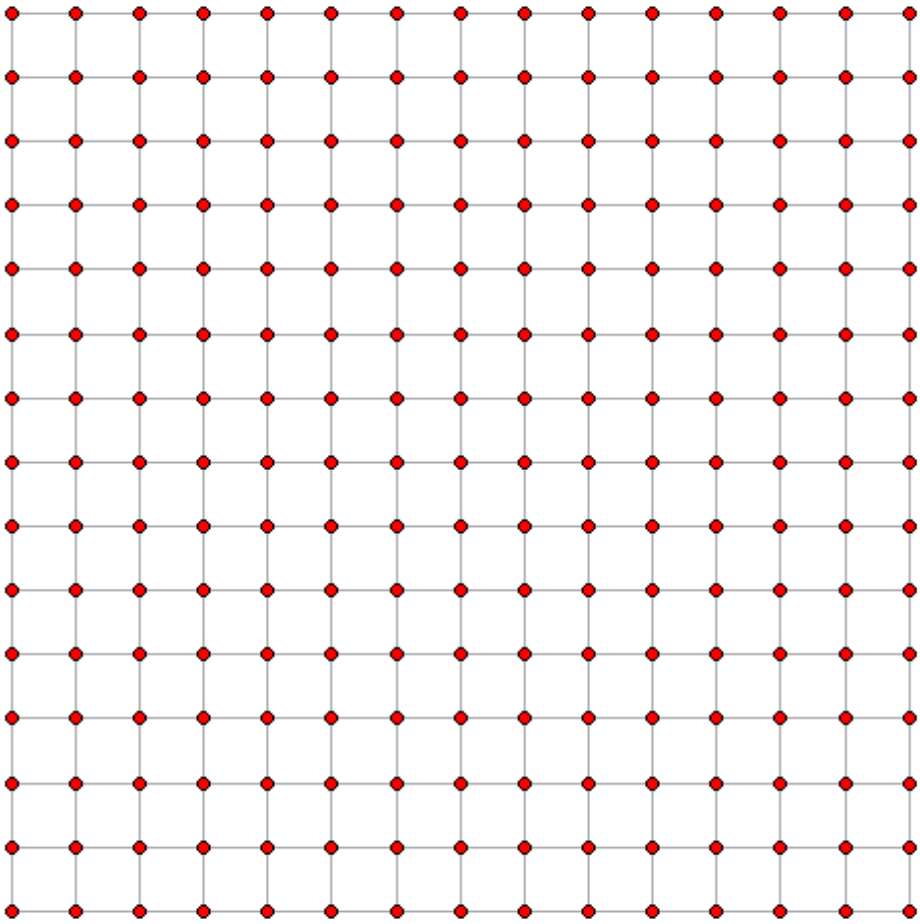

Figure A.18: 15×15 lattice with percolation probability $p = 1$

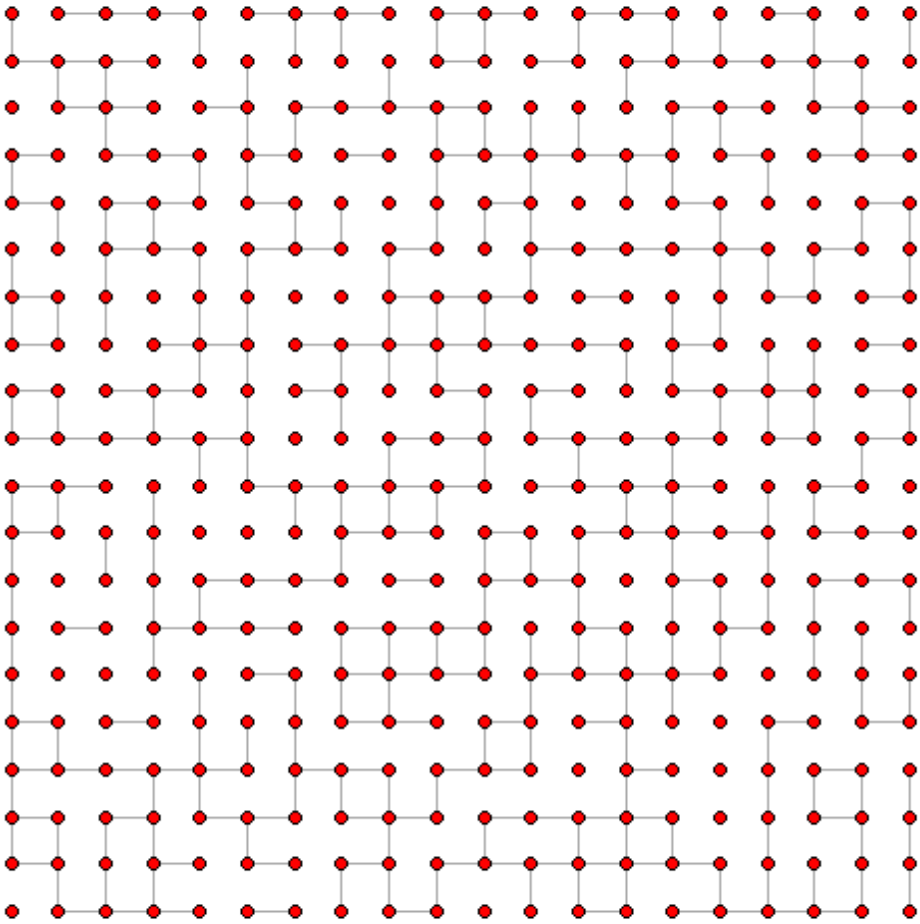

Figure A.19: 20×20 lattice with percolation probability $p = 0.5$

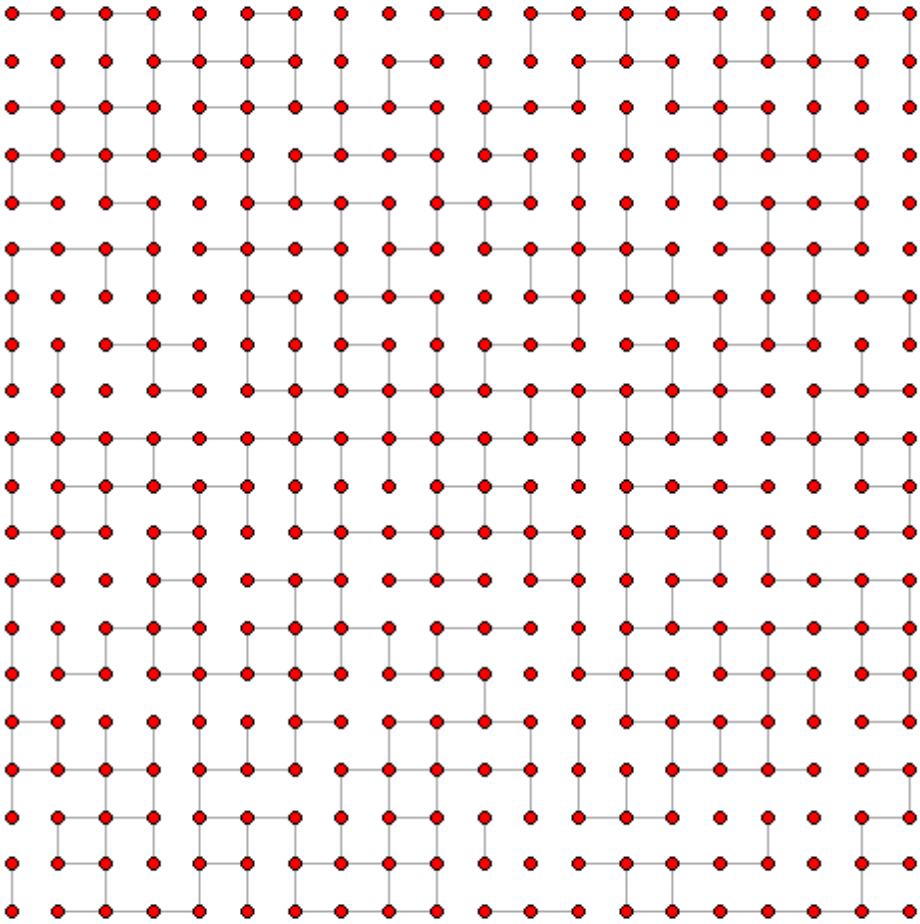

Figure A.20: 20×20 lattice with percolation probability $p = 0.6$

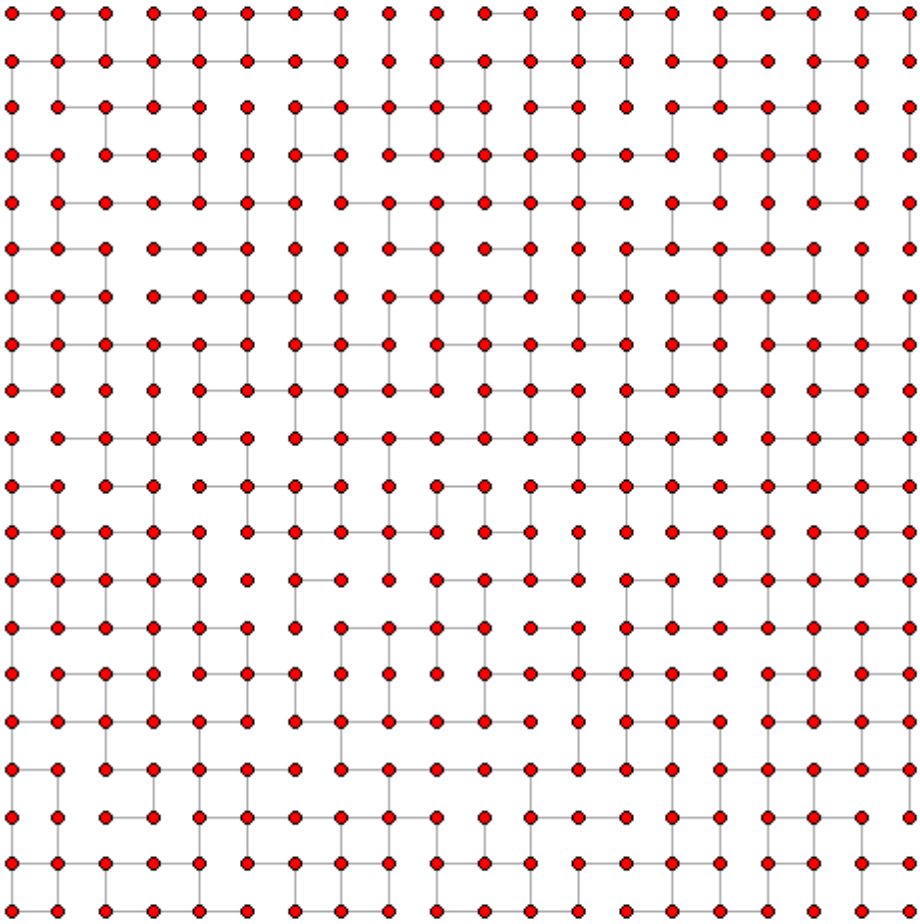

Figure A.21: 20×20 lattice with percolation probability $p = 0.7$

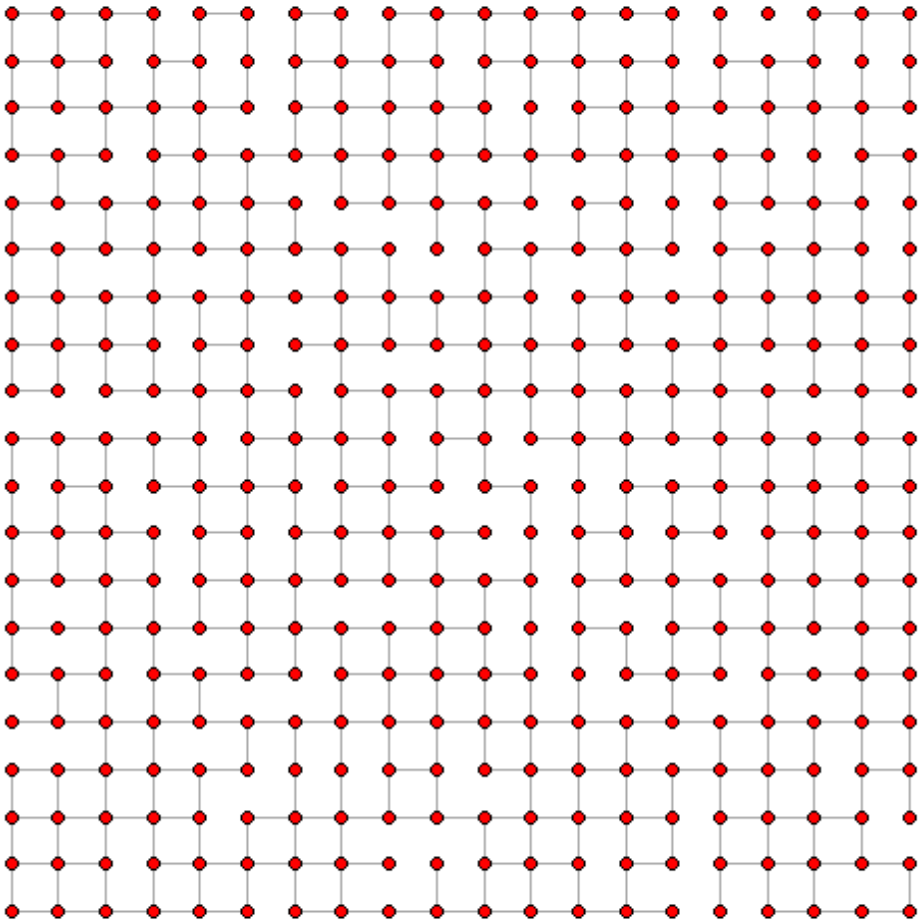

Figure A.22: 20×20 lattice with percolation probability $p = 0.8$

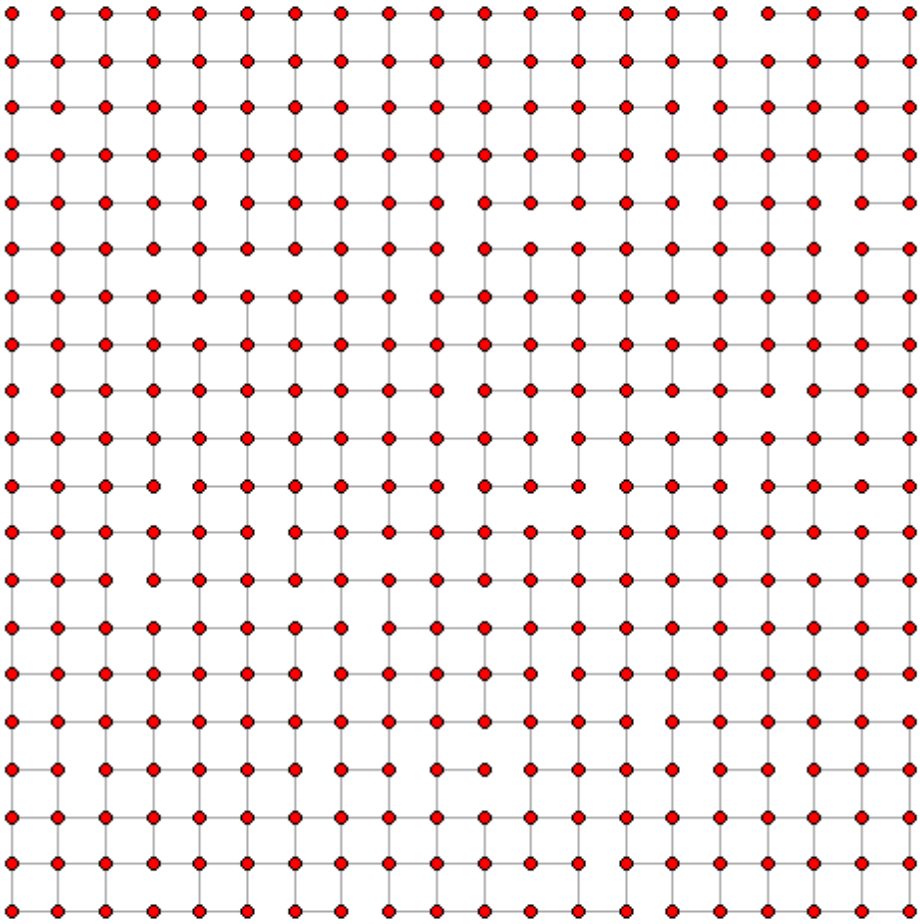

Figure A.23: 20×20 lattice with percolation probability $p = 0.9$

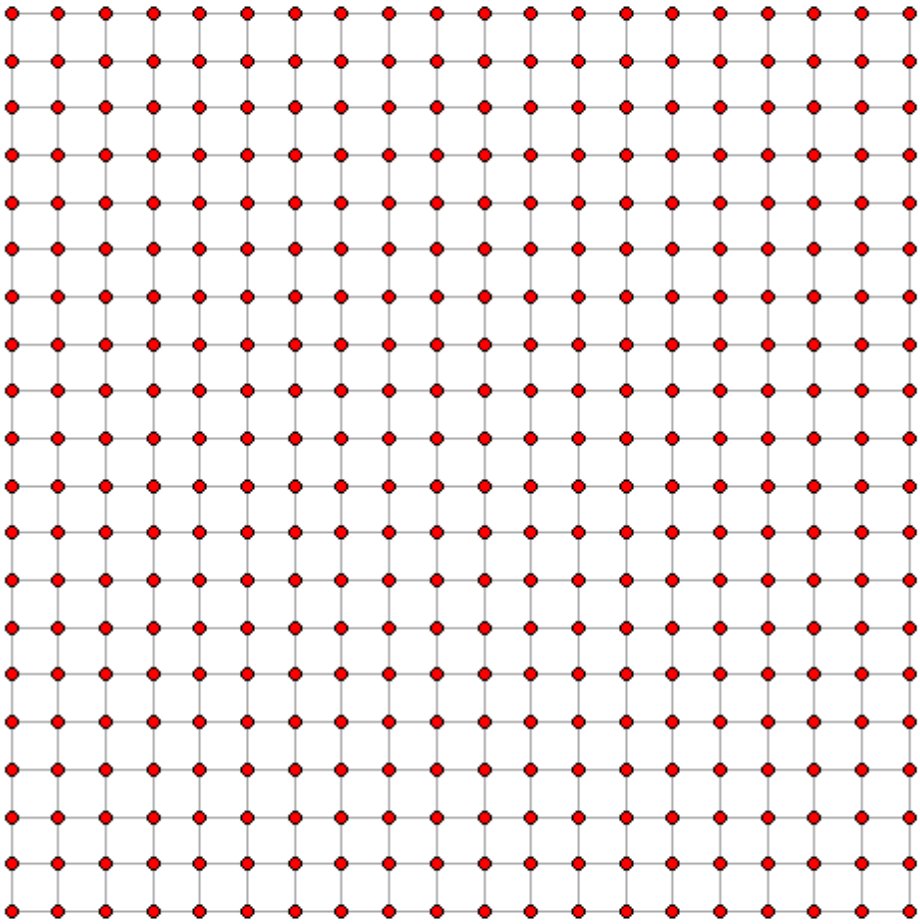

Figure A.24: 20×20 lattice with percolation probability $p = 1$

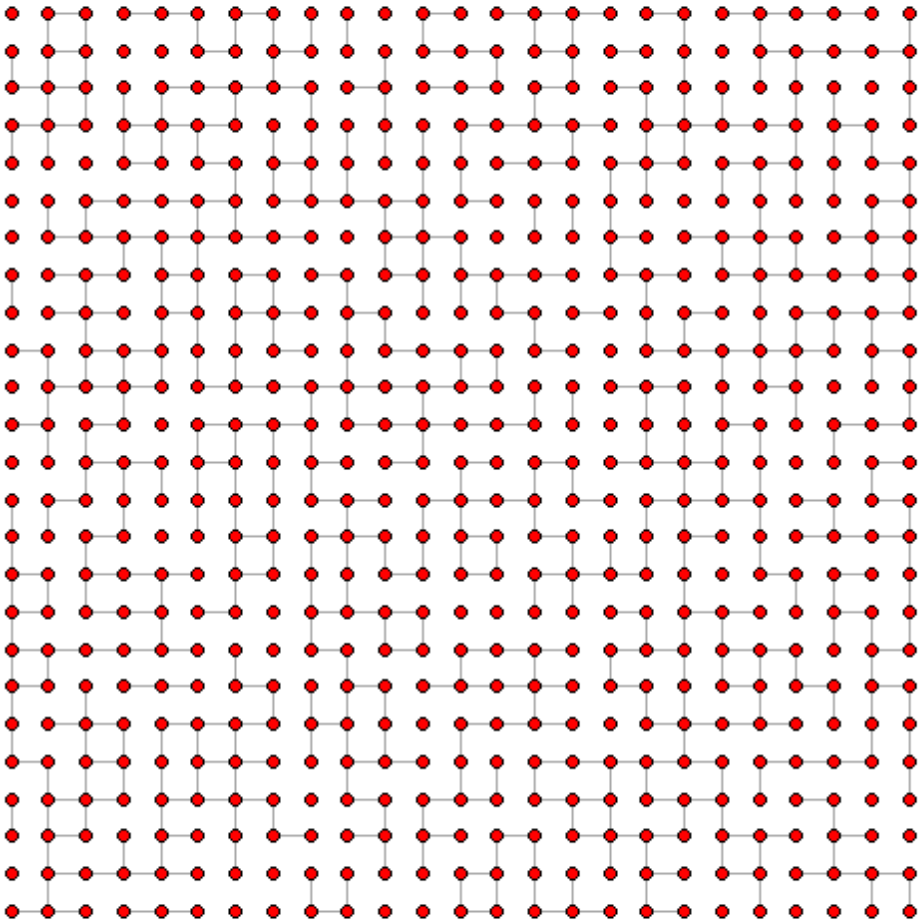

Figure A.25: 25×25 lattice with percolation probability $p = 0.5$

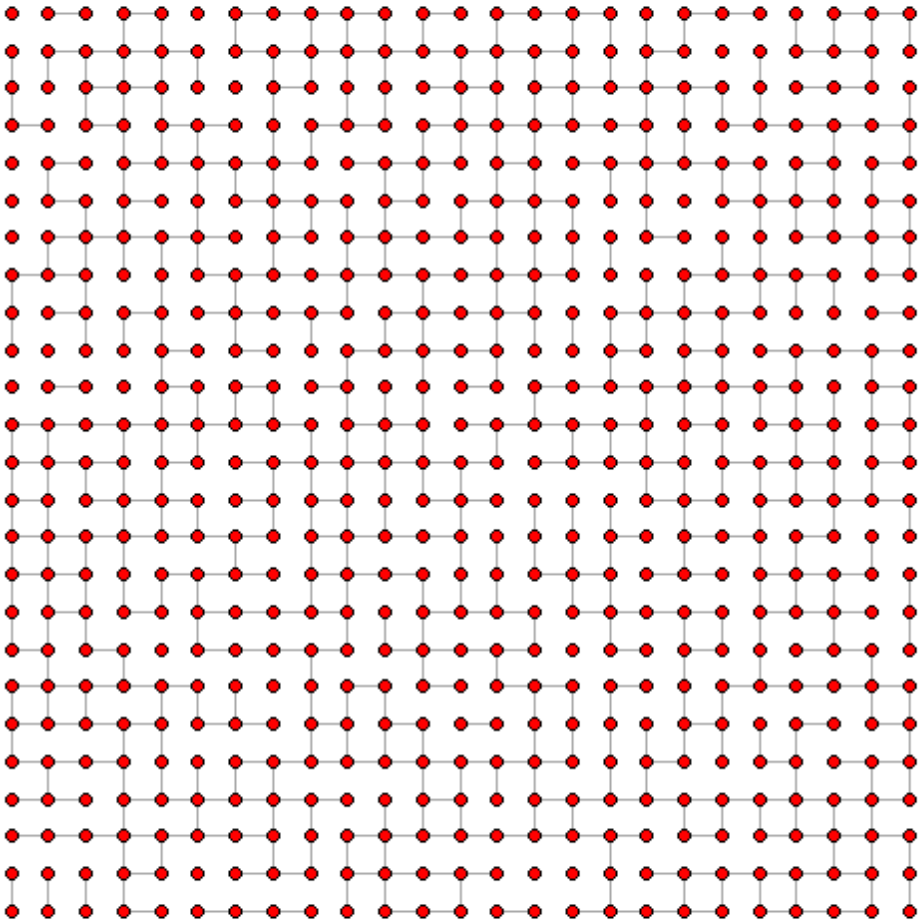

Figure A.26: 25×25 lattice with percolation probability $p = 0.6$

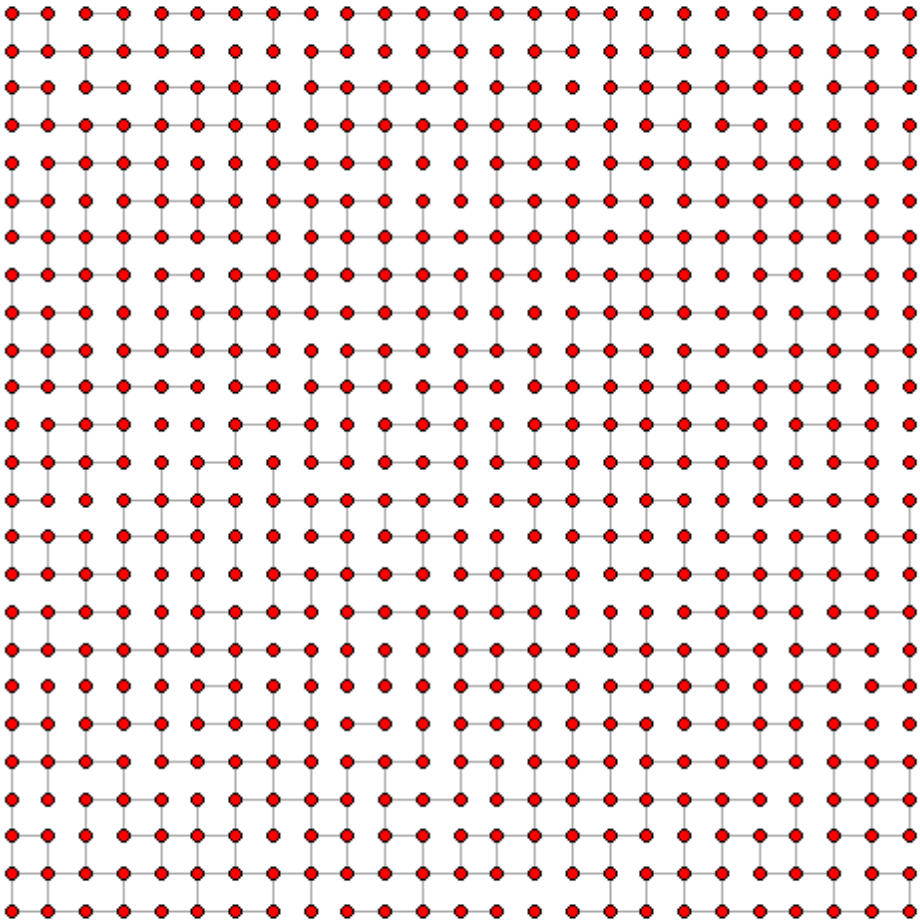

Figure A.27: 25×25 lattice with percolation probability $p = 0.7$

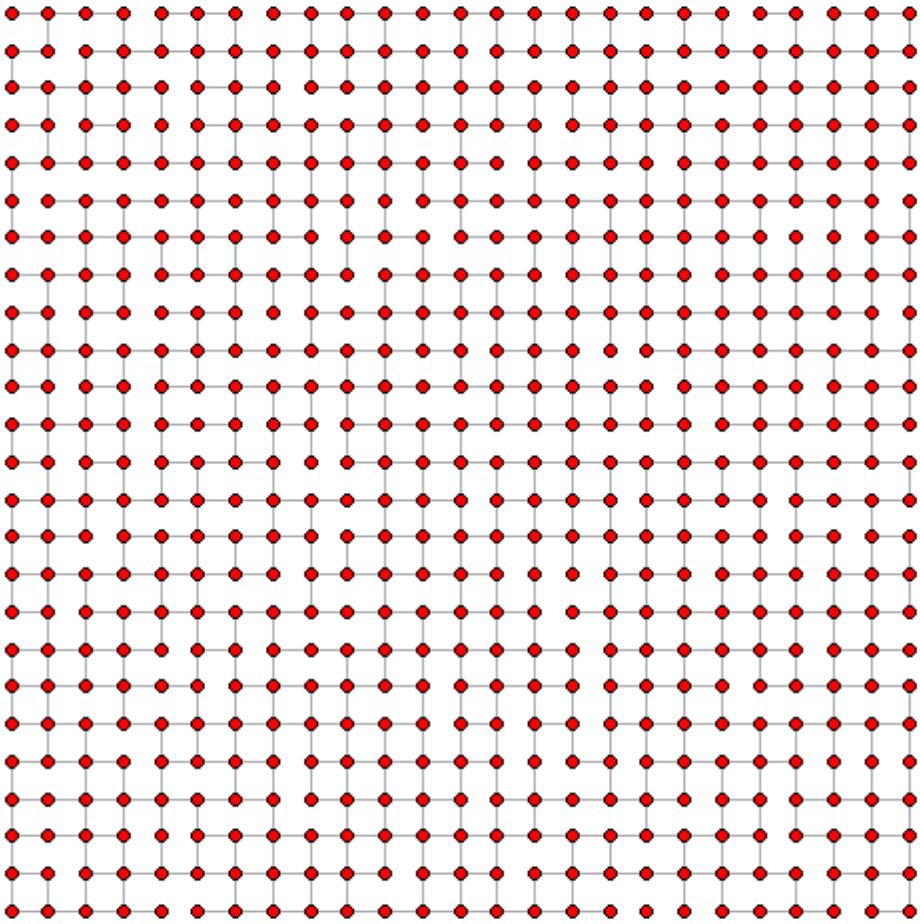

Figure A.28: 25×25 lattice with percolation probability $p = 0.8$

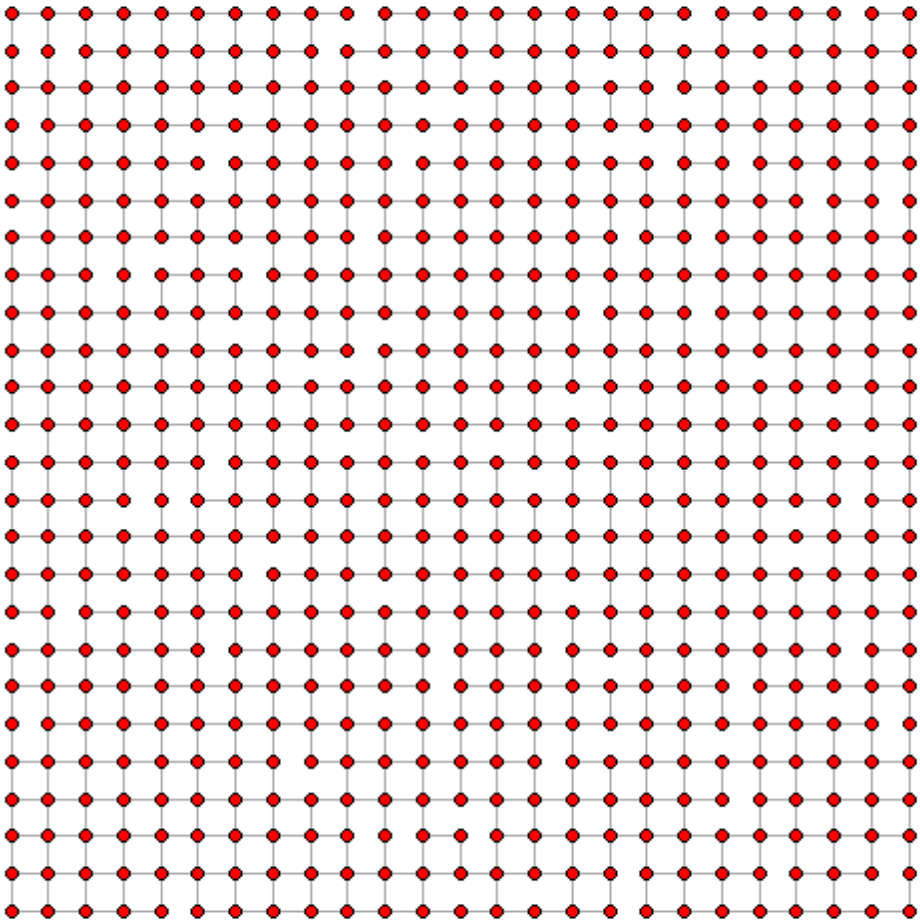

Figure A.29: 25×25 lattice with percolation probability $p = 0.9$

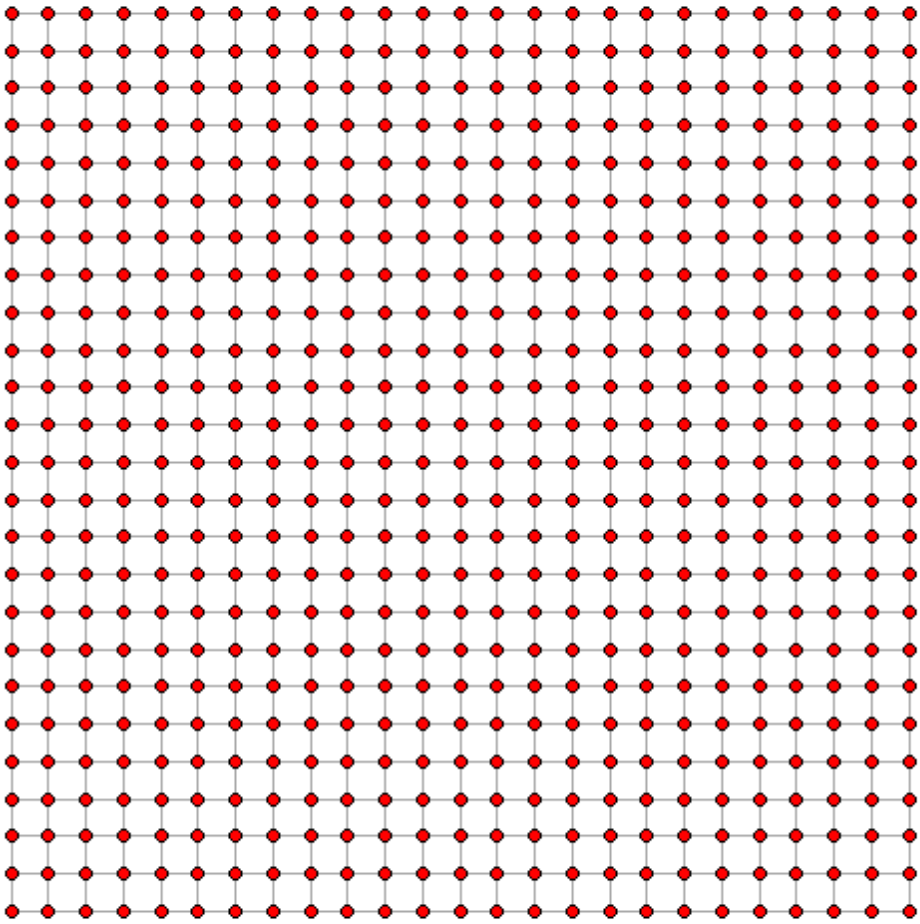

Figure A.30: 25×25 lattice with percolation probability $p = 1$

Below we will present the results that led to the plot in Figure 9.1.

We start with the results for the 10×10 lattice with percolation.

Table A.4: $p = 0.5$

λ										
0	102	102	102	102	102	102	102	102	102	102
0.2	160	138	154	160	148	148	156	150	136	140
0.3	194	162	168	166	130	166	214	144	168	216
0.4	198	364	184	222	222	182	206	262	246	228
0.5	398	234	258	306	266	290	282	182	166	322
0.6	382	582	624	654	476	440	330	448	290	414
0.8	908	1618	1446	1824	610	582	1664	2001	760	2001
0.9	2001	1958	1278	1310	2001	2001	2001	736	1210	2001
1	2001	2001	800	1608	2001	2001	2001	2001	1330	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Table A.5: $p = 0.6$

λ										
0	102	102	102	102	102	102	102	102	102	102
0.2	138	158	160	160	166	154	172	152	152	142
0.3	232	182	216	182	182	214	208	188	248	200
0.4	294	234	258	318	240	238	322	264	320	176
0.5	342	372	294	424	312	234	366	426	342	532
0.6	710	324	548	606	282	928	402	452	736	558
0.8	2001	2001	2001	1064	2001	2001	2001	2001	2001	1620
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Table A.6: $p = 0.7$

λ										
0	102	102	102	102	102	102	102	102	102	102
0.2	186	198	142	174	158	158	164	164	162	200
0.3	268	326	232	282	260	246	220	290	258	254
0.4	398	438	262	372	268	258	312	388	274	460
0.5	806	1814	1268	792	832	760	672	1306	590	1180
0.6	650	2001	2001	964	1556	488	1826	2001	1668	950
0.8	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Table A.7: $p = 0.8$

λ										
0	102	102	102	102	102	102	102	102	102	102
0.2	186	158	168	170	168	150	160	170	158	192
0.3	270	276	232	256	310	256	214	206	252	186
0.4	262	458	414	414	358	378	382	430	446	414
0.5	490	876	986	934	654	456	424	538	1426	816
0.6	2001	1170	2001	2001	2001	2001	1788	994	1528	1032
0.8	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Table A.8: $p = 0.9$

λ										
0	102	102	102	102	102	102	102	102	102	102
0.2	238	194	226	224	156	182	216	218	190	272
0.3	378	296	290	264	224	284	192	258	226	254
0.4	464	440	448	372	730	424	480	468	564	302
0.5	990	2001	2001	2001	1458	714	444	1318	500	1552
0.6	2001	2001	2001	2001	2001	2001	1988	2001	2001	2001
0.8	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Table A.9: $p = 1$

λ										
0	102	102	102	102	102	102	102	102	102	102
0.2	220	210	220	202	198	240	194	248	228	216
0.3	272	414	364	348	330	428	562	386	422	274
0.4	1682	1040	1154	422	934	1714	890	482	996	1410
0.5	2001	2001	2001	2001	1636	2001	2001	1142	2001	2001
0.6	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.8	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Next, we present the results for the 15×15 lattice with percolation.

Table A.10: $p = 0.5$

λ										
0	227	227	227	227	227	227	227	227	227	227
0.2	383	315	275	383	343	321	317	359	359	363
0.3	329	415	397	367	463	393	425	371	431	385
0.4	485	559	487	587	535	517	489	457	683	561
0.5	621	707	563	853	619	689	651	713	521	813
0.6	971	971	973	1237	661	1031	937	1175	775	611
0.8	2001	2001	2001	1925	2001	2001	2001	1673	2001	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Table A.11: $p = 0.6$

λ										
0	227	227	227	227	227	227	227	227	227	227
0.2	369	367	359	373	365	379	311	351	323	343
0.3	415	395	461	427	373	435	417	381	383	387
0.4	539	645	561	593	507	619	643	567	639	613
0.5	573	649	721	701	611	775	1057	873	945	869
0.6	1391	1403	1271	825	1133	895	1387	1093	1123	1195
0.8	2001	2001	2001	2001	2001	2001	2001	2001	1247	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Table A.12: $p = 0.7$

λ										
0	227	227	227	227	227	227	227	227	227	227
0.2	407	329	389	387	393	355	377	395	405	411
0.3	497	561	527	425	533	505	619	461	463	541
0.4	949	577	709	915	663	649	987	635	807	969
0.5	1183	1581	1109	1161	825	1067	1249	1181	1293	1705
0.6	1717	1811	2001	2001	2001	1461	2001	2001	2001	2001
0.8	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Table A.13: $p = 0.8$

λ										
0	227	227	227	227	227	227	227	227	227	227
0.2	411	445	435	407	415	399	459	447	407	395
0.3	671	599	631	563	585	645	597	523	519	631
0.4	1767	881	1265	1125	771	873	931	1077	1341	1061
0.5	2001	2001	2001	2001	1709	2001	2001	1759	1919	2001
0.6	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.8	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Table A.14: $p = 0.9$

λ										
0	227	227	227	227	227	227	227	227	227	227
0.2	513	447	421	415	473	481	499	429	495	423
0.3	713	719	617	765	651	653	993	841	751	695
0.4	1041	1387	1169	1477	1863	1887	1087	1503	1883	785
0.5	2001	2001	2001	2001	2001	2001	2001	2001	2001	1687
0.6	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.8	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Table A.15: $p = 1$

λ										
0	227	227	227	227	227	227	227	227	227	227
0.2	423	425	513	509	465	461	503	517	529	529
0.3	871	995	905	747	791	815	907	913	977	845
0.4	2001	1213	1343	2001	2001	2001	1695	2001	2001	2001
0.5	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.6	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.8	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Next, we present the results for the 20×20 lattice with percolation.

Table A.16: $p = 0.5$

λ										
0	402	402	402	402	402	402	402	402	402	402
0.2	652	634	652	646	586	644	648	650	636	620
0.3	698	766	788	694	736	766	746	782	698	766
0.4	804	936	978	1006	796	1100	1020	946	896	966
0.5	1106	1156	1042	1356	1214	1086	1186	1024	1132	1396
0.6	1670	1288	1554	1870	1426	1626	1860	2001	1442	1544
0.8	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Table A.17: $p = 0.6$

λ										
0	402	402	402	402	402	402	402	402	402	402
0.2	644	612	650	642	644	582	658	598	608	612
0.3	862	928	720	730	868	806	842	870	958	826
0.4	1250	858	1132	1380	1064	1076	1032	1172	1124	1078
0.5	1506	1844	1936	1466	1456	1456	1390	1348	1760	2001
0.6	2001	2001	2001	2001	2001	2001	2001	2001	2001	1996
0.8	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Table A.18: $p = 0.8$

λ										
0	402	402	402	402	402	402	402	402	402	402
0.2	658	712	686	648	654	666	648	658	698	638
0.3	888	920	970	1040	964	1056	948	852	976	862
0.4	1450	1842	1254	1116	1406	1478	1308	1186	1250	1248
0.5	2001	2001	2001	1972	2001	1906	2001	2001	1608	1820
0.6	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.8	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Table A.19: $p = 0.8$

λ										
0	402	402	402	402	402	402	402	402	402	402
0.2	782	768	866	696	772	712	762	856	724	700
0.3	1098	1124	950	1126	1032	1232	1036	1186	1156	1004
0.4	1844	1520	2001	2001	1694	1704	2001	1786	2001	1612
0.5	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.6	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.8	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Table A.20: $p = 0.9$

λ										
0	402	402	402	402	402	402	402	402	402	402
0.2	900	810	808	928	868	812	930	786	882	786
0.3	1370	1422	1292	1158	1364	1304	1570	1208	1598	1236
0.4	2001	2001	2001	2001	2001	2001	1994	2001	2001	2001
0.5	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.6	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.8	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Table A.21: $p = 1$

λ										
0	402	402	402	402	402	402	402	402	402	402
0.2	942	916	772	1038	864	856	910	954	894	984
0.3	1528	1896	1738	1810	1744	1430	1474	1730	1718	1890
0.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.5	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.6	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.8	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Next, we present the results for the 25×25 lattice with percolation.

Table A.22: $p = 0.5$

λ										
0	627	627	627	627	627	627	627	627	627	627
0.2	967	981	921	939	935	895	921	1019	951	993
0.3	1171	1105	1301	1085	1153	1109	1289	1123	1289	1103
0.4	1445	1455	1479	1463	1461	1409	1351	1381	1641	1467
0.5	1547	1857	1865	2001	1807	1863	1681	1829	2001	1997
0.6	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.8	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Table A.23: $p = 0.6$

λ										
0	627	627	627	627	627	627	627	627	627	627
0.2	1009	1051	1061	1031	995	1011	985	1003	1015	969
0.3	1307	1245	1483	1365	1303	1509	1357	1365	1389	1279
0.4	1841	2001	1697	1919	1771	1653	1671	1911	1633	1983
0.5	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.6	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.8	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Table A.24: $p = 0.7$

λ										
0	627	627	627	627	627	627	627	627	627	627
0.2	1115	1109	1021	1063	1045	1121	1147	1113	1013	1049
0.3	1515	1395	1391	1515	1467	1403	1353	1453	1453	1525
0.4	2001	2001	1791	2001	1973	2001	1761	2001	2001	1981
0.5	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.6	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.8	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Table A.25: $p = 0.8$

λ										
0	627	627	627	627	627	627	627	627	627	627
0.2	1213	1249	1219	1295	1113	1195	1137	1143	1327	1235
0.3	1775	1767	1501	1835	1849	1861	1925	1823	1831	1427
0.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.5	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.6	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.8	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Table A.26: $p = 0.9$

λ										
0	627	627	627	627	627	627	627	627	627	627
0.2	1115	1313	1289	1321	1361	1287	1363	1285	1425	1237
0.3	2001	2001	2001	2001	1865	2001	2001	1959	1865	2001
0.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.5	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.6	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.8	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Table A.27: $p = 1$

λ										
0	627	627	627	627	627	627	627	627	627	627
0.2	1415	1481	1521	1479	1323	1441	1499	1307	1521	1381
0.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.5	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.6	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.8	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
0.9	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.2	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.3	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001
1.4	2001	2001	2001	2001	2001	2001	2001	2001	2001	2001

Appendix B

Source Code

Netshield source code

```
Netshield <- function(G,k,idx_out)
{
###find k nodes, and if we delete them, will produce maximum drop in terms
###of the 1st eigen-value of A
#A is the given graph
#k is the number of nodes to delete
#idx is the index of deleted nodes
#del is the difference of 1st eigen-value of A after deleting the nodes

A <- as.matrix(get.adjacency(G))
if (nargs()<3)
{
idx_out <- c()
}

if (k<0)
{
idx <- -1
return
}
###pre-processing? (e.g., to exclude those degree-1 nodes)

spectrum <- eigen(A, symmetric = TRUE, only.values = FALSE)
# make sure all elements of u positive"
u <- spectrum$vectors[,1]
```

```
lam <- spectrum$values[1]
pos <- which(abs(u)==max(abs(u)))
if (u[pos[1]] < 0)
{
u <- -u
}
n <- dim(A)[1]
u0 = (2 * lam * rep(1,n) - diag(A))*(u^2)
#top 1
tmp <- u0
tmp[idx_out] <- -1
pos <- which(tmp==max(tmp))
idx <- pos[1]
###greedily find the other nodes
if(k>1)
{
for (i in 2:k)
{
A0 <- A[,idx]
r <- as.matrix(u[idx])
tmp <- A0 %*%r

tmp <- u0 - 2 * ((tmp)*u)
tmp[idx] <- -1 #exclude those already selected
tmp[idx_out] <- -1
pos <- which(tmp==max(tmp))[1]
idx <- append(idx,pos)
}
}

A0 <- A
A0[,idx] <- 0
A0[idx,] <- 0
spectrum <- eigen(A0, symmetric = TRUE, only.values = FALSE)
u00 <- spectrum$vectors
lam00 <- spectrum$values[1]
del <- lam - lam00

return(list(first = idx, second = max(tmp) ,third = del))
}
```

Netshield+ source code

```
 Netshield_plus <- function( G,k,batch,idx_out )
 {
# find k nodes, and if we delete them, will produce maximum drop in terms
# of the 1st eigen-value of A
# greedy way by matrix perturbation theory and submodularity
# A is the given graph
# k is the number of nodes to delete
# batch is unit delete size
# idx is the index of deleted nodes
# del is the difference of 1st eigen-value of A after deleting the nodes


A <- as.matrix(get.adjacency(G))

if( nargs()<4)
 {
idx_out <- c()
 }

if (k<0)
{
totalidx <- -1
return
}


round = ceiling(k/batch)
totaldel = 0
totalidx = c()
Ao = A
for (r in 1:round)
{

spectrum <- eigen(Ao, symmetric = TRUE, only.values = FALSE)
# make sure all elements of u positive
u <- spectrum$vectors[,1]
lam <- spectrum$values[1]
pos <- which(abs(u)==max(abs(u)))
```

```
# u <- ifelse(u<0, u*-1, u)
if (u[pos[1]] < 0)
{
  u <- -u
}

n <- dim(A)[1]
u0 = (2 * lam * rep(1,n) - diag(Ao))*(u^2)
#top 1
tmp <- u0
tmp[idx_out] <- -1
pos <- which(tmp==max(tmp))
idx <- pos[1]

if (k > 1)
{

if(r == round && k%%batch !=0)
{
batch = k-(round-1)*batch;
}

if (batch >=2)
{
#greedily find the other nodes
for (i in 2:batch)
{
  A0 <- A[,idx]
  r <- as.matrix(u[idx])
  tmp <- A0 %*%r
  tmp <- u0 - 2 * ((tmp)*u)
  tmp[idx] <- -1 #exclude those already selected
  tmp[idx_out] <- -1
  pos <- which(tmp==max(tmp))[1]
  idx <- append(idx,pos)
}
}
}

# Ao <- A
Ao[,idx] <- 0
```

```
Ao[idx,] <- 0
spectrum <- eigen(Ao, symmetric = TRUE, only.values = FALSE)
u00 <- spectrum$vectors[,1]
lam00 <- spectrum$values[1]
# u00 <- spectrum$vectors
# lam00 <- spectrum$values
del <- lam - lam00

totaldel = totaldel + del;
totalidx = append(totalidx,idx)

}
return(list(first = totalidx, second = totaldel))

}
```

GA_0 source code - Based on Algorithm 1

```
# 1+1 GA #

GA_0 <- function(G,maxEval,k)
{
  start.time <- proc.time()
  # G is the graph/network

  A <- as.matrix(get.adjacency(G))

  # Eigendecomposition of adjacency matrix
  spectrum <- eigen(A, symmetric = TRUE, only.values = FALSE)
  lambda <- spectrum$values[1]
  if (lambda<0) {lambda <- lambda*-1}
  v <- spectrum$vectors[,1]
  v <- ifelse(v<0, v*-1,v) # make sure all elements of u positive
```

```
n <- vcount(G) # size of graph/number of nodes

pm = 1/n # mutation probability

evalcount <- 0
hist_fit <- c()

fit <- c()

# Initialize population and evaluate
S <- seq(n)
S <- sample(S)


fit <-  eigen_drop(S[1:k],G, lambda)
evalcount <- evalcount+1
hist_fit[evalcount] <-  fit


fit_new <- c()

# Evaluation loop (mutation)
while (evalcount < maxEval )
{
  # interchanging the vector elements

  t <- S
  for (i in 1:k)
  {
    ind_one <- t[1:k]
    ind_zero <- t[(k+1):n]
    random_zero <- sample(length(ind_zero),1, replace = FALSE,
                          rep(pm,length(ind_zero)))
    random_one <- sample(length(ind_one),1, replace = FALSE,
                         rep(pm,length(ind_one)))
    temp <- ind_zero[random_zero]
    t[(random_zero+k)] <- ind_one[random_one]
    t[random_one] <- temp
  }

  offspring_S <- t
```

```
    fit_new <- eigen_drop(offspring_S[1:k],G, lambda)


    J <- (fit_new)
    K <- (fit)

    if(J>K)
    {
      fit <- fit_new
      S <- offspring_S
    }


    evalcount <- evalcount+1
    hist_fit[evalcount] <-  fit


  }


  end.time <- proc.time()
  time.taken <- end.time - start.time
  return(list( first = S[1:k], second  = fit , third = hist_fit ,
               fourth = evalcount , fifth = time.taken))

}
```

GA_1,2,3,4,5 source code - Based on Algorithm 2

```
# (mue + mue)-GA #

GA_i <- function(G,maxEval,k, p)
{
  # G is the graph/network
  start.time <- Sys.time()

  # G is the graph/network

  A <- as.matrix(get.adjacency(G))

  # Eigendecomposition of adjacency matrix
  spectrum <- eigen(A, symmetric = TRUE, only.values = FALSE)
  lambda <- spectrum$values[1]
  if (lambda<0) {lambda <- lambda*-1}
  v <- spectrum$vectors[,1]
  v <- ifelse(v<0, v*-1,v) # make sure all elements of u positive
  sorted <- sort.int(v, decreasing = TRUE, index.return = TRUE)
  leigen_comp <- sorted$ix[1:k] # k largest eigenscore nodes
  n <- vcount(G) # size of graph/number of nodes


  # GA parameters
  mu = 50
  pc = 0.75
  pm = 1/n # mutation probability
  pool = 30 # pool size for tournament selection

  evalcount <- 0
  hist_fit <- c()



  # Initialize population and evaluate
  r <- seq(n)
  S <- matrix(0, nrow = mu, ncol = n)
  for ( i in 1:mu)
  {
    S[i,] <- sample(r)
```

```
}

fitness <- rep(0, mu)

for( i in 1:mu)
{

  fitness[i] <-  eigen_drop(S[i,(1:k)],G, lambda)

}

idx <- which(fitness == max(fitness))[1]
opt <- S[idx, 1:k]


evalcount <- evalcount+1
hist_fit[evalcount] <-  max(fitness)


fitness_new <- rep(0, mu)
temp <- 0
prev_fitness <- max(fitness)

# Evaluation loop (recombination - mutation)
while (evalcount < maxEval && temp < (k*(n-k)))
{

  # Generate new population
  # parent selection - tournament method
  offspring_S <- matrix(0, nrow = mu, ncol = n)

  for (j in 1:mu)
  {
    parent_1 <- select_scProportional(S, fitness)
    r <- runif(1)

    if(r < pc)
    {
      parent_2 <- select_scProportional(S, fitness)
```

```
  un <- union(parent_1[1:k], parent_2[1:k])

  if (length(un) > k)
  {
    dif <- length(un) - k
    for(l in 1:dif)
    {
      rm <- sample(un, 1, replace = FALSE)
      un <- un[-rm]
    }

  }

  t <- c(un, setdiff(seq(n),un))
}

else
{parent_2 <- parent_1; t <- parent_2}
# print(t)

prob <- rep(pm, n)
prob[which(t %in% leigen_comp)] <- p


for (i in 1:k)
{
  ind_one <- t[1:k]
  ind_zero <- t[(k+1):n]
  random_zero <- sample(length(ind_zero),1, replace = FALSE,
                                                  prob[(k+1):n])
  random_one <- sample(length(ind_one),1, replace = FALSE,
                                                    prob[1:k])
  random_zero <- ind_zero[random_zero]
  random_one <- ind_one[random_one]
  ind1 <- which(t == random_one)
  ind2 <- which(t == random_zero)
  t <- replace(t, c(ind1,ind2), t[c(ind2, ind1)])
}
```

```
  offspring_S[j,] <- t

}



Union <- rbind(S, offspring_S)



for( i in 1:mu)
{

  fitness_new[i] <-  eigen_drop(offspring_S[i,(1:k)],G, lambda)
}

fitness_union <- append(fitness,fitness_new )

ftns <- sort.int(fitness_union, decreasing = TRUE, index.return = TRUE)
S <- Union[ftns$ix[c(1:mu)],]
fitness <- fitness_union[ftns$ix[c(1:mu)]]



idx <- which(fitness == max(fitness))[1]
opt <- S[idx, (1:k)]


evalcount <- evalcount+1
hist_fit[evalcount] <-  max(fitness)


# termination criterion - stagnation

if (max(fitness) - prev_fitness == 0)
{
  temp <- temp + 1
  # print(temp)
}
else {temp <- 0}
```

```
    prev_fitness <- max(fitness)
  }


  end.time <- Sys.time()

  time.taken <- end.time - start.time
  return(list( first = opt, second  = max(fitness), third = hist_fit ,
                            fourth = evalcount, fifth = time.taken))

}
```

MultiObjective GA source code - Based on the 'ecr' package by Jakob Bossek

```
#########################################
# MOEA / SMS - EMOA / NSGA-II           #
# Assumptions:                          #
# G is a connected graph                #
# k-invariant                           #
#########################################



bi_Objective_GA <- function(G, maxEval)
{
  # visits <- c(117,126,90,102,123,120,117,123,105,135,108,123,
  # 84,120,120,120,138,108,117,123,99,114,141,99,81,84, 117)
  # populations <- c(723724,2830144,3280123,8124427, 548359,
  # 424096,3146804,7489022,2141839, 596204,1316218,4991000,2029936,
  # 10034830, 10472629, 7836243, 5753612,7160094,11215130,11969284,
  # 4328067, 18410000,  7088000,  4497000,
  # 7602069,  9860000,  8372440, 15017783,  7347000,  2491662,2590815,
```

```
# 4935988,  3496586, 10546511,  8556798,
# 4444513,  2090001,  2091491,  9464000,  9020089, 10059502, 11595183,
# 4893495,  7857121,  7235084,
# 8659409,  3911500,   386740 )

# Uncomment below, depending on the cost function

# G <- set.vertex.attribute(G, 'importance', V(G), degree(G)*betweenness(G,
                                                    #V(G), directed = FALSE))
# G <- set.vertex.attribute(G, 'pop', V(G), populations)
# G <- set.vertex.attribute(G, 'pop', V(G), visits )
A <- as.matrix(get.adjacency(G))
# Eigendecomposition of adjacency matrix
spectrum <- eigen(A, symmetric = TRUE, only.values = FALSE)
eigen_lambda <- spectrum$values[1]
if (eigen_lambda<0) {eigen_lambda <- eigen_lambda*-1}
v <- spectrum$vectors[,1]
v <- ifelse(v<0, v*-1,v)

n <- vcount(G)  # size of graph/number of nodes

sorted <- sort.int(v, decreasing = TRUE, index.return = TRUE)
ind <- sorted$ix


# GA parameters

mu = 50L # population size
pc = 0.75 # recombination probability
pm = 1/n # mutation probability

ref.point <- c(1,10000L)
lambda =1L # change to 50L in NSGA-II EMOA
fit_Matrix <- matrix(0, nrow = 2, ncol = (mu+lambda))

# uncomment below depending on the solver algorithm

survival.selector =     setup(selDomHV, ref.point = ref.point) # SMS-EMOA
                                                #  selNondom # NSGA-II EMOA #
recombinator = recCrossover
parent.selector = selSimple
```

```
mutator = mutBitflip

# initialization

population = genBin(mu,n)


control = initECRControl(fitFunction3, n.objectives = 2L,
                         minimize = c(FALSE, TRUE))
control = registerECROperator(control, "selectForMating",
                              parent.selector)
control = registerECROperator(control, "recombine",
                              recombinator)
control = registerECROperator(control, "mutate",
                              mutator)
control = registerECROperator(control, "selectForSurvival",
                              survival.selector)

# fitness calculation

fitness = evaluateFitness(control, population, G, eigen_lambda)


##############################################################################
# Evaluation loop (recombination - mutation) in a steady-state manner
#
##############################################################################


for(iter in 1:maxEval)
{

  print(iter)

  # parent selection and mutation #


  offspring = recombinate(control, population, fitness = fitness,
              lambda = lambda, p.recomb = pc, slot = 'recombine')

  offspring <- mutation(offspring,ind,n)
```

```
    # calculate costs of new schedules
    fitness.o = evaluateFitness(control, offspring, G, eigen_lambda)

    # apply (MU + LAMBDA) selection

    sel = replaceMuPlusLambda(control, population, offspring,
                              fitness, fitness.o)
    population = sel$population
    fitness = sel$fitness



  }


  return(list(fitness = fitness, inds = population))

}
```

Cost function for the MultiObjective GA source code

```
##############################################################################
# The cost function has a as a 1st objective the eigen-drop (maximization)
#
# and as a 2nd objective the cost of immunization defined as the sum of the #
# nodes to be immunized/removed.
#
##############################################################################



fit_Function <- function(S, G, lambda)
```

```
{

  nodes <- which(S==1)
  G_prime <- delete.vertices(G, nodes)
  A_prime <- as.matrix(get.adjacency(G_prime))
  if(dim(A_prime)[1]!=0){
    spectrum_prime <- eigen(A_prime, symmetric = TRUE, only.values = FALSE)
    lambda_prime <- spectrum_prime$values[1]

    delta_eigen <- lambda - lambda_prime}
  else
    delta_eigen <- 0
  # print(lambda_prime)
  # print(eigen_drop)

  cost <- sum(V(G)[nodes]$pop)

  c(delta_eigen, cost)

}
```

Ordinary (bond) percolation source code

```
  ##########################################
  # ordinary (bond) percolation            #
  ##########################################

  bond_percolation <- function(G,p)
  {
    edge <- get.edgelist(G)
    edges_to_delete <- c()
    temp <- 0
    for (i in 1:dim(edge)[1])
```

```
  {

    r <- runif(1)
    if (r < (1-p)) { temp <- temp + 1; edges_to_delete[temp] <- i }

  }


  G_new <-  delete_edges(G, edges_to_delete)


  V(G_new)$color <- 'white'
  cluster <- clusters(G_new)
  colors <- rainbow(max(membership(cluster)))
  max_ind <- which(clusters(G_new)$csize==max(clusters(G_new)$csize))[1]
  ind <- which(clusters(G_new)$membership==max_ind)
  V(G_new)[ind]$color <- 'red'

  plot(G_new, layout=layout_on_grid,
       vertex.size=3, vertex.label = NA, edge.width = 1)

  print(cluster$csize)
  return(G_new)


}
```

CTMC simulation for the SIS epidemic spread on the square lattice

```
#################################################
# Stochastic Simulation Algorithm (CTMC)   #
#################################################
```

```
SSA <- function(G,b,d)
{
  V(G)$state <- rep(1,vcount(G)) # all nodes made infected
  n_inf <- c()
  n_inf[1] <- sum(V(G)$state==1)
  # print(n_inf)
  n_susc <- c()
  n_susc[1] <- 0
  rate <- rep(0, vcount(G))
  alive <- TRUE
  t <- 0
  time <- c()
  time[1] <- 0

  iter <- 1
  while( alive && iter <= 2000)
  {
    print(iter)
    iter <- iter + 1
    # print(V(G)$state)
    for (j in 1:vcount(G))
    {
      # print(j)
      #### Constructing the infection-rate Matrix ####

      output <- ifelse(V(G)[j]$state == 0, make_rates(G,j,b), d)
      # print(output)

      rate[j] <- output


    }

    if(sum(rate)>0)
    {alive <- TRUE}
    else
    {alive <- FALSE}

    if(alive)
    {
      rate_of_leaving <- sum(rate)
```

```
      DeltaT <- rexp(1,rate_of_leaving)
      sumprob <- 0
      u <- runif(1)

      for ( j in 1:vcount(G))
      {
        prev_sumprob <- sumprob
        sumprob <- sumprob + rate[j]/(sum(rate))
        if( prev_sumprob < u && u <= sumprob )
        {
          if (V(G)$state[j] == 0)
          {V(G)$state[j] <- 1}
          else if (V(G)$state[j] == 1)
          {V(G)$state[j] <- 0}
          break
        }


      }
    n_inf[iter] <- sum(V(G)$state==1)
    t <- t + DeltaT
    time <- append(time, DeltaT)
    }



  }

  return(list(graph = V(G)$state, time = time, iters = iter,
              infected = n_inf/vcount(G)))

}




make_rates <- function(G,j,b)
{

  nb <- neighbors(G,V(G)[j], mode=c("all"))
```

```
  v <- sum(V(G)[nb]$state == 1)
  {rate <- b*v}
  # if(v > 0)
  # {rate <- b}
  # else
  #   rate <- 0
  #
  return(rate)
}
```