Warsaw University of Technology
Faculty of Mathematics
and Information Science

BACHELOR'S THESIS
COMPUTER SCIENCE

# INFLUENTIAL NODE DISCOVERY IN SOCIAL NETWORKS

ZNAJDOWANIE WPŁYWOWEGO WIERZCHOŁKA W SIECIACH SPOŁECZNOŚCIOWYCH

Author:
Sajjad Hassanpour

Coauthor:
Azim Ahmadzadeh

Supervisor:
Dr. Tomasz Brengos

Warsaw, May 2016

............................................ ............................................

Supervisor's signature Author's signature

# Contents

# Abstract

This is a study on the idea of discovering the most influential nodes in social networks based on the *Authority Discovery* model proposed by a notable research published in 2011 [1]. The paper takes advantage of statistical and data mining techniques to find an efficient set of nodes that if provided with information bits, would maximizes the spread of information throughout a network. In our work, we implement their proposed method called *RankedReplace*, and deploy that in a desktop application called *MOST DESIRED* that we developed as an experimental tool. Using this application, one can visualize a desired network in a graphical panel, adjust the size of the requested set of nodes that they want to be discovered, run the algorithm on it and observe the illustration of the most influential nodes and analyze them.

The second main goal of this work is to evaluate the effectiveness of the `RankedReplace` method not in maximizing the spread but in protecting the network against the spread. To this end, we designed an experiment in our application that “vaccinates” a selected group of most influential nodes against an “infection”. We then compared the number of infected nodes in the network with a similar situation in which we vaccinated a different group, i.e. the greatest degrees nodes. We did so to decide whether or not protecting the set of the most influential nodes discovered by the approach in the literature could be a reliable selection in minimizing the damages done to a network.

The result of the experiment confidently indicates that the selection of nodes discovered by *RankedReplace* method should not be used in any tasks in which the protection of the network is desired.

**Keywords:** Data Mining - Network Mining - Virus Spread - Most Influential Nodes - Network Protection - Flow Authorities Model

# Abstract

Tematem pracy jest badanie problemu znajdowania najbardziej wpływowych wierzchołków w sieciach społecznościowych bazując na modelu zaproponowanym w 2011 [1]. Artykuł ten wykorzystuje techniki data mining do znalezienia zbioru wierzchołków, który maksymalizuje tzw. spread informacji w sieci. W naszej pracy implementujemy algorytm z [1], noszący nazwę RankedReplace, i testujemy go w stworzonej przez nas aplikacji "Most Desired". Nasza aplikacja wizualizuje zadaną na wejściu sieć, ilustruje zbiór najbardziej wpływowych wierzchołków i pozwala na ich analizę.

Drugim głównym celem pracy jest ocena efektywności działania algorytmu RankedReplace w przypadku próby ochorny sieci przed spreadem. Dokładniej, w naszej aplikacji zaprojektowaliśmy doświadczenie, w którym "szczepimy" wybrany zbiór wierzchołków przeciw "chorobie" i obliczamy liczbę wierzchołków "zarażonych". W pierwszym kroku wybranym zbiorem wierzchołków jest zbiór wierzchołków najbardziej wpływowych. W drugim kroku, zbiór wierzchołków o najwiekszych stopniach wyjścia. Porównując oba kroki eksperymentu stwierdzamy, że wybór wierzchołków metodą RankedReplace nie powinnien być używany w przypadku, gdy w grę wchodzi ochrona sieci.

**Keywords:** Data Mining, Network Mining, Rozprzestrzenianie Się Wirusa, Najbardziej Wpływowe Wierzchołki, Ochrona Sieci, Model Flow Authorities

# Chapter 1

# The Core Idea

In the following sections at first, we define all concepts and properties we build our thesis upon and then we elaborate on the Campaign Problem and different approaches in solving such a problem. At the end, we discuss the mathematical steps that results in the algorithms introduced by [1] and methods designed by us.

## 1.1. Basic Concepts and Definitions

In this section we define the concepts that our project draws upon. Some of these concepts are from the mathematical folklore, some others are introduced by us and the rest are borrowed from [1].

By a **network** we mean a directed, weighted graph $N(V, E, p)$[1] where $V$ is a set of the network nodes, $E$ is a set of ordered pairs $(i, j)$ of nodes $i, j \in V$, each of which indicates an edge of the network. That is, a pair $(i, j)$ indicates an edge starting from $i$ and ending in $j$ and this edge is different than $(j, i)$ that connects the same nodes but in the opposite direction. So, we need to distinguish edges with respect to their directions towards each node. Therefore, $(i, j)$ shall be called an **incoming edge** from the perspective of $j$, and at the same time, an **outgoing edge** from the perspective of $i$. Consequently, for a node $i$ , the number of its incoming edges is called the **indegree** of $i$ and the number of its outgoing edges is called the **outdegree** of $i$. In addition, the function $p : E \to [0, 1]$ associates probabilities to the network's edges $e \in E$. These assigned probabilities represent interactions between nodes such that the weight of each edge shows the probability of data being transmitted from one end of the edge to another, along with the direction of the edge. We denote such probability by $p_{ij}$ or $p(i, j)$ which is known as **transition probability**. The matrix containing all such probabilities is another known mathematical object as **transition matrix**[2] and it is denoted by $P = [p_{ij}]$.

We also use the term **information bits I** and we believe no strict definition for that is needed. By information bits we mean data being transmitted throughout, for instance, the Internet. They might form a text message in an IM application, a video file shared on Youtube, an image on Facebook, etc.

[1]We intentionally use the letter N instead of G, which is a conventional math notation for graph, to emphasize it being a network.

[2]The term *transition matrix* that we use in this document is a common math term, however, let's note that our source literature uses the term *transmission matrix* instead. We preferred the widely known concept.

Having talked about information bits, the concept that should immediately be discussed is the **spread** of information. Although "spread" is a fairly intuitive word, there are many different mathematical models describing it. We briefly talk about them in Section 1.3. Loosely speaking, the term "information spread" carries a measure for availability of information among the nodes of a network. If the spread of information over a network is higher in the situation A than in B, it means the information bits become more available for all the nodes in A than in B. In other words, greater spread in a network means the higher expected number of nodes having access to the information that was spread. In Section 2.1, we describe a method called `SteadyStateSpread` that computes the expected number of nodes which would receive the information bits, if a given set of nodes were initially provided with those bits. Having this measure, called the ***SSS*** value, the quality of any given set of nodes in terms of their spreading role in the network can be precisely determined.

Any node may be initially given information bits $I$. It is assumed that if a node contains some information bits, then all of its neighbors are automatically exposed to those bits. This concept is represented by the term **information exposure**. But let's note that $I$ will not necessarily be transmitted to the nodes which are exposed to it. Let's elaborate on it. Suppose nodes $i$ and $j$ are adjacent and $i$ has the information. Then, node $j$, as $i$'s neighbor, is exposed to the information and with the probability $p_{ij}$, it will receive the information. But this is not true in general, since we do not know the probability of node $i$ having the information in the first place. Given $\tau(i)$ be the probability of $i$ having the information, and $p_{ij}$ the transition probability from $i$ to $j$, the probability of $j$ receiving the information is $\tau(i) \times p_{ij}$ and we call this the probability of **information assimilation** (*cf.* Section 1.4). We just explained a very important remark that can be stated in the form below.

**Remark 1.1.** Information exposure should not be equated with information assimilation. That is, information exposure does not necessarily result in the transmission of those bits from the node that contains the bits to those which are exposed to the bits.

Now that we have introduced several concepts describing the flow of information over a network, it is time to raise this question that how a set of nodes should be selected to maximize the spread throughout the network, given that those nodes are initially provided with information bits. In other words, how can be discovered a set of nodes with greatest $SSS$ value. The set we want to discover is called a[3] *most influential* set[4]. This phrase is borrowed from [1] and is peripherally related to the concept of *hubs and authorities* ([3] [6]) in web networks. The word "influence" implies the power of a node or a collection of nodes in spreading the information. This is their influence on other nodes and ultimately over the entire network. In this study, we always denote this set by $\boldsymbol{S}$ and its size by $\boldsymbol{K}$ (*cf.* 1.2). We define this concept in the following way:

**Definition 1.1.** *A set $S$ is called a* ***most influential set*** *if there is no other sets of the same size with a greater $SSS$ value.*

In other words, it is said that a set $S$ of nodes is a most influential set if they collectively spread the information in a greater or equal scale than any other equally large collection of nodes.

[3]We intentionally use an indefinite article 'a' to indicate that such set is not necessarily unique.

[4]It is also called *Information Authorities*, but since the term *most influential* seems more intuitive, we mostly use this one instead.

A single node is called a **most influential node** (or simply **influential**) if it belongs to a most influential set of nodes. Therefore, unless we are interested in $S$ to be of size 1, we do not define a most influential node independent from a most influential set. (To avoid any confusion, see the remark below.)

**Remark 1.2.** It is critical to bear in mind that $S$ is not necessarily a collection of nodes with the $K$ highest $SSS$ values but it is very often a set of nodes that only collectively and as a whole reaches the highest $SSS$ value. This idea can be observed in 2.1.2.

**Remark 1.3.** We would like to add that the definition of a most influential set is a theoretical description of such sets. It illustrates the set we want to discover, however, the heuristic algorithm proposed in [1] (and in Chapter 2) never claims that what it finds is indeed a set with the greatest $SSS$ value but a strongly reliable one. The purpose of our thesis also is not to prove whether or not the discovered set is the best possible collection of node in spreading the information. Therefore, when we use the term "most influential set" as the result of the Flow Authorities algorithm, we mean the output of the algorithm.

Spread of information bits is not always desired to be maximized but in many cases it has to be constrained instead. For instance, the information we are talking about might be a computer virus spreading throughout a network of devices, or a contagious terminal disease which is rapidly spreading in a society. What all this type of examples have in common is the negative and disruptive effect of the information being spread. In our work, such information whose spread is not desired is called **Virus** and consequently instead of "information assimilation" we use the term **infection**. We say the node $i$ becomes **infected** when the virus has been transmitted from one of its neighbors to $i$. In Chapter 2, we explain the exact behavior of the virus we employ in our experiments, regarding its spread.

If we want to evaluate the quality of a most influential set, another collection of nodes, comparatively influential, should be available. Since the number of neighbors of a node also plays an important role in the influence of that node, the comparable selection of nodes that we want could be the $K$ greatest outdegrees nodes. The behavior of such a set in spreading the virus should be compared with the behavior of a most influential set. We call this set a **high-degrees set** and each of its members a **high-degree node** or simply **high-degree**.

In addition to the infected, high-degree and influential nodes that we have already defined, there are two more properties that we should introduce. We say a node is **vaccinated** if it does not become infected regardless of being exposed to the virus. That is, no matter how many of a vaccinated node's neighbors are infected, it does not get infected and consequently, it never causes the infection of any other nodes. In the experiment that we describe in Chapter 4, we observe the behavior of the virus spread when the influential or the high-degree nodes are vaccinated. It means, the set of the vaccinated nodes is always either the same as the most influential set or the high-degrees set. But to start this experiment we need to know how the infection should be initiated in the system. For this, we choose a set of nodes to which we initially inject the virus. We say that a collection of nodes are **initially infected** if their infection is artificially made and it was not the result of the virus spread.

So far, we have defined different sets of nodes. Here we shortly review their properties. The main set $V$ contains all nodes of the network. Two other sets are most influential and high-degrees sets. We denote the former by $M_{Inf}$ and the latter by $H_{Deg}$. Then:

1. $M_{Inf} \subseteq V$ and $H_{Deg} \subseteq V$,

2. $M_{Inf} \cap H_{Deg} \neq \emptyset$,
3. $\mid M_{Inf} \mid = \mid H_{Deg} \mid = K$.

Note that, (2) states that there are no restrictions that an influential node cannot be a high-degree at the same time or vice versa. And (3) must be held to keep the comparison valid and it says that the cardinality of both $M_{Inf}$ and $H_{Deg}$ must always be equal to $K$. In our study, we vaccinate each of these sets in a separate experiment. So, whenever we talk about vaccinated nodes, it is either $M_{Inf}$ or $H_{Deg}$ set that is vaccinated, however in general, we denote a vaccinated set by $Vac$.

We also defined the concept of infection, the set of infected nodes and the set of initially infected nodes. Let's denote the former set by $I_{Eve}$[5] and the latter by $I_{Ini}$. Then:

1. $I_{Ini} \subseteq I_{Eve}$,
2. $I_{Ini} = I_{Eve} \iff$ No spread takes place,
3. $Vac \cap I_{Eve} = \emptyset$,
4. $Vac \cap I_{Ini} = \emptyset$,
5. Either $Vac = M_{Inf}$ or $Vac = H_{Deg}$ .

## 1.2. The Campaign Problem

The campaign problem is mostly discussed when a graph describing a form of dynamic network[6] is to be analyzed. This network, in a medium scale, could be derived from interactions between members of a club, students of a university, football teams in an international championship, and in the large scale, could be seen in the social media (e.g: Facebook, Twitter), or the network of all devices connected to the Internet in a city, all costumers of a business and infinitely many other examples. To define this problem, we stick to the definitions of the keywords given in the previous section.

For a given integer $k$, the problem is to identify a set of $k$ nodes of the network as the most influential set. It is important to notice that we are not interested in a group of $k$ most influential nodes, but the nodes that collectively with other chosen nodes shape the most influential set. This concept can be intuitively defined as:

> "the problem of determining a set of targeted individuals whose positive opinion about an information item will maximize the overall positive opinion on that item in the network." [2]

This problem is also structured in our source literature which is the definition we base our work upon and comes in the following:

**Definition 1.2 ([1]).** ***(Campaign Problem)*** *Determine the set $S$ of $k$ data points at which the release of information bits $I$ would maximize the expected number of nodes over which $I$ is assimilated.*

[5]'Eve' stands for the word "Eventually" since those nodes are infected as the final result of the spread.
[6]A network whose structure as well as its weights are constantly changing.

## 1.3. Approaches

There are different approaches for finding the most influential nodes of a graph. The differences are mainly because of the differences in the information spread algorithms. They mostly follow either *Linear Threshold*[5] or *Independent Cascade*[4] models. Furthermore, some approaches may be interested in the procedure of the opinion formation and others may concentrate on the information adoption. In our work, however, we are following a different study which employs a third model, i.e. the *Flow Authorities* model, and we target the procedure of the information adoption.

## 1.4. The Math Behind The Code

The purpose of this section is to elaborate on the steps that are required for the design of Flow Authorities algorithm (see Chapter 2) which is originally introduced in [1] and this present work aims at evaluating its effectiveness.

There are two methods shaping this algorithm: `SteadyStateSpread` (Chapter 2, method 1) and `RankedReplace` (Chapter 2, method 2). The former is used to determine the expected information spread for a given starting set of nodes and the latter is used to find the set which results in the maximum impact on the entire graph. The `RankedReplace` method calls `SteadyStateSpread` iteratively until it finds the best possible set of nodes. So, it is safe to say that the heart of Flow Authorities algorithm is the formula used in the most inner method, `SteadyStateSpread`.

We assume a network, as a directed-weighted graph, to contain the transmission probability of its edges. Denoting the transmission probability from node $i$ to $j$ by $p_{ij}$ and the probability for the node $i$ to have the information $I$ by $\tau(i)$, we can say that the probability for $i$ to eventually transmit $I$ to $j$ is equal to $\tau(i) \times p_{ij}$. But $\tau(i)$ is only known for those nodes that initially possess the information and we do not know how this variable changes during the spread. In other words, we need to know how the chances are for a node to obtain the information $I$ and meanwhile, compute the information assimilation value for each node. To explain how we find such data, we discuss one simple example from [9](page 229).

**Example 1.1.** Bob, Alice and Carol are playing Frisbee. Bob always throws to Alice and Alice always throws to Carol. Carol throws to Bob 2/3 of the times and to Alice 1/3 of the times. In the long run, what percentage of the time will each of the players have the Frisbee?

This example can be summarized in the matrix:

$$P = \begin{array}{c|ccc} & A & B & C \\ \hline A & 0 & 0 & 1 \\ B & 1 & 0 & 0 \\ C & 1/3 & 2/3 & 1 \end{array} \tag{1.1}$$

and the percentage referring to the number of times each player will have the Frisbee is in the solution of the equation below:

$$\lim_{n\to\infty} P^n = \begin{bmatrix} x & y & z \\ x & y & z \\ x & y & z \end{bmatrix} \tag{1.2}$$

where $x$ shows the percentage that Alice has the Frisbee and so does $y$ and $z$ for Bob abd Carol, respectively. To solve this equation for $x$, $y$ and $z$, we need to solve a system of three equations or equivalently:

$$\begin{bmatrix} x & y & z \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1/3 & 2/3 & 0 \end{bmatrix} = \begin{bmatrix} x & y & z \end{bmatrix}. \tag{1.3}$$

There is one extra condition that must be taken into consideration: $x + y + z = 1$, simply because they all represent probabilities. Now we are able to solve this system and the solutions are $x = 3/8$, $y = 1/4$ and $z = 3/8$. That is, 3/8 of all time, Alice had the Frisbee, 1/4 of time Bob and finally 3/8 Carol.

Having this example explained, we may picture nodes of the network as the players and the edges as a concept that carries the transition probability and the direction of transmission. Solving a similar system of equations should gives us the probability $\tau(i)$ that each node $i$ contains the information $I$, but expressing the probability of spread at any particular node of a network cannot be expressed in a linear form (*cf.* Equation.1.4). therefore, solving such a system of equations for a network is actually a difficult problem. That is why a stochastic model is chosen.

Let $\pi(i)$ be the steady-state probability of node $i$ assimilating the information. If we want node $i$ to assimilate the information, it should be exposed to the information, and moreover, the transmission should also take place. This argument has another side, that is, in order for this node not to assimilate the information (that has the probability equal to $1 - \pi(i)$), it must not receive the transmission from any of its neighbors. In other words, for each of $i$'s adjacent nodes ($l$), assimilation ($\pi(l)$) and transmission ($p_{li}$) should not take place together. So we will have:

$$1 - \pi(i) = \prod_{l \in N(i)} (1 - \pi(l) \cdot p_{li}). \tag{1.4}$$

We should also consider that for each of the $k$ nodes in $S$ (see 1.2) at which the information is released, we set the corresponding value of $\pi(\cdot)$ to 1, because assimilation is trivial for them. The following system of equations is the conclusion of what we explained so far:

$$\begin{cases} 1 - \pi(i) = \prod_{l \in N(i)} (1 - \pi(l) \cdot p_{li}) \\ \pi(i) = 1 \qquad \forall i \in S. \end{cases} \tag{1.5}$$

As we already mentioned, `SteadyStateSpread` method is designed to solve this system of equations in a stochastic fashion. For details see Chapter 2.

# Chapter 2

# Algorithms and Implementation

In this Chapter we discuss two algorithms: *Flow Authorities* and *Virus Spread*. The former is proposed in [1] and the latter is our design as an experimental tool for evaluation of effectiveness of Flow Authorities algorithm. The mathematical background of the Flow Authorities algorithm is already discussed in Chapter 2. Here we focus on the design of the algorithm instead.

## 2.1. Discovery: Flow Authorities Algorithm

The algorithm is employed by our application for finding the most influential nodes. As it is introduced by its authors, the algorithm suggests a stochastic approach in order to model the flow behavior in social networks. There are two methods to be implemented: `SteadyStateSpread` (method 1) and `RankedReplace` (method 2). The former is used to determine the expected information spread for a given starting set of nodes and the latter is used to find the set which results in the maximum impact on the entire graph. The `RankedReplace` method corresponds to the broad approach of ranking the nodes and iterative replacement based on the steady state flow impact.

This approach heuristically promises to maximize the amount of the total flow of information in the network, instead of maximizing the expected number of nodes exposed to the information.

### 2.1.1. `SteadyStateSpread` Method

In Section 1.4 (Equation.1.4) we explained how the following formula is derived:

$$\begin{cases} 1-\pi(i) = \prod_{l\in N(i)}(1-\pi(l)\cdot p_{li}) \\ \pi(i) = 1 \qquad \forall i \in S. \end{cases}$$

An algorithm that solves this system of equation is implemented in method 1. Note that we use the letter $q$ instead of the Greek letter $\pi$. This method takes two arguments: the set $S$ of nodes which initially possess the information and the transition matrix. At first, it initializes the assimilation status of each node with 1's for all nodes in $S$ and 0's for others. This initialization is in fact the implementation of the second equation in 2.1.1. Then, in an iterative approach, based on the current values of information assimilation of each node ($q^t(\cdot)$) and the first equation in 2.1.1, it computes the new values ($q^{t+1}(\cdot)$).

**Method 1** SteadyStateSpread(InitialSet: S, TransmissionMatrix: P)

```
1: procedure SteadyStateSpread
2:     for all $i \in S$ do $q^0(i) \leftarrow 1$;
3:     for all $i \notin S$ do $q^0(i) \leftarrow 0$;
4:     $t \leftarrow 0$;
5:     repeat
6:         for all $i \in S$ do $q^{t+1}(i) \leftarrow 1$;
7:         for all $i \notin S$ do
8:             $q^{t+1}(i) = 1 - \prod_{l \in N(i)}(1 - p_{li} \times q^t(l))$;
9:         $C_{t+1} = \sum_{i \notin S} |q^{t+1}(i) - q^t(i)|$;
10:        $t \leftarrow t + 1$;
11:    until $C_t < 0.01 \times C_1$;
12:    return $(\sum_{i \notin S} q^t(i))$;
```

The stop condition for this loop is when the total differences between the old and the new assimilation values for all nodes in $U-S$ (i.e. $\sum_{i \notin S}|q^{t+1}(i)-q^t(i)|$) gets negligible in comparison with the first time that the total differences were computed. The proposed negligibility is determined by the fraction 0.01 in $C_t < 0.01 \times C_1$. The fact that what fraction should be considered as negligible depends on the level of confidence we expect from this method and of course the smaller the value, the more time consuming the algorithm becomes.

Perhaps the most important aspect of this method is the number it returns. The purpose of this algorithm is to associate a meaningful number to each input set $S$ that indicates the influence of $S$ over the entire network. The value computed as $(\sum_{i \notin S} q^t(i))$ is the expected steady state number of nodes which assimilate the information and we call this value as the ***SSS*** value of the set $S$. Note that in this sum nodes are chosen from $U-S$. Therefore, the $SSS$ value does not contain the number of nodes which were initially given the information.

### 2.1.2. `RankedReplace` Method

`RankedReplace` employs the `SteadyStateSpread` method to determine the $k$ nodes that collectively obtain the maximum influence throughout the network. However, it starts the procedure by finding $k$ individual nodes whose $SSS$ values are higher than others, with no focus on their cooperation. This set of nodes are considered to be the initial candidates of the flow authorities denoted by $S$. Then, in an iterative fashion, it tried to improve $S$ by replacing the best nodes (with respect to their $SSS$ values) in $U - S$ with the worst candidates in $S$ and checking the $SSS$ value of $S$ as a whole (not individually). This iteration is to check the existence of any better combination of nodes that should finally ultimately represent the flow authorities. At the end, this method returns the set of $k$ nodes that together maximize the flow of information over the entire graph.

**Method 2** RankedReplace(TransmissionMatrix: P, NumberOfAuthorities: k)

```
1: procedure RankedReplace
2:     for all $i \in U$ do
3:         $SteadyStateSpread(\{i\}, P)$;
4:     $S \leftarrow$ Initial set of k authority nodes with the highest value of $SteadyStateSpread(\cdot)$;
5:     Sort nodes in $U - S$ in descending order of $SteadyStateSpread(\cdot)$ ;
6:     for all $i \in U - S$ do
7:         Sort the list $S$ in ascending order of $SteadyStateSpread(j, P)$;
8:         Pick the first element (if it exists) of sorted list $S$ which is such that replacing $i$
   with it increases value of $SteadyStateSpread(S, P)$,
9:         if no replacement has occurred in the last $r$ consecutive iterations, then return $S$;
10:    return $S$;
```

## 2.2. Evaluation: Virus Spread Algorithm

There are different virus spread models that may take a variety of properties into account and are proposed and used for different purposes. Many of the properties they require are extra information which do not play a role in our study (e.g. the immunization and resistance factor of each node or the change in power of the virus in time). Since we are studying the generality of the flow authorities model and not any of its particular applications, we introduce a simple virus model with minimal determining factors.

Our virus spread algorithm is designed to spread the virus that is initially injected to a carefully chosen set of nodes ($I_{Ini}$) throughout the network, relying only on the weight of the network's edges and its topology.

This algorithm contains a method (`FindNewInfectedNodes`) that discovers the neighboring nodes of an infected one, that are consequently infected. This method is called repeatedly in the main method (`Spread`) which is responsible to discover the infection spread among all nodes of the network. (For the details of the experiment, see Chapter 4).

### 2.2.1. `FindNewInfectedNodes` Method

This method is responsible for computing and recording new changes in the graph that a newly infected node may cause. In other words, every time a new node becomes infected, this method should be called with the infected node as its input argument, to decide which of its neighbors would be infected as a result. To determine whether the virus transmits from one node ($i$) to another ($j$), we take the weight of the incident edge ($p_{ij}$) as a threshold. If a uniformly generated floating point number from $[0, 1]$ at the given time lies in the interval $[0, p_{ij}]$ then we assume that the virus transmits. Otherwise, the neighbor remains uninfected for this event. This assumption relies upon an intuitive virus spread behavior that the smaller the weights, the harder the virus transmits. That is, two nodes with a small transition probability rarely infect each other, however, this is not impossible. In addition, note that for each neighbor that is either vaccinated or infected, no changes take place in this method.

When this method discovers which neighbors of the given node become infected, it returns their IDs and halts.

**Method 3** FindNewInfectedNodes(InfectedNode: x)

```
procedure FindNewInfectedNodes
    List⟨Nodes⟩ newIsInfectedNodes;
    for all Neighbors n of x do
        if (n.isVaccinated() ‖ n.isInfected()) then
            //do nothing;
        else
            w ← n.getWeight(); //Weight of the edge from x to n
            randW ← generate a random weight from $[0,1]$;
            if (randW $\leqslant$ w) then
                n.setIsInfected(True);
                newInfectedNodes ← n;
    return newInfectedNodes;
```

### 2.2.2. `Spread` Method

Spread method works very simply but expeditiously. It takes the nodes from $I_{Ini}$ (initially infected) and iterates over them. It keeps them in a queue of infected nodes (`InfectedQueue`), calls them one by one, finds the infected neighbors for each of them and pushes them into the queue. The main responsibility of this method is to count the number of nodes that will ultimately become infected.

**Method 4** Spread(Initially Infected: S)

```
procedure Spread
    Queue InfectedQueue ← S;
    while (! infectedQueue.isEmpty()) do
        node ← infectedQueue.dequeue(); //Pop the first element
        newInfectedNodes ← findNewInfectedNodes(node);
        if (! newInfectedNodes.isEmpty()) then
            numberOfInfectedNodes += newInfectedNodes.size();
            infectedQueue.enqueueAll(newInfectedNodes);
```

It is crucially important to make sure that this method ever halts. This is true in this method because neither of the infected nodes nor the vaccinated ones will ever be put back more than once into the `infectedQueue`. We know this because the if-statement in method 3 (line 4) guarantees it. So `FindNewInfectedNodes` never returns any nodes twice and thus we can confidently say that the queue accepts each node at most only once. We also know that there is only a finite number of nodes in our graph. So, the computation halts within a finite number of times.

# Chapter 3

# Data Structures and Visualization

## 3.1. Efficiency Matters

Since we aim at working on real data sets, our implementation has to be ready for big data sets. For instance, the algorithm should face no problem in dealing with a graph of one million nodes and millions of edges. A naive data structure design would lead to *heap space error* or, at best, too lengthy computations. So the efficiency in defining data structures inside the algorithm, as well as the structure of the input graph, should be considered as a determining factor during our implementation.

## 3.2. Experiments For Efficient Data Structures

There are several different data structures to be employed for graph representation, each of which are suitable for some specific goals. In our work we tried a few of them to choose the right model that served our requirements best. For instance, we allowed the input file to be as big as it could get and instead, we tried to find a quick way for reading its content: We implemented codes to use the Java class called `RandomAccessFile` and to map a page of our large input file (instead of the entire file) to the memory, every time we need to read or modify the graph. In this experiment, the input file could be as big as the hardware allowed it to be, however, due to the memory heap limitation, we did not succeed in reading and modifying a 1 million $\times$ 1 million matrix of `doubles`. In another attempt, we realized that the `Hashmap` structure of the Java program also did not fulfill our requirements, because of the *hash collision* which occurs on very large data sets. We also considered $2D$ arrays but because of the sparseness of the expected matrix, we moved on.

It took us a few other experiments to realize some important facts about what we needed which appear in the following:

- As we mentioned earlier, matrices representing the graphs of real networks are, in many cases, sparse matrices. Therefore, reserving memory for all possible edges many of which are not present, as we do in adjacency matrix representation, does not seem to be a good idea.

- It is important to note that we are working on directed graphs. However, in the entire algorithm regarding finding the most influential nodes, we are interested only in incoming edges (defined in Chapter 1). Therefore, there is no need to keep both endings of every edge in this algorithm.

Finally, we came up with a new representation of the graph in which we did not lose any pieces of information, yet we performed our computations over minimal amounts of information. The details are explained in the next section.

## 3.3. Designed Data Structures

In this work, our goals are to implement the algorithm, design efficient data structures and test the effectiveness of the algorithm. Therefore, we do not put any effort into employing databases and integrating them with our application. We use simple text files as our inputs, since they are easy to be generated from any real database describing any networks. And moreover, once the application read them into memory, they are no longer needed to be modified. Needless to mention that in the real situation, when an analysis of real-time stream of big data sets is required, a very quick access to the database, which is being modified extensively within a fraction of a second, appears to be a bottleneck for achieving any valuable results and it cannot be ignored. However, we focused solely on investigation to find a solution for campaign problems and evaluating the results.

Figure 3.1: Simple Graph

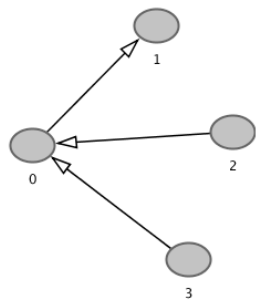


Table 3.1: Input File Pattern

| ID:From | ID:To | Weight |
|---|---|---|
| 0 | 1 | 0.5 |
| 2 | 0 | 0.5 |
| 3 | 0 | 0.5 |

### 3.3.1. Reading Data

Regarding the design of an input file, we expect them to follow a simple pattern that can be seen in table 3.1 which represents the graph illustrated in figure 3.1, with the assumption that all weights are equal to 0.5 for simplicity. In the text file, no header is needed and any number of spaces or tabs are accepted as delimiters.

**Remark 3.1.** It is important to note that there might be many nodes without any adjacent neighbors in a given graph. Therefore, we do not need to keep all of them in the input file. However, we should bear in mind that the following assumptions are made:

- The IDs of nodes are `integer` numbers: First and second columns.
- Weights are assumed to be of type `double`.
- No characters, apart from `integer`s and `double`s, should be present in the input file.
- The IDs of nodes start from 0, regardless of whether they are explicitly given in the input file or not.
- The IDs of nodes are consecutive numbers. That is, all numbers from 0 to the maximum IDs are considered to represent nodes of the graph, even though they might not be explicitly mentioned in the input file.

- The order of lines of the input file does not affect the graph's topology nor the computation.

### 3.3.2. Primary Data Structures

**Remark 3.2.** Since we are working on directed graphs, we have two types of edges from a node's perspective: incoming edges and outgoing edges, which are self-explanatory concepts, but they are also defined in Chapter 1.

Now, we introduce a few objects that will be used in the implementation and their details should be discussed for further explanations of the main idea.

**Structure 3.1.** An object of type **NodeAndWeight** contains the ID of a node and the weight of an *incoming* or outgoing edge incident to that node. Such definition is relative and it is important to determine that from which node's perspective this object is being considered.

**Structure 3.2.** A list of objects of type `NodeAndWeight` corresponding to each node that represents neighbors of the node with an incoming edge, is called **neighborsList_In**.

**Structure 3.3.** A list of $N$ `neighborsList_In`s describes the entire graph with respect to its incoming edges. $N$ is the size of the graph or simply the number of its nodes. This list is called **nodesList_In**.

**Remark 3.3.** We may define **neighborsList_Out** and **nodesList_Out** in the same way by replacing the word incoming edge with outgoing edge in the definitions 3.2 and 3.3 respectively.

Finally, to see how these integral objects shape the desired structure, a quick example might help.

**Example 3.1.** For a simple graph shown in figure 3.1, we store the graph in the following fashion: (Without loss of generality, we still assume the weights of all edges to be 0.5.)

- `nodesList_In`: $[[(2, 0.5), (3.0.5)], [(1, 0.5)], [\ ], [\ ]]$, which comprises of:
  - `neighborsList_In[0]` : $[(2, 0.5), (3.0.5)]$
  - `neighborsList_In[1]` : $[(1, 0.5)]$
  - `neighborsList_In[2]` : $[\ ]$
  - `neighborsList_In[3]` : $[\ ]$

**Remark 3.4.** In the implementation, we defined the two main containers (i.e. `nodesList_In` and `nodesList_Out`) to be of type `ArrayList` because:

1. their sizes need not to change and they are both equal to the size of the given graph,

2. we need to access their elements very frequently,
3. during the entire computation, there is no need to add/remove elements to/from these lists,

and `ArrayList`s are indeed the best choice to fulfill such requirements.

**Remark 3.5.** Unlike the main containers, the inner containers (i.e. `neighborsList_In` and `neighborsList_Out`) are defined as `LinkedList`s. This structure is chosen because:

1. their sizes cannot be set before the entire graph is read,
2. every time we need to access their elements, we need to iterate on them entirely, so no random access is required,
3. while the input file is being read and the graph is being stored in these containers, constant insertion of new data is required,

and `LinkedList`s are certainly the best choice to fulfill these expectations.

### 3.3.3. Visualization Data Structures

In addition to the objects we already introduced, we need more information about features of every single node. These pieces of information are used for both visualization of the graph and the virus spread experiment.

**Structure 3.4.** An object of type **SVertex** is needed to be constructed for all nodes in order to illustrate them in the graphical panel. This object contains graphical information like:

- `ID`, which lets us synchronize a list of such objects with any other list of nodes.
- `Coordinate`, to position nodes on a panel and avoid collision.
- `Radius`, to help showing how dense a node is, with respect to the number of incident *incoming edges.*
- `Neighbors`, to keep track of all neighbors of each node with an incoming edge toward that node.

and some information for the virus spread experiment:

- `IsInK`, to determine if this node is one of the $k$ most influential nodes for which one of the main methods is responsible to discover.
- `IsInMax`, to determine if this node is one of the high-degree nodes.
- `IsInfected`, to determine if this node has become infected due to the virus spread.
- `IsVaccinated`, to determine if this node has been vaccinated prior to the virus spread.

## 3.4. Employed Sorting Method

In the next Chapter, we discuss the algorithms used in this application. One of them that is named `RankedReplace` method (see method:2 in Chapter 2) calls a sorting method frequently. To have an efficient implementation of this method, the sorting method must be chosen carefully to minimize the time and space complexity. It is not hard to see that the collection $S$, which should be sorted over and over, after it is once put in order, does not change significantly. To be more precise, in every iteration of the method, only one element of $S$ might be misplaced. Therefore, we decided to use the *insertion* sort which is adaptive, i.e. "efficient for data sets that are already substantially sorted" [7]. For more information about the corresponding Java class, look at 6.3.3.

## 3.5. Visualization

### 3.5.1. Main Window

The main window of this application (`MainFrame`) is a Java `JFrame` and runs the main Swing thread. In this frame, the user can set his or her requirements (figure 3.2). At first, users should provide the input file that describes the network (about the design of the input file, see 3.3.1). Since the input file could be very large and therefore time consuming for the application to load, a separated thread (using Java `SwingWorker`) will be created to handle this process.

When the input file is successfully loaded, the user should enter a value for $K$ and then by pressing the `FIND MOST-DESIRED` button, the ID of the $K$ most influential nodes and the $K$ high-degree nodes, and the $SSS$ value for each discovered set will be displayed on the right panel (figure 3.3). At this point, it is possible to proceed experiments in a graphical mode or in a text mode. The graphical mode directs the user to the Graph Visualization window, and the text mode prints out the results on the application's console.

Figure 3.2: Screen shot of the menu panel in the main window

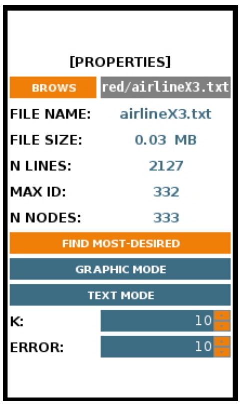
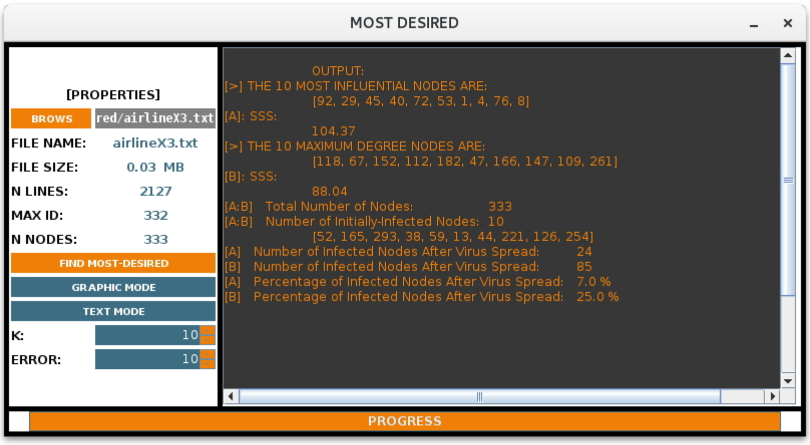

[PROPERTIES]
BROWS red/airlineX3.txt
FILE NAME: airlineX3.txt
FILE SIZE: 0.03 MB
N LINES: 2127
MAX ID: 332
N NODES: 333
FIND MOST-DESIRED
GRAPHIC MODE
TEXT MODE
K: 10
ERROR: 10

Figure 3.3: screen-shot of the main frame with the results in the text mode


MOST DESIRED

[PROPERTIES]
BROWS red/airlineX3.txt
FILE NAME: airlineX3.txt
FILE SIZE: 0.03 MB
N LINES: 2127
MAX ID: 332
N NODES: 333
FIND MOST-DESIRED
GRAPHIC MODE
TEXT MODE
K: 10
ERROR: 10

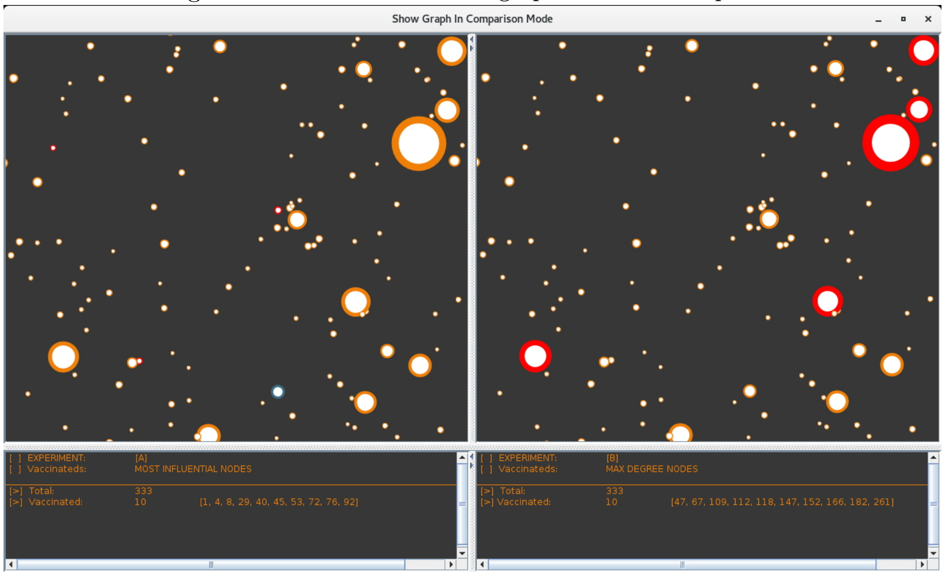
```
              OUTPUT:
[>] THE 10 MOST INFLUENTIAL NODES ARE:
              [92, 29, 45, 40, 72, 53, 1, 4, 76, 8]
[A]: SSS:
              104.37
[>] THE 10 MAXIMUM DEGREE NODES ARE:
              [118, 67, 152, 112, 182, 47, 166, 147, 109, 261]
[B]: SSS:
              88.04
[A:B]   Total Number of Nodes:               333
[A:B]   Number of Initially-Infected Nodes:  10
              [52, 165, 293, 38, 59, 13, 44, 221, 126, 254]
[A]   Number of Infected Nodes After Virus Spread:        24
[B]   Number of Infected Nodes After Virus Spread:        85
[A]   Percentage of Infected Nodes After Virus Spread:   7.0 %
[B]   Percentage of Infected Nodes After Virus Spread:   25.0 %
```

PROGRESS

### 3.5.2. Graph Visualization

This is a separate window (Java `Jframe`) which contains 4 Java `JPanels`. As figure 3.4 shows, two larger panels illustrate two copies of a graph provided by the user and the two smaller panels presents useful information about the graph and the experiment's result.

Figure 3.4: Screen-shot of the graph frame with 4 panels



```
[ ]  EXPERIMENT:        [A]
[ ]  Vaccinateds:       MOST INFLUENTIAL NODES

[>]  Total:             333
[>]  Vaccinated:        10          [1, 4, 8, 29, 40, 45, 53, 72, 76, 92]
```

```
[ ]  EXPERIMENT:        [B]
[ ]  Vaccinateds:       MAX DEGREE NODES

[>]  Total:             333
[>] Vaccinated:         10          [47, 67, 109, 112, 118, 147, 152, 166, 182, 261]
```

In the left panel, the most influential nodes are vaccinated and distinguished with vertices highlighted with bold red borders. One should see this panel in comparison with the one on the right where the vertices with bold red borders represent the high-degree.

Vertices on these panels are Java `Ellipse2D` objects representing the nodes of the given graph. In order to have a clear illustration, we decided to hide all edges and instead use ellipses' sizes to show how popular each vertex is. Note that the radii of vertices are calculated proportional to their outdegrees.

In order to maximize the efficiency of the application in the graph visualization component, instead of using pre-defined Java objects we decided to draw primitive shapes on the panel and keep their properties in a supporting class called `SVertex`. Using this class we allow the user to click on any ellipse and see the ID of any vertex and highlight all nodes to which this vertex is connected to with an outgoing edge. The colors used in the graph visualization panels are applied to let the user achieve a sufficient level of understanding of the topology of the input graph without relying on its edges. These colors are categorized in the table 3.2.

Table 3.2: Vertices' color coding in graph visualization

| N | Role | Sample | Color |
|---|---|---|---|
| 1 | Default | 🟠 | Orange |
| 2 | Vaccinated | 🔴 | Red |
| 3 | Neighbors | 🟢 | Green |
| 4 | Hovered | 🔵 | Blue |
| 5 | Infected | 🟡 | Yellow |
| 6 | Clicked | ⚫ | Black |

A collection of all vertices are supposed to represent the input graph. This collection is stored in a Java `JPanel` class. This class, supported by `VertexMouseEvent` class and an observer pattern, is fully responsible for the visualization of an interactive graph.

The graph visualization is designed to:

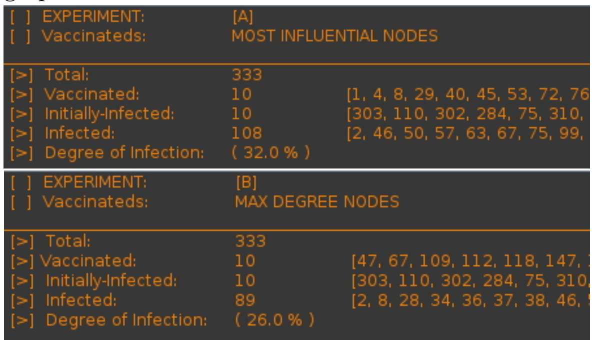
- show the $K$ vertices discovered by the `RankedReplace` method, known as the most influential nodes. These vertices are highlighted with red borders on the left-side panel.
- show the $K$ high-degree nodes. These vertices are highlighted with red borders on the right-side panel.

- show the neighbors of each vertex when clicked. All neighbors will be highlighted with green borders.
- show the ID of each vertex when clicked.
- show all the infected nodes after the spread is finished. Infected nodes will be highlighted with yellow borders.

The side by side design of this window (figure 3.4) is to provide the user with an environment which makes the comparison of the two networks possible. Since these two networks only differ in their colorings, it is easy for the user to compare the result of the virus spread algorithm on different areas of the networks. For instance, the user may observe in many situations that a node with a great degree (i.e comparatively large ellipse) has a red border on the right-side panel, but an orange border on the left-side one. This should be interpreted as that node of the network, despite its great degree, is not a significantly influential node.

Below each side-panel, a console panel is provided to show the important information. Figure 3.5 illustrates two consoles for both side-panels. As one can see, the total number of nodes, the number of vaccinated nodes as well as their IDs, the number of *initially infected* nodes and their IDs, the number of *infected* nodes and their IDs, and eventually the *infection degree* in percentage, are computed and shown on these consoles.

Figure 3.5: Screen shot of the bottom console in the graph visualization window



```
[ ]  EXPERIMENT:            [A]
[ ]  Vaccinateds:           MOST INFLUENTIAL NODES

[>]  Total:                 333
[>]  Vaccinated:            10              [1, 4, 8, 29, 40, 45, 53, 72, 76
[>]  Initially-Infected:    10              [303, 110, 302, 284, 75, 310,
[>]  Infected:              108             [2, 46, 50, 57, 63, 67, 75, 99,
[>]  Degree of Infection:   ( 32.0 % )
```

```
[ ]  EXPERIMENT:            [B]
[ ]  Vaccinateds:           MAX DEGREE NODES

[>]  Total:                 333
[>] Vaccinated:             10              [47, 67, 109, 112, 118, 147,
[>]  Initially-Infected:    10              [303, 110, 302, 284, 75, 310,
[>]  Infected:              89              [2, 8, 28, 34, 36, 37, 38, 46,
[>]  Degree of Infection:   ( 26.0 % )
```

# Chapter 4

# The Experiment and Results

One of the major goals of this project, as we discussed several times in this document, is to evaluate the effectiveness of the `RankedReplace` method, that is how selective the most influential nodes are and the possibility that this set is not necessarily the "most influential" when it comes to protecting a network. This evaluating experiment can be designed in many different ways. We try to discuss some of them here so that we may hopefully convince the reader why we have chosen the virus spread experimental model.

## 4.1. Different Approaches

### 4.1.1. Brute-force Approach [Not Applicable]

It is practically impossible to design an experiment which tries to find a better solution in a brute-force manner so that based on its success or failure, the effectiveness of the `RankedReplace` method could be determined. Let's look at a simple example: Imagine that a small network of only 1000 nodes is given and the user is interested in discovering the most influential set of size 10. In an exhaustive attempt, one should gather $\binom{1000}{10}$ sets and run the `SteadyStateSpread` method for each of them, therefore it should be called nearly $2.63 \times 10^{23}$ times! Multiply this number by the time complexity of the method and you will see how impractical it would be.

### 4.1.2. Semi Brute-force Approach [Not The Best]

We may try to find numerous sets as the candidate sets, each of which are of the same size as the claimed most influential set (i.e. $k$). These candidate sets should be chosen such that they are potentially significantly influential sets and possibly even more influential than the one discovered by the `RankedReplace` method. Then, simply by calling the `SteadyStateSpread` method for each of those sets, we could find out whether the discovered set is indeed the most influential set.

This experiment is valid, since the `RankedReplace` method never claimed to guarantee that the discovered set is the best set in terms of the influence. In fact, after a certain number of iterations during which no changes in the members of the discovered set occur, the algorithm halts and returns the current set as the most influential one. Therefore, there is a chance, however small, that there exists another combination of nodes which forms a better candidate set and it is not found.

Although there is a chance that this approach could find a better candidate, if we do not find guidelines in finding the potentially influential sets, then the number of sets to be generated in order to increase this chance to a reasonable level is too high and in reality, only coincidentally it may happen that a better combination exists and we manage to find it.

Unfortunately, this experiment does not seem to be sufficient, given the fact that if it does not manage to find a better set, since it is impossible to test all possible candidates, it does not imply that no better sets exist and therefore it does not result in any meaningful conclusion. On the other hand, even if we manage to find a more influential set, the experiment does not determine the effectiveness of the algorithm, but only disproves that the outcome is always the best, which was not claimed by the algorithm in the first place.

### 4.1.3. Virus Spread Approach [Our Choice]

In this approach, instead of using the `SteadyStateSpread` method which was itself used in the discovery of the most influential nodes to evaluate the discovery method, we picture the scene in the exact opposite way: If providing information bits to the most influential nodes of a network results in the maximum spread, their absence should deprive many nodes of the information. To have the ability to measure and compare the proportion of deprived nodes, we realized that we need a copy of the network so that two experiments can take place in parallel. In the following section the detailed scenario of this experiment is explained.

## 4.2. Virus Spread Experiment

### 4.2.1. The Scenario

At first, we clone the given network so that we are able to run two experiments, A and B, in parallel. In the application, this is shown in a two sided panel, each for one experiment. In both societies we need a set of influential nodes to be discovered and marked. The size of this set, $k$, should be adjustable by the user. In A, we run the `RankedReplace` method to find the $k$ nodes which are claimed to be the most influential ones, while in B, we find the $k$ high-degree nodes and mark them as well. It is worth repeating that, since we are working on directed graphs, by degree of a node we mean the outdegree. At this point, the only difference between two networks is the set of marked nodes which may or may not have an intersection. We vaccinate those marked nodes so that they will not be able to pass the virus (information bits) to their neighbors if they are given at all. Next, we need a set of randomly chosen nodes of an arbitrary size which can play the role of *initially infected* nodes or the nodes to which the information bits are given from the outside of the society. This set should be the same in both networks A and B.

**Remark 4.1.** The set of initially infected nodes must not contain any nodes which are already marked as either a most influential or a high-degree node.

Now that we have a network in which only the claimed most influential nodes are vaccinated and a similar network in which only the high-degree nodes are vaccinated, and the *initially* *infected* nodes are already chosen, we should run the *virus spread* algorithm on both A and B, and observe the *infection degree* in each network in the absence of the two different groups of nodes.

### 4.2.2. The Logic of The Experiment

Vaccination of any sets of nodes (influential or not) in a network would doubtlessly result in a lower (or equal) degree of spread, and it is safe to say that the more influential the vaccinated nodes, the less widespread the distribution would become. Thus, if the claimed most influential nodes discovered by the `RankedReplace` method, are indeed the most influentials, their vaccination, in comparison with the vaccination of any other set of nodes, must have a significant impact on the spread of the virus throughout the society.

To understand the statistics coming from the experiment on the vaccination of the claimed most influential nodes and their impact, we need to derive similar statistics in an analogous situation so that we have data to compare our results with. That is why we find the $k$ high-degree nodes as well. These nodes shall also be considered as a set of influential nodes due to the numerous neighbors that each of them are in touch with. Vaccination of such a set should also have a significant negative impact on the spread of the information bits. And, in the long run, comparing the results of these two sets being vaccinated, one in the experiment A and another in B, should let us determine how precisely the most influential nodes are discovered and how effective the *RankedReplace* method is.

**Remark 4.2.** We are aware of the fact that the high-degrees set may seem a naive candidate to be highly influential. In fact, it is a naive candidate, because we are only considering the degree of the nodes and not the weights of their outgoing edges, and we know that the product of those two factors could give us a much better set. Although, taking those weights into account would give us a more reliable set, we believe that the most influential set discovered by `RankedReplace` algorithm cannot even compete with such a naive set, when it comes to protecting the network.

There is an important point in this scenario that should be taken into consideration. As it was already discussed in 4.2.1, a randomly chosen set of nodes is needed to be initially infected. This set should be the same in both experiment A and B, otherwise the results of the two experiments are not comparable. And based on the remark 4.1, this set is free of any vaccinated nodes (most influential or high-degree) and as a result, for instance, in experiment B, although only the high-degree nodes are vaccinated, the virus spread cannot benefit from starting the spread from one of the most influential nodes, since they are vaccinated in A. And for the same reason, starting from one of the $k$ high-degree nodes is also not allowed in A. This constraint puts the evaluation of the `RankedReplace` method in a more restricted situation, because the method cannot count on a good start by targeting the influential nodes to be initially infected. However, in each experiment, one set is vaccinated and the other one is free to participate in the spread of the virus but after the initial step took place. Therefore, in the long run, the claimed most influential nodes have a fair chance to prove their impact by both their absence in A and their presence in B.

### 4.2.3. The Expected Results

After sufficiently repeating this experiment with different $k$'s, different selections of *initially infected* nodes and different numbers of *initially infected* nodes, we expect to observe a more significant impact in the experiment A than in B. So, the degree of spread in the network where the claimed most influential nodes are vaccinated is expected to drop to a lower level than what the experiment B illustrates.

You can find the details and results of these experiments in the Appendix A.

## 4.3. Results and Conclusions

### 4.3.1. The Algorithm's Behavior

First, we demonstrate how the algorithm finds the sets of the most influential nodes and compare how the nodes are added and removed in the two discovered sets.

In figure 4.1, the black curve represents the $SSS$ values (`SteadyStateSpread`) for the $K$ most influential nodes discovered by the `RankedReplace` method and the gray curve illustrates the $SSS$ values for the high-degree nodes. The black curve in this figure, clearly shows that the bigger the set becomes (by increasing $K$), the more influence it exerts on the network. The gray curve, however, is not strictly increasing. That is, a larger set of such nodes is not always the best solution when greater spread is desired.

The logarithmic growth in both curves should be taken into consideration as the similarity in their behavior, as well as their difference that the impact of the most influential set is always greater than of its corresponding set of high-degree nodes.

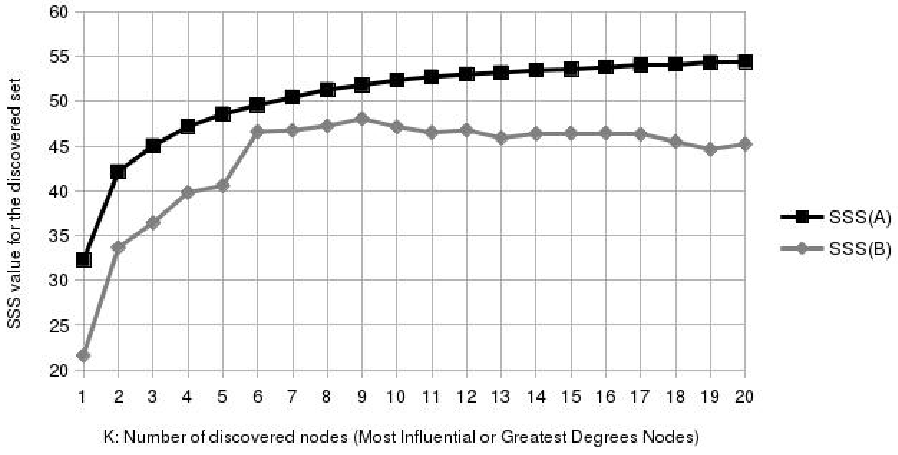


Figure 4.1: Behavior of the `RankedReplace` method with different $K$'s

If we take a closer look at those 20 experiments and observe how the nodes are selected, we can see that the RankedReplace method does look for the nodes that are of the greatest influence in correlation with other nodes, and nodes that individually have a great influence on the network are not necessarily the chosen ones.

Table 4.1 shows the number of nodes added (+) and removed (−) in each step. As one may observe, although we force the method to discover only one more node in every run, in several attempts more than one modification takes place for the best decision to be made. In other words, increasing the size of the requested set by one, may cause more than one modification. This is perfectly in line with the design of the algorithm which aims at the discovery of the most influential set of size $K$ and not just a collection of the $K$ most influentials.

On the other hand, finding the set of the $K$ high-degree nodes is a straightforward procedure - nodes are chosen only because of their individual features that is their degrees. That is why we always see 1 in the second + column.

Table 4.1: Behavior of the RankedReplace method in terms of selecting new nodes
+: Number of added nodes in this run
−: Number of removed nodes in this run

| N | ***SSS*(A)** | + | − | ***SSS*(B)** | + | − |
|---|---|---|---|---|---|---|
| 1 | 32.26 | 1 | 0 | 21.6 | 1 | 0 |
| 2 | 42.14 | 1 | 0 | 33.66 | 1 | 0 |
| 3 | 45.02 | 1 | 0 | 36.43 | 1 | 0 |
| 4 | 47.16 | 1 | 0 | 39.81 | 1 | 0 |
| 5 | 48.55 | 1 | 0 | 40.57 | 1 | 0 |
| 6 | 49.55 | 1 | 0 | 46.59 | 1 | 0 |
| 7 | 50.44 | 1 | 0 | 46.72 | 1 | 0 |
| 8 | 51.27 | 1 | 0 | 47.24 | 1 | 0 |
| 9 | 51.79 | 1 | 0 | 48.03 | 1 | 0 |
| 10 | 52.32 | 1 | 0 | 47.13 | 1 | 0 |
| 11 | 52.71 | 1 | 0 | 46.49 | 1 | 0 |
| 12 | 53 | 1 | 0 | 46.75 | 1 | 0 |
| 13 | 53.18 | 2 | 1 | 45.9 | 1 | 0 |
| 14 | 53.44 | 3 | 2 | 46.37 | 1 | 0 |
| 15 | 53.55 | 3 | 2 | 46.41 | 1 | 0 |
| 16 | 53.78 | 3 | 2 | 46.44 | 1 | 0 |
| 17 | 54.01 | 5 | 4 | 46.3 | 1 | 0 |
| 18 | 54.12 | 2 | 1 | 45.47 | 1 | 0 |
| 19 | 54.32 | 4 | 3 | 44.63 | 1 | 0 |
| 20 | 54.39 | 2 | 1 | 45.21 | 1 | 0 |

### 4.3.2. Expected Results

Let's once again review our assumptions: We have two copies of a network, A and B, where:

- **A** is a copy of the network in which only the $K$ most influential nodes are vaccinated and therefore they are unable to take part in the spread.
- **B** is another copy in which only the $K$ high-degree nodes are vaccinated and therefore unable to take part in the spread.

And the selected nodes through which the spread will be initiated (*initially infected* nodes) are chosen randomly and yet the same in both copies. This set of nodes contains no vaccinated nodes either in A or B.

It is trivial that having any set of nodes of any size vaccinated in a network, would diminish the degree of spread (in our experiment: the infection degree). If, instead of any random set, the most powerful nodes of a network become disabled (or vaccinated) to pass the virus, this network should be intuitively and logically expected to have the lowest infection degree in comparison with any other copy of the network where any selection of $K$ nodes are vaccinated.

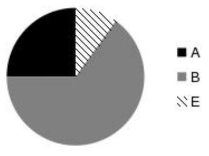
Therefore, if we spread the virus in both A and B, using the same spreading method, A should always have fewer number of infected nodes as a result of this virus spread. In the following section we investigate whether we achieve what we expect or not.

### 4.3.3. Results of The Experiments

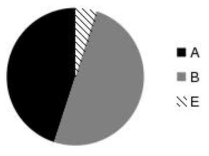
The particular experiment that we want to employ to explain the conclusion, is carried out on a data set representing a network of **US Airports** containing 332 nodes which are connected with weighted edges. This data set was originally provided in *Vladimir Batagelj*'s web page[1] for public use.

The table of pie charts (figure 4.2) shown below is the abstract of 200 experiments. Each pie chart is the result of 20 runs of the experiment with constant values for $K$ and $S$. This many runs for each pair of $K$'s and $S$'s are to minimize the side-effects caused by the randomness in selecting the *initially infected* nodes. Since each experiment compares two copies of the network, A and B, a pie chart is used to illustrate the relative frequency of a society with higher infection degree after the spread.

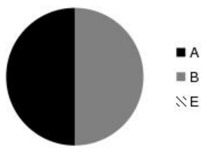
Figure 4.2: Relative frequency of the experiment (A or B) with higher infection degree.
Data-set: US Airports
Total Nodes: 332
Default weights of the graph is maintained.
**A** represents the relative frequency of the times A has higher infection degree.
**B** represents the relative frequency of the times B has higher infection degree.
**E** represents the situations in which A and B are equally infected.



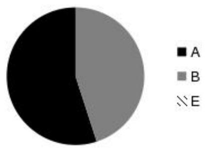
(a) K = 5, S = 5



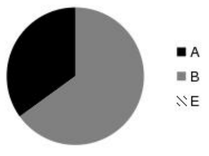
(b) K = 5, S = 10



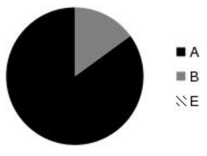
(c) K = 5, S = 20



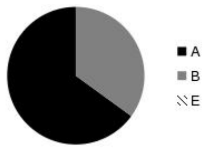
(d) K = 10, S = 5



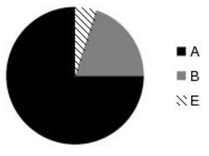
(e) K = 10, S = 10



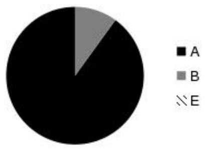
(f) K = 10, S = 20



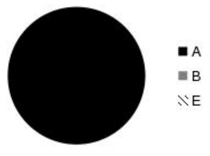
(g) K = 20, S = 5



(h) K = 20, S = 10



(i) K = 20, S = 20



(j) K = 20, S = 30

In every row of this table of pie charts, $K$ remains the same while $S$ varies. As we explained in Section 4.3.2 it seems logically correct to expect A to always experience a lower infection degree in comparison with B. However, despite our expectation, not only we do not see the pie charts *always* fully gray, but also it seems that as $K$ increases, the tendency is to have a greater infection degrees in A. This means:

> In a society where the most influential nodes are unable to play their critical role in spreading information bits, the virus spreads in a greater scale than the society in which the high-degree nodes are disabled to participate.

This observation made us think that the distributions of weights in this particular network might be playing a major role and causing such surprising results. The minimum weight of

[1]`http://vlado.fmf.uni-lj.si/pub/networks/data/`

this network is 0.0009 and the maximum weight is 0.5326, with the skewness of 1.6096 and the average of 0.0722. Since the chances for the virus to pass through edges with such small weights are very low, the difference between a normal node and a node with a great influence might become insignificant.

Thus, we decided to reconstruct this network and increase the chances of the spread by multiplying its weights once by 3 and once by 5 (and setting any weights exceeding 1.0 to 1.0). The distribution of weights in the reconstructed network (which was multiplied by 5) shows the skewness of 0.8999. In the figures 4.3 and 4.4 the difference of distributions is illustrated. In figure 4.5 the results of our experiments on the reconstructed network is illustrated.

In this second attempt (figure 4.5), it is clearly visible that the spread of the virus in A (the network where the most influential nodes are vaccinated and unable of taking any action) is more often greater than the spread in B. This is exactly in contradiction with our expectation explained in Section 4.3.2.

In the next section we explain why this is not in line with our intuitive expectation and why it is perfectly natural.

Figure 4.3: Distribution of weights ×1 in the default network

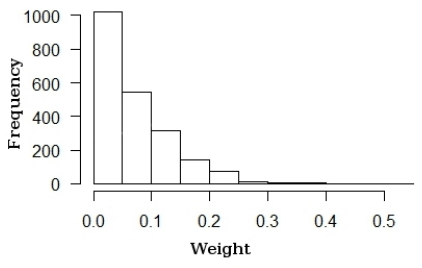


Figure 4.4: Distribution of weights ×5 in the modified network

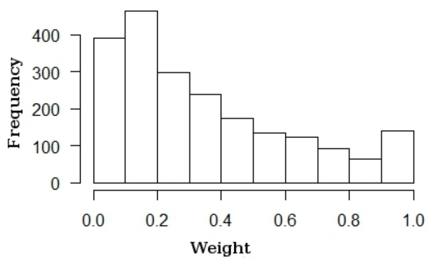
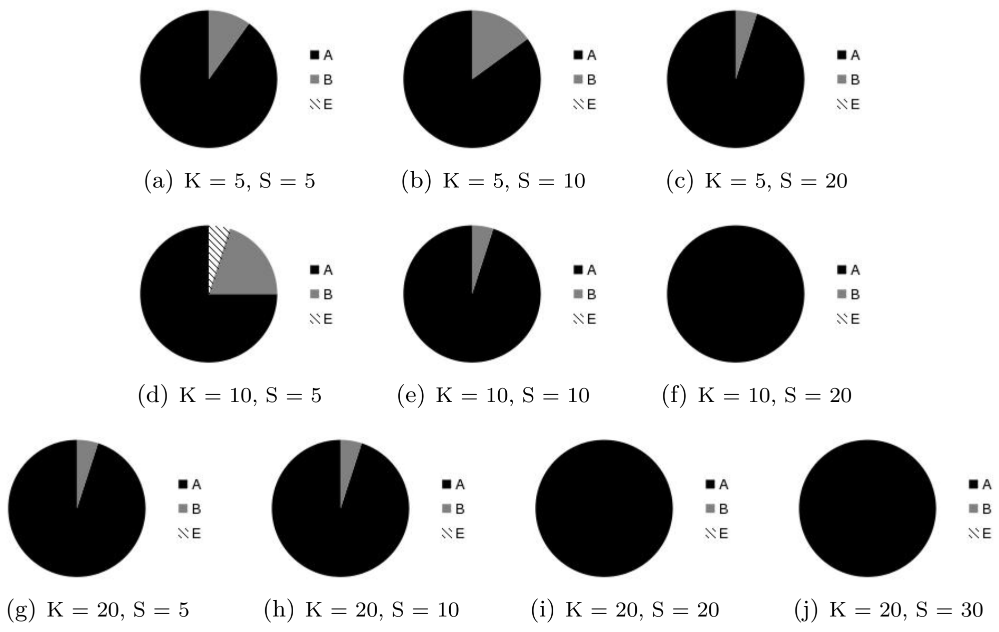


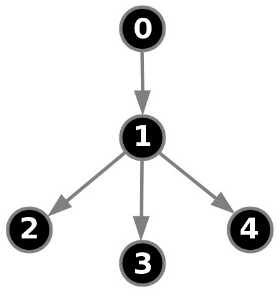
Figure 4.5: Relative frequency of the experiment (A or B) with higher infection degree.
Data-set: US Airports
Total Nodes: 332
Weights of the graph is 5 times greater than its original ones.
**A** represents the relative frequency of the times A has higher infection degree.
**B** represents the relative frequency of the times A has higher infection degree.
**E** represents the situations in which A and B are equally infected.



### 4.3.4. The Reason For This Contradiction

To explain the reason, we need to consider a very simple graph shown in the figure 4.6 where the node 1 is a high-degree node ($deg_{out}(1) = 3$) having 3 outgoing edges. In this simple graph we are not considering weights of the edges which determine the power of nodes in terms of their influences, but without loss of generality we may assume that the node 0 is a most influential node because it can have influences on 4 nodes (i.e 1,2,3 and 4). Having this assumption, the node 0 is the best candidate to which providing information bits maximizes the spread but what we are experimenting directly does not concern the presence of the most influential nodes, but their absence.

Figure 4.6: A most influential node (0) depending on a node with a high degree (1)



The key point is that if we disable the node 0 from passing information bits, the node 1 is still a high-degree node and in case it receives information bits it will not have any less power in spreading them - It has a good chance of passing information bits to the nodes 2, 3 and 4. However, the vaccination of node 1 results in a very different situation. In this case, node 0 is not a most influential node any more, and therefore its power in spreading information bits becomes significantly lower than its power in the presence of 1.

### 4.3.5. Final Conclusion

In general, the *Flow Authorities* model discovers the most influential nodes that indeed form a highly reliable set of nodes to be employed if the goal is to spread the information bits. But now we can confidently state that this set of nodes is absolutely not the right selection to be protected if the goal is to minimize the spread. For instance, in order to secure the physical Internet routers of a country, we should not use this model to find the routers that require maximum protection, or in case of an epidemic, concentration on the most influential inhabitants discovered by this model does not guarantee eradication of the disease.

In Section 4.3.2, we explained why it is a reasonable prediction for the network A to have a lower infection spread and now we see that the exact opposite of our expectation took place - the spread in the network A where the most influential nodes are vaccinated is actually higher. That is so, as we explained in 4.3.4, because the vaccination of a set of selective nodes may affect the topology of the network and change the features of its nodes. Therefore, we may draw the following conclusion:

> The most influential nodes discovered by the *Flow Authorities* model **must not** be used in any practices that aim at protecting a network from the spread of the information bits.

Chapter 5

# Business Analysis

In this chapter we attach the business analysis documentation exactly as it was prepared and agreed upon prior to the implementation of the application. There were several assumptions that had to be changed on later versions. You can see these changes in Chapter 6 *Technical Documentation.*

## 5.1. Document Change Control

### 5.1.1. Document Metric

| **Document Metric** | | | | | . |
|---|---|---|---|---|---|
| **Project** | Most Desired | | | **Company** | WUT |
| **Name** | Most Desired | | | | |
| **Topics** | Problem Definition, Functional/Non-functional Requirements, Planning | | | | |
| **Authors** | Azim Ahmadzadeh | | | Sajjad Hassanpour | |
| **Files** | BA-Thesis.tex, BA-Thesis.pdf | | | | |
| **Version** | 0.9 | **Status** | Ongoing | **Opening Date** | 15.10.2015 |
| **Summary** | The detailed description of development procedure | | | | |
| **Authorized by** | | | | **Last Modified** | 26.10.2015 |

### 5.1.2. Document History

| **Version** | **Change Description** | **Author** | **Date** |
|---|---|---|---|
| 0.1 | Initial Document's Structure | Sajjad Hassanpour | 17.10.2015 |
| 0.2 | Problem Definition | Azim Ahmadzadeh | 19.10.2015 |
| 0.3 | Goals and Benefits | Azim Ahmadzadeh | 21.10.2015 |
| 0.4 | GUI and Functional Requirements | Sajjad Hassanpour | 23.10.2015 |
| 0.5 | Revision | Azim Ahmadzadeh | 26.10.2015 |

### 5.1.3. Document Approvers

| Role | Name | Date | Signature |
|---|---|---|---|
| Supervisor | Dr.Tomasz Brengos | 27.10.2015 | Taken |
| Instructor | Prof.Wladyslaw Homenda | 27.10.2015 | Taken |

### 5.1.4. Document Scope

The scope of this requirement document describes the importance of the ability to discover the most influential nodes of a social network of any kind and the functionality of this desktop application that we develop to test the effectiveness of the algorithm that we employ and illustrate the results.

### 5.1.5. Document Guidelines

Requested Delivery Date: 27.10.2015

- Approvals via email will be stated as such under Signature column.
- Above list of people will also approve the deployment of the system into production.

### 5.1.6. Notion Dictionary

| **Notion** | **Explanation** |
|---|---|
| *Social Network* | a group of people/objects connected with each other according to certain rules (e.g the network representing the students' interaction in a school) |
| *Virtual Social Networks* | any application in the same category as Facebook, Twitter, Whatsap, etc. |
| *Graph* | a mathematical tool which we use to describe a given Social Network. |
| *Nodes* | elements of a graph representing persons/objects of a social network. |
| *Edges* | elements of a graph representing the relationship/connectivity between nodes. |
| *Data-sets* | a plain text file containing the basic information about the proposed social network. |
| *Small Data-set* | at this point we assume a data-set containing less than 100 nodes to be small. (This may change in later agreements due to experiments.) |
| *Large Data-set* | at this point we assume a data-set containing greater than 1000 nodes to be large. (This may change in later agreements due to experiments.) |

## 5.2. Introduction

### 5.2.1. Background and Problem Definition

The behavior of a society towards different issues has always been an interesting subject for Sociologists and Economists. This enthusiasm was the motive behind a myriad of theorems to be discovered and proved. Since the very beginning of the twenty-first century, thanks to the creation of virtual social networks, we have been given the chance to re-evaluate these theories much more precisely that before and then employ them in a huge variety of subjects.

In the present project, we are interested in having a light-weight desktop application which can assist users in understanding the famous social network problem which is called the *Campaign Problem* and applying a technique to find a solution for that. A solution, from the business point of view, can be viewed as a set of target customers in them a business is interested. The same method may be applied to find the most critical nodes of a network that should be protected vigilantly due to security reasons. This is discussed in *Business Problem* and *Purpose and Benefits* sections in more details.

**Campaign Problem** "is the problem of identifying a set of target individuals whose positive opinion about an information item will maximize the overall positive opinion for that item in the social network." [1]

### 5.2.2. Business Problem

It is very critical for businesses to constantly feed their market with new products. This is called advertisement and it is usually not cheap. Since it is impossible to have the whole network of the market under the influence of the ads, finding a comparatively small subset of the whole market can be extremely cheaper and even more effective in spreading the message. This small group of customers may not exactly be the regular customers of a business but the best in delivering its marketing messages. This software aims at finding such a subset from a given database.

In addition to the marketing application, it can also be applied to other fields of study within which an opinion, disease, etc. has become epidemic. This can teach us how to choose the best approach to combat the spread or in some cases to promote the spread.

### 5.2.3. Goals

There are two main goals in making of this software:

- To provide a technique which discovers an answer for the aforementioned problems.
- To provide the ability to check the correctness and effectiveness of such a technique in real examples.

### 5.2.4. Benefits

In this project, we employ one of the best algorithms suggested recently to find the $k$ nodes of a given network from the whole population $S$, that guarantees that the set contains the most influential nodes; e.g. the best customers in terms of passing the market's message;

or the prior choices to cure immediately if the aim is to combat an epidemy. Of course for any given $k$, a different set is chosen and this is the user (e.g. the business manager) who decides on the size of the required set. In practice, this can be seen as an important marketing/management decision since there definitely is a certain budget allocated to invest on those $k$ customers/patients, therefore, a greater $k$ requires a bigger budget, and without having this target set, our advertisement/cure would be much less efficient and somehow blind.

In addition to discovering the most influential nodes, this application should go further and illustrate a graphical model through which in some fashion the user can see the reliability of the aforementioned algorithm. There will be two graphical models suggested to satisfy this need that we will discuss later.

### 5.2.5. Critical Success Factors

The following Critical Success Factors represent business driven criteria for all deadlines that, if met, will measure the success for the delivery of this initiative.

- The ability to load large data sets of a specific format.
- Representation of the most important facts regarding any given data set of a specific format.
- Success in applying the algorithm and outputting the most influential subset.
- Implementation of a graphical model to illustrate the relative correctness of the solution.

### 5.2.6. Requirement Assumptions and Constraints

These terms are beyond the requirement assumptions and were already agreed to be covered in this software:

- The ability of the software to accept any data set as its input in any format apart from the one that is agreed upon.
- The ability of the software to illustrate the graphical model for very large (greater than 1000 nodes) data sets.
- The ability of the software to function in any Operating Systems apart from Linux distributions and Microsoft Windows.

### 5.2.7. Open Issues

| Id | Issue | Resolution |
|---|---|---|
| 1 | The choice of the graphical model | Two suggestions are given regarding the illustration model in the sectoin 4.7 |

## 5.3. Non-Functional Requirements

### 5.3.1. Platform Requirements

This software is required to work on both Linux (popular distributions e.g *Ubuntu*, *Mint*, *Arch*) and MS Windows operating systems that have *JAVA* installed.

### 5.3.2. Performance Requirements

This software should perform the same in both Linux distributions and Microsoft Windows operating systems. It must find a proper answer for any given data sets, large or small, but regarding the graphical module, it is only responsible for small data sets to fulfill an intuitive understanding purpose.

## 5.4. Functional Requirements

The followings are the Functional Requirements. Each requirement has a unique identifier for tracking and traceability purposes.

| *FR Id* | *FR Description* |
|---|---|
| 1 | The Graphical User Interface of the software (GUI) |
| 2 | Loading/saving options for data sets/solutions |
| 3 | The accepted format of the input |
| 4 | The accepted format of the output |
| 5 | Summary of the input in GUI |
| 6 | Summary of the solution in GUI |
| 7 | Graphical representation for the application of the algorithm |

### 5.4.1. FR 1 - Graphical User Interface

This software must have a graphical user interface the details of which are described in the next sections.

### 5.4.2. FR 2 - Loading/saving options for data-sets/solutions

There must be two options for the user, regarding *loading* and *saving*. The loading enables the user to use the GUI to load the desired data sets into the system and then see the summary of the input file in GUI. Obviously the functionality of this module depends on the format of the input file which is discussed in the next section.

### 5.4.3. FR 3 - The accepted format of the input

In the first release of the software we expect the input file to be a plain text file with ”txt” format containing several lines where each line describes the relationship between one node and one of its neighbors. For example, 4 : 12 : 0.12 means the node indexed with 4 has a neighbor indexed with 12 and the weight of the edge going from the first node to the second node is 0.12. Of course it is possible to add some other types of files later, but the ”txt” file is a *must*.

### 5.4.4. FR 4 - The accepted format of the output

At this point we assume the format of the output should be compatible with the input file thus the software should produce the output in the ”txt” format. Every output is a solution found by the software containing the index of all nodes which was discovered by the provided technique. The content of the output is one line of plain text storing the discovered indexes separated by ”:”.

### 5.4.5. FR 5 - Summary of the input in GUI

In GUI there should be provided a panel, called *Input Data Panel*, where the user can see a summary of the data that they loaded on the memory. You can see the smallest set of information that is expected in this panel in this example:

| | | |
|---|---|---|
| Total | : | 10546 |
| Min W | : | 0.02 |
| Max W | : | 0.89 |
| Total i: 56 | | |

where **W** stands for the *weight of an edge* and **Min W** and **Max W** showing the *minimum existing weight* and *maximum existing weight* respectively, and finally, **Total i** gives information about the *number of isolated nodes.*

### 5.4.6. FR 6 - Summary of the solution in GUI

Similar to the Input Data Panel, another panel called *Solution Panel* should be provided that gives the same pieces of information but only regarding the solution.

### 5.4.7. FR 7 - Graphical illustration for the application of the algorithm

The details about the graphical illustration is an open issue (with ID = 1) at this level. The purpose of this module is to graphically show the user the applications of the adopted technique. Two scenarios are suggested:

1. In the graphical module, the given graph will be drawn on a panel and the user can see the density of the edges going through every node. This can help the user to verify the correctness of the solution provided by this application.
2. Having the information about the given network, we design a matching game where nodes pass pieces of information to their neighbors. Before starting the game, once the user can randomly initialize nodes with data (some will have them and some will wait)

and another time he/she can initialize them based on the discovered subset of nodes (all nodes of the subset will have the information and the remaining will wait). Comparing the time needed for the flow of information in both models to become stable, one can have an estimation on the correctness of our technique.

It was agreed to decide on the graphical model later.

## 5.5. Planning Criteria

We planned the schedule of all steps of this project as follows:

| Term | Deadline | Milestone |
|---|---|---|
| 1-st | 27.10.2015 | Schedule and preliminary requirements analysis |
| 2-nd | 17.11.2015 | Detailed requirements analysis and preliminary architectural documentation |
| 3-rd | 01.12.2015 | Detailed architectural documentation and preliminary implementation |
| 4-th | 15.12.2015 | technical documentation and advanced implementation |
| 5-th | 12.01.2015 | Finished set of documentation, user manual and final version of the project |

### 5.5.1. Specific Testing Requirements

At least one sample as a large data set and one as a small data set should be provided with every release of the application at any deadlines. The large data set should be chosen in a way that guarantees the verifiability of the algorithm.

## 5.6. Use Cases

1. Brief Description
   The Use Case steps are to be performed by the user to obtain the solution and observe the application of the algorithm.
2. Actors
   - Most Desired's User
   - Most Desired System
3. Preconditions
   - A large/small data set must be available in the agreed format to obtain any solution.
   - A small data set must be available in the agreed format to see the graphical illustration.

4. Triggers
    - When the user tries to **load** a data set of the **proper format** using the load button:
        - the whole data set will be read and stored in RAM
        - the user can see a summary of the input data in the input data panel including the number of nodes and edges, min and max weight.
        - the software continues to wait for the user's action.
    - When the user tries to **load** a data set of any **improper format** using the load button:
        - an error message will pop up,
        - the software continues to wait for the user's action.
    - When the user tries to **save** immediately after the load action using the save button,
        - an error message will pop up indicating that no procedure is applied yet and no solution is obtained.
        - the software continues to wait for the user's action.
    - When the user tries to **find a solution** using the find solution button, after an input was successfully read and a number as the value of $k$ was given using the k-textbox,
        - the software will run the algorithm on the given input and when the computation is complete, a subset of the given data set is found,
        - a summary of the solution appears the solution panel in GUI,
            * the save button will be available to function,
            * in case of the small data sets, the illustrate button will be available to function.
    - When the user tries to **save** using the save button after successful completion of the algorithm,
        - a pop-up window asks the directory in which the user wants the solution to be stored,
        - then indexes or IDs or names of the discovered nodes will be stored in the agreed format,
        - the software continues to wait for the user's action.
    - When the user tries to **illustrate** using the illustrate button after successful completion of the algorithm,
        - if the data set is considered to be large:
            * an error message informs the user that the given data is too large to be illustrated.
        - if the data set is considered to be small:
            * the graphical model is shown. (Not agreed on details yet.)

# Chapter 6

# Technical Documentation

## 6.1. Document Change Control

### 6.1.1. Document Metric

| **Document Metric** | | | | | . |
|---|---|---|---|---|---|
| **Project** | Most Desired | | | **Company** | WUT |
| **Name** | Most Desired | | | | |
| **Topics** | Technical models, System architecture, Graphical UI | | | | |
| **Authors** | Azim Ahmadzadeh | | | Sajjad Hassanpour | |
| **Files** | FinalDoc-Thesis.tex, FinalDoc-Thesis.pdf | | | | |
| **Version** | 1.0 | **Status** | Done | **Opening Date** | 1.11.2015 |
| **Summary** | The technical descriptions of the project | | | | |
| **Authorized by** | | | | **Last Modified** | 10.01.2016 |

### 6.1.2. Document History

| **Versi** | **Change Description** | *Author* | *Date* |
|---|---|---|---|
| 0.1 | General structure of the technical documentation | Azim Ahmadzadeh | 19.10.2015 |
| 0.2 | Design of the UML diagrams | Sajjad Hassanpour | 17.10.2015 |
| 0.3 | Documentation of the system structure | Azim Ahmadzadeh | 21.10.2015 |
| 0.4 | Description of Classes and Interfaces | Sajjad Hassanpour | 23.10.2015 |
| 0.5 | Description of Methods | Sajjad Hassanpour | 23.10.2015 |
| 0.6 | Description of modules' interactions | Sajjad Hassanpour | 23.10.2015 |
| 0.7 | Final documentation for the stage II | Azim Ahmadzadeh | 26.10.2015 |
| 0.8 | Change of data structure to HashMap | Azim Ahmadzade | 19.11.2015 |
| 0.9 | Change of data structure to Lists | Azim Ahmadzadeh | 23.11.2015 |
| 0.10 | Adding click listener to objects | Sajjad Hassanpour | 03.12.2015 |
| 0.11 | Implementation of *SSS* method | Azim Ahmadzadeh | 07.12.2015 |
| 0.12 | Adding zoom functionality to the graphical panel | Sajjad Hassanpour | 08.12.2015 |
| 0.13 | Adding another GUI for user's input data | Sajjad Hassanpour | 11.12.2015 |
| 0.14 | Finishing implementation of *SSS* method | Azim Ahmadzadeh | 11.12.2015 |
| 0.15 | Finishing implementation of Ranked-Replace method | Azim Ahmadzadeh | 13.12.2015 |
| 0.16 | First integration of visualization and algorithm | Sajjad hassanpour | 13.12.2015 |
| 0.17 | Correctness test of algorithm for different data | Azim Ahmadzadeh | 13.12.2015 |
| 0.18 | Adding JUnit test cases | Azim Ahmadzadeh | 14.12.2015 |
| 0.19 | Adding progress-bar to GUI | Sajjad hassanpour | 14.12.2015 |
| 0.20 | Design a more user-friendly GUI | Azim Ahmadzadeh | 03.01.2016 |
| 0.21 | Integration of the new GUI with the program | Sajjad Hassanpour | 05.01.2016 |
| 0.22 | Choosing a virus spread model | Azim Ahmadzadeh | 09.01.2016 |
| 0.23 | Design of the virus spread algorithm | Azim Ahmadzadeh | 11.01.2016 |
| 0.24 | Integration of the virus spread classes | sajjad Hassanpour | 14.01.2016 |
| 0.25 | Change of the graph's GUI | Sajjad Hassanpour | 15.01.2016 |
| 0.26 | Integration of the new graph'GUI | Sajjad Hassanpour | 17.01.2016 |

### 6.1.3. Document Approvers

| Role | Name | Date | Signature |
|---|---|---|---|
| Supervisor | Dr.Tomasz Brengos | 17.01.2016 | Taken |
| Instructor | Prof. Bohdan Macukow | 17.01.2016 | Taken |

### 6.1.4. Document Scope

This document provides better and further details with respect to the agreed requirement specifications. The scope of this technical document describes the structure of all modules and their dependencies, the integration model, the development model and the programming platform. In addition, technical difficulties are discussed and a solution is proposed.

### 6.1.5. Document goals

In this document, we aim at describing as many details of the application as possible, according to all terms of the agreement, based on the Business Analysis provided before.

### 6.1.6. Document Guidelines

Requested Delivery Date: 12.01.2016

- Approvals via email are stated as such under Signature column.
- Above list of people also approve the deployment of the system into production.

## 6.2. Basic Assumptions

### 6.2.1. Technical Models

All technical models needed to start this project are agreed as follows:

| Subject | Model | Description |
|---|---|---|
| *Development* | Waterfall [Classic] | Each phase starts after completion and termination of the previous one. |
| *Integration* | Ad Hoc | Modules are connected in the same way they are produced. |
| *Tests* | | Unit tests → Integration tests → System tests |

### 6.2.2. Programming Infrastructure

The employed programming language is JAVA, chosen based on two proposals (JAVA and C-Sharp). The Integrated Development Environment will be ECLIPSE under Eclipse Public License EPL. We employ JAVA SE (Standard Edition) for algorithms implementation and JAVA SWING technology for achieving the graphical purposes.

### 6.2.3. Critical Success Factors

The success level for this project will be achieved if:

1. the system accepts the input file of the properly designed format with no exception.
2. the system reads the content of the input file and stores it as a graph.
3. the system discovers the expected number of elements from the input file.
4. the discovered set by the system is an optimal solution.
5. the system is able to illustrate medium-sized graphs ($<$ 1000 nodes) in the graphical mode.
6. the system is able to correctly distinguish the discovered set from the rest of the graph for the user.
7. the presented test should provide the user with a graphical evaluation for the effectiveness of the algorithm.

## 6.3. System Overview

### 6.3.1. Input

*MOST DESIRED* expects an input file in a format tailored as it is explained here. The input file should be a *txt* file containing information about the graph of the social network. Every line of this file describes an edge of the graph with the following structure:

$\langle$ ID of the starting node$\rangle$ $\langle$ ID of the ending node $\rangle$ $\langle$ weight of this edge$\rangle$

and an example would look like "`120 85 0.26`" which means the node with `id=120` is connected to the node with `id=85` where the weight of this (directed) edge is `0.26`.

**Remark 6.1.** Formerly, the expected format was slightly different: the IDs of nodes was separated with an *underscore* instead of a *single space*. We made this minor change only because the new format helps us make the process of reading the input data faster. In this format the program no longer needs to split each line to distinguish the values.

### 6.3.2. System architecture

The general structure of the system is illustrated using UML diagrams that are shown here. The *Class Diagram* (6.1) and the *Sequence Diagram* (6.2) help us picture how classes of this system are implemented and how they interact while the user is using the application. In addition, the *Action Diagram* (6.3) of the application is brought to help understand of the application.

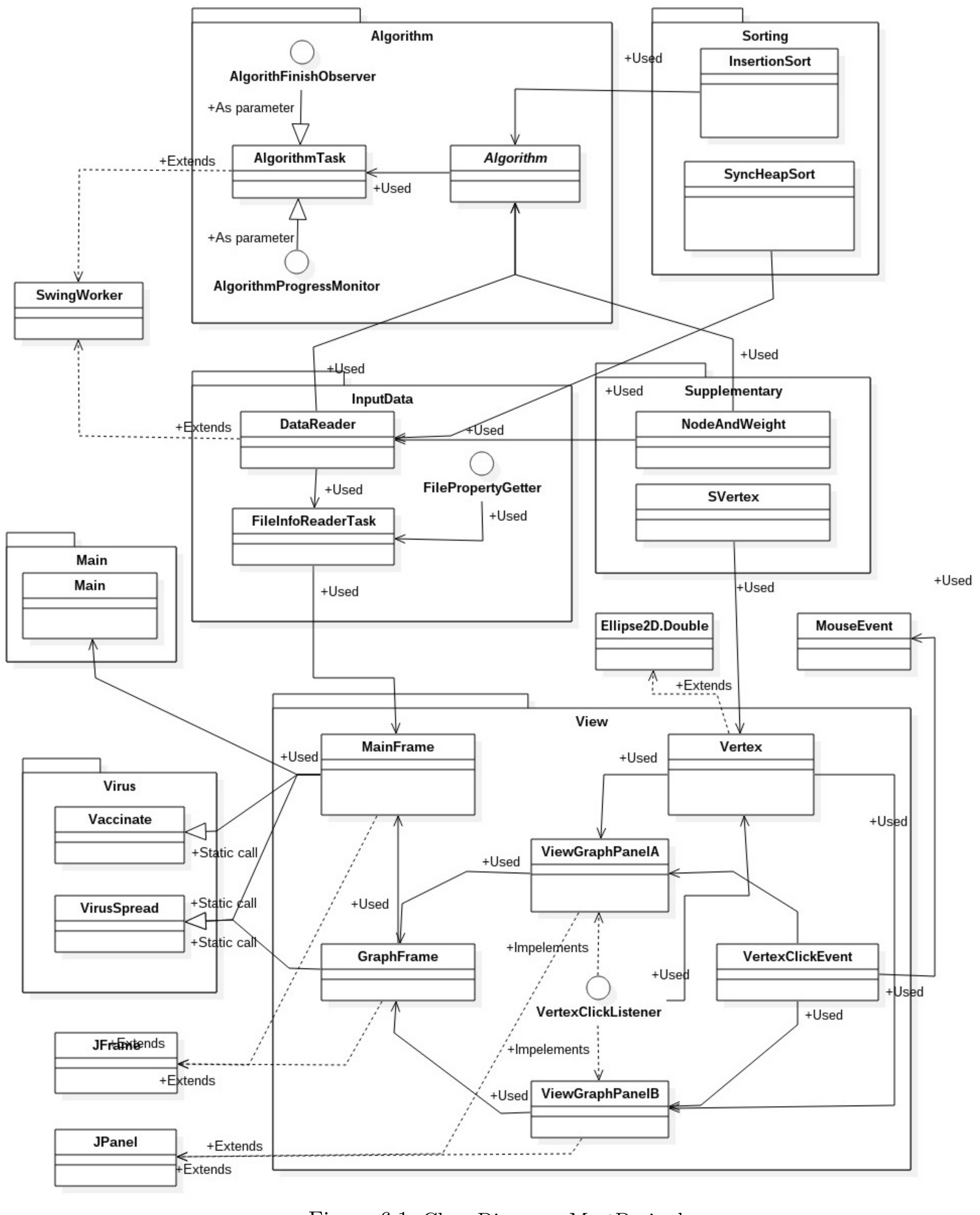


Figure 6.1: Class Diagram: MostDesired

### 6.3.3. Classes

In this section, first we briefly list all the classes and interfaces of the architecture and then give more details about the important ones.

In the table below, we list all classes and interfaces needed to be implemented, in addition to their type, modifier and the name of their wrapping packages. Since all classes and interfaces

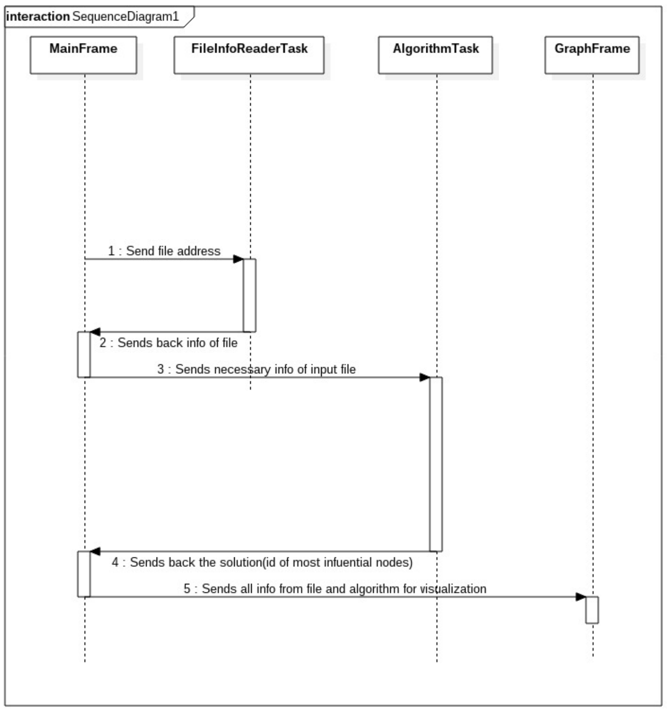

interaction SequenceDiagram1
MainFrame
FileInfoReaderTask
AlgorithmTask
GraphFrame
1 : Send file address
2 : Sends back info of file
3 : Sends necessary info of input file
4 : Sends back the solution(id of most infuential nodes)
5 : Sends all info from file and algorithm for visualization

Figure 6.2: Sequence Diagram: MostDesired

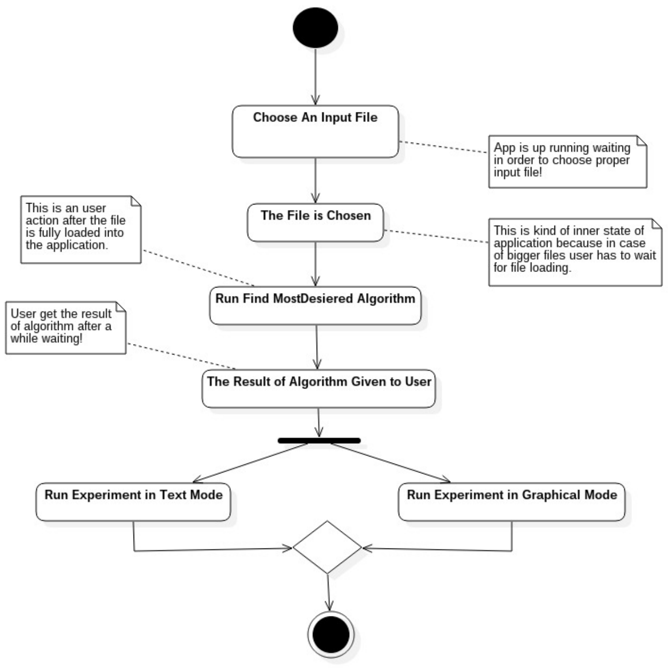

Choose An Input File
App is up running waiting in order to choose proper input file!
The File is Chosen
This is kind of inner state of application because in case of bigger files user has to wait for file loading.
This is an user action after the file is fully loaded into the application.
Run Find MostDesiered Algorithm
User get the result of algorithm after a while waiting!
The Result of Algorithm Given to User
Run Experiment in Text Mode
Run Experiment in Graphical Mode

Figure 6.3: Action Diagram: MostDesired

are assumed to be defined as *public*, their access level is not mentioned. In this table, **C** stands for *Class* and **I** for *Interface*.

| No | Name | C/I | Modifier | Extends | Package |
|---|---|---|---|---|---|
| *1* | Main | C | - | - | main.sajjad.mostDesired |
| *2* | DataReader | C | - | - | inputData.azim.mostDesired |
| *3* | Algorithm | C | Abstract | - | algorithm.azim.mostDesired |
| *4* | SVertex | C | - | - | model.sajjad.mostDesired |
| *5* | InsertionSort | C | - | - | sortingClasses.azim.mostDesired |
| *6* | SyncHeapsort | C | - | - | sortingClasses.azim.mostDesired |
| *7* | NodeAndWeight | C | - | - | supplementaryClasses.azim.mostDesired |
| *8* | AlgorithmTask | C | - | SwingWorker | task.sajjad.mostDesired |
| *9* | AlgorithmFinished | I | - | - | task.sajjad.mostDesired |
| *10* | ProgressMonitor | I | - | - | task.sajjad.mostDesired |
| *11* | CloseListener | I | - | - | view.sajjad.mostDesired |
| *12* | FileInfoReaderTask | C | - | SwingWorker | view.sajjad.mostDesired |
| *13* | FilePropertyGetter | I | - | - | view.sajjad.mostDesired |
| *14* | MainFrame | C | - | JFrame | view.sajjad.mostDesired |
| *15* | Vertex | C | - | Ellipse2D | view.sajjad.mostDesired |
| *16* | VertexClickEvent | C | - | - | view.sajjad.mostDesired |
| *17* | VertexClickListener | I | - | - | view.sajjad.mostDesired |
| *18* | ViewGraphFrame | C | - | JFrame | view.sajjad.mostDesired |
| *19* | GraphFrame | C | - | JFrame | newView.sajjad.mostDesired |
| *20* | ViewGraphPanel | C | - | JPanel | view.sajjad.mostDesired |
| *21* | Vaccinate | C | Abstract | - | virusSpread.azim.mostDesired |
| *22* | VirusSpread | C | Abstract | - | virusSpread.azim.mostDesired |
| *23* | AlgorithmTest | C | - | - | test.azim.mostDesired |
| *24* | DataReaderTest | C | - | - | test.azim.mostDesired |

**Public Class DataReader{...}**

- **Definitions**: Suppose `X` is a node. From `X`'s perspective:
  - an `incoming edge` is an incident edge starting from an `X`'s neighbor and ending at `X`.
  - an `outgoing edge` is an incident edge starting from `X` and ending at any of its neighbors.
  - an `outgoing neighbor` of `X` is a neighbor incident to an edge starting from `X`.
  - an `incoming neighbor` of `X` is a neighbor incident to an edge ending at `X`.
- The **Input File** is a text file which describes a graph in the following fashion:
  1. Every line, representing an edge, contains 3 numbers separated by single space.
  2. The first number (integer) represents the ID of the starting node.
  3. The second number (integer) represents the ID of the ending node.
  4. The third number (double) represents the weight of this edge.
  5. The ID of nodes starts from zero.

6. Some nodes may be absent in this file, because they are isolated nodes.

- The following **Data Structure**s are designed to store the input file's data:

  1. **`nodesList_Out`**
     - It contains a lists of objects of type `NodeAndWeight`.
     - Its $i-th$ element is a set of `outgoing neighbor`s of i.
     - Its size is equal to the total number of vertices ($= maxIndex + 1$).
     - We do not let any of these lists be empty: They always contain a special instance of class `NodeAndWeight` no matter whether they have any `outgoing neighbor`s.
  2. **`nodesList_In`**
     - It is the same as `nodesList_Out` when we replace `outgoing neighbor`s by `incoming neighbor`s.
  3. **`neighborsList_Out[i]`**
     - It contains objects of type `NodeAndWeight`.
     - Its $j-th$ element represents the $j-th$ `outgoing neighbor` of $i-th$ vertex and the weight of that `outgoing edge`.
     - Its size is equal to the number of `outgoing neighbor`s of the corresponding node.
  4. **`neighborsList_In[i]`**
     - It is the same as **`neighborsList_Out[i]`** when we replace the `outgoing neighbor`s by the `incoming neighbor`s and the `outgoing edge`s by the `incoming edge`s.

**Public Abstract Class Algorithm{...}**

This class contains two important methods:

-
  ```
  public static Double steadyStateSpread(
  ArrayList s,
  ArrayList<LinkedList<NodeAndWeight>> nodes_In
  )
  ```

  This computes the steady state probability for a set of nodes (given by $s$) chosen from all nodes. It also needs the transition probability matrix (given by `nodes_In`). From each node's point of view, we are interested in `incoming edge`s and not the `outgoing edge`s. The result is the expected number of nodes under influence of the set $s$.

-
  ```
  public static ArrayList<Integer> rankedReplace(
  ArrayList<LinkedList<NodeAndWeight>> nodes_In,
  int k
  )
  ```

This method is responsible for finding a set of k nodes which in collaboration with each other give the greatest steady state probability in comparison to any other set of *k* nodes. In other words, it finds the *k* most influential nodes.

**Public Class SVertex{...}**

This class is mainly designed to hold the information related to the the graphical presentation of the graph in addition to the virus spread procedure.

- Fields:
  - `private int id;`

    :the ID of each node
  - `private int x;`

    : the x coordinate of the node on the graphical panel
  - `private int y;`

    : the y coordinate of the node on the graphical panel
  - `private int d;`

    : the radius of a node on the graphical panel, representing its density
  - `private ArrayList<Integer> neighbors;`

    : a list of `incoming neighbors` of this node
  - `private boolean isInK;`

    : whether this node is one of the *k* most influential nodes or not
  - `private boolean isInMax;`

    : whether this node is one of the *k* high-degree nodes
  - `private boolean isInfectedA;`

    : whether this node is marked as an infected node in experiment A which aims the most influential nodes
  - `private boolean isInfectedB;`

    : whether this node is marked as an infected node in experiment B which aims the high-degree nodes
  - `private boolean isVaccinatedA;`

    : whether this node is marked as vaccinated in experiment A
  - `private boolean isVaccinatedB;`

    : whether this node is marked as vaccinated in experiment B

#### Public Class InsertionSort{...}

This is a class with a static sort method which uses the *Insertion Sort* algorithm. The difference is that this method takes two arrays and sorts them simultaneously - It sorts the first array in ascending order and meanwhile rearranges elements of the second array in correspondence with changes in the first one.

In our project, the combination of these two arrays gives a list of nodes sorted by their computed $SSS$ values where the $SSS$ value for a node is what the method `steadyStateSpread{}` returns having this node as its only initially infected node. In other words, using this class, we can sort nodes by their influence power towards the entire network while we do not lose keeping track of their ids, without using any complicated data structure.

"Although the *Insertion Sort* is one of the elementary sorting algorithms with $O(n^2)$ worst-case time, the insertion sort is the algorithm of choice when the data is nearly sorted (because it is adaptive)"[8]. Since in the main algorithm in our work, in *RankedReplace* method, we need to iteratively sort a list of nodes which does not change significantly, we employ this sorting method to achieve a lower computation time.

#### Public Class SyncHeapSort{...}

This is another class with a static sorting method which uses the *Heap Sort* algorithm. Like the previously mentioned class, this is also implemented to sort two arrays simultaneously, where one is sorted in ascending order and the other one is rearranged correspondingly.

This class was designed to be tested, in comparison to the *Insertion Sort* in regards with their computation time for nearly-sorted, very long arrays. Since the test results was in favor of the *Insertion Sort*, we decided not use this class in the implementation of our algorithm.

#### Public Class NodeAndWeight{...}

This class introduces a unique approach towards the graph representation. It only has two fields:

- `private int adjacentVertex;`
- `private int weight;`

In the class `Algorithm`, as we explained before, we use a list of lists to keep track of the neighbors of all nodes. The inner lists contain objects of type `NodeAndWeight`. This structure should be seen as follow: In the main list, the $i-th$ element, which is a list itself, contains neighbors of the node with ID $i$. The $j-th$ element of the list represents the $j-th$ neighbor of the node $i$. We need to know the ID of this neighbor, as well as the weight of the edge connecting this neighbor to the node $i$. Using the `NodeAndweight` class, the ID of this neighbor can be accessed by `this.getAdjacentVertex()` and the weight of the corresponding edge by `this.getWeight()`.

#### Public Class AlgorithmTask{...}

This is a child class of Java `SwingWorker<T,S>`. It is designed to run the `RankedReplace` algorithm in a background process. Parameters `T` and `S` are of types `Integer` and `String` respectively. These parameters by the means of `setProgress` and `publish` method are used to support the progress bar tailored in the bottom of the application and also the log state of the main console of the application.

### Public Interface AlgorithmFinished{...}

This Interface is defined to pass data from `MainFrame` to the `AlgorithmTask`. It passes the result of the `RankedReplace` method when `AlgorithmTask` finishes its task.

### Public Class FileInfoReaderTask{...}

Since the size of the input file can be significantly large, a separate thread is needed to read the input file. Therefore, this class is designed which extends `SwingWorker<T,S>` and reads the input file in a background process.

### Public Class FilePropertyGetter{...}

This interface is used to pass the information from `FileInfoReaderTask` to `MainFrame`.

### Public Interface ProgressMonitor{...}

Methods `setProgress` and `publish` can be called from `AlgorithmTask` and not from the `Algorithm` class itself. This interface is therefore designed to pass information between these classes and make monitoring the progress of the `RankedReplace` method possible.

### Public Interface CloseListener{...}

This is responsible for disposing of an object created of type `GraphFrame`. It is used when the graph visualization window is closed by the user.

### Public Class MainFrame{...}

This class contains the architecture of the main window of the application. For more information about this window, see Section 3.5.2.

### Public Class VertexClickEvent{...}

This class extends `MouseEvent`. A new filed is added to this class to remember the ID of each vertex in the graph visualization window.

**Public Abstract Class Vaccinate{...}**

This abstract class is designed to mark nodes of the graph as vaccinated in the two following different fashions:

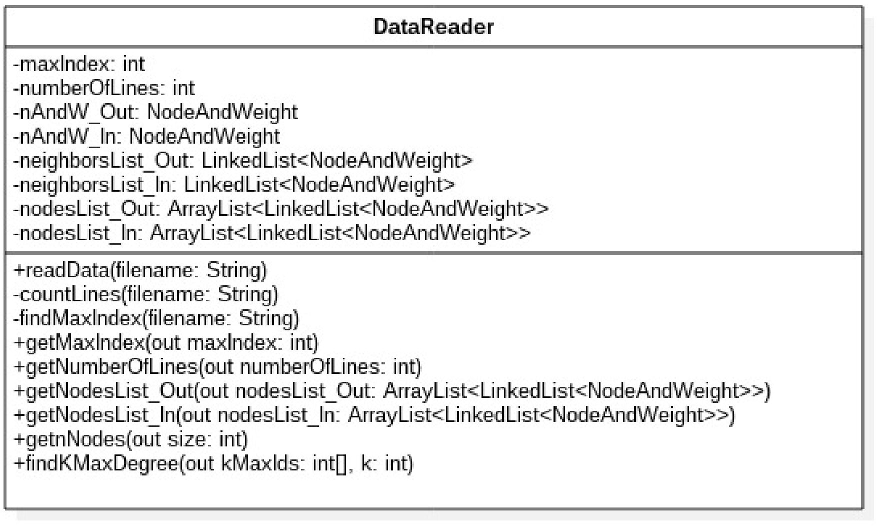
- Only those $k$ nodes which are discovered as the most influentials.
- Only those $k$ nodes which are discovered as the high-degree nodes.

**Remark 6.2.** As one may note, this class is not responsible for discovering the nodes which should be vaccinated, it only marks the already discovered nodes as vaccinated.

**Public Abstract Class VirusSpread{...}**

This class is tailored to spread the virus throughout the network with two different starting points: The initially infected nodes ($I_{Ini}$) are regarded once as the most influential nodes and another time as the high-degree nodes. In both experiments, the behavior of the virus is the same. However, because of the differently chosen $I_{Ini}$, it is expected to not result in the same set of the infected nodes in both of them. Its design is described in details in Section *Algorithms and Implementation*.

## 6.4. Class Diagrams



```
DataReader
-maxIndex: int
-numberOfLines: int
-nAndW_Out: NodeAndWeight
-nAndW_In: NodeAndWeight
-neighborsList_Out: LinkedList<NodeAndWeight>
-neighborsList_In: LinkedList<NodeAndWeight>
-nodesList_Out: ArrayList<LinkedList<NodeAndWeight>>
-nodesList_In: ArrayList<LinkedList<NodeAndWeight>>
+readData(filename: String)
-countLines(filename: String)
-findMaxIndex(filename: String)
+getMaxIndex(out maxIndex: int)
+getNumberOfLines(out numberOfLines: int)
+getNodesList_Out(out nodesList_Out: ArrayList<LinkedList<NodeAndWeight>>)
+getNodesList_In(out nodesList_In: ArrayList<LinkedList<NodeAndWeight>>)
+getnNodes(out size: int)
+findKMaxDegree(out kMaxIds: int[], k: int)
```

Figure 6.4: Class: DataReader

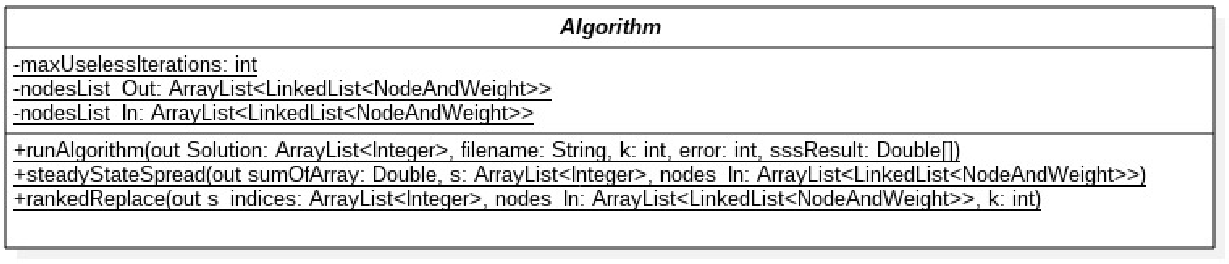

Algorithm
-maxUselessIterations: int
-nodesList_Out: ArrayList<LinkedList<NodeAndWeight>>
-nodesList_In: ArrayList<LinkedList<NodeAndWeight>>
+runAlgorithm(out Solution: ArrayList<Integer>, filename: String, k: int, error: int, sssResult: Double[])
+steadyStateSpread(out sumOfArray: Double, s: ArrayList<Integer>, nodes_In: ArrayList<LinkedList<NodeAndWeight>>)
+rankedReplace(out s_indices: ArrayList<Integer>, nodes_In: ArrayList<LinkedList<NodeAndWeight>>, k: int)

Figure 6.5: Class: Algorithm

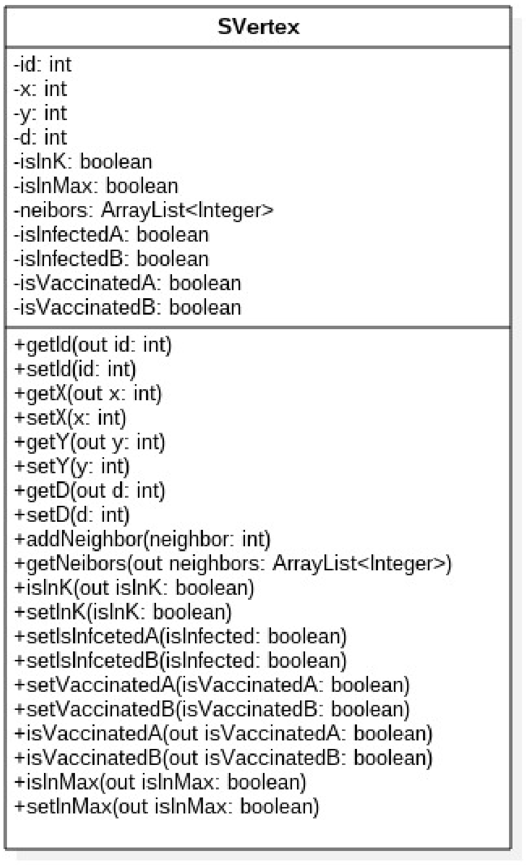

SVertex
-id: int
-x: int
-y: int
-d: int
-isInK: boolean
-isInMax: boolean
-neibors: ArrayList<Integer>
-isInfectedA: boolean
-isInfectedB: boolean
-isVaccinatedA: boolean
-isVaccinatedB: boolean
+getId(out id: int)
+setId(id: int)
+getX(out x: int)
+setX(x: int)
+getY(out y: int)
+setY(y: int)
+getD(out d: int)
+setD(d: int)
+addNeighbor(neighbor: int)
+getNeibors(out neighbors: ArrayList<Integer>)
+isInK(out isInK: boolean)
+setInK(isInK: boolean)
+setIsInfcetedA(isInfected: boolean)
+setIsInfcetedB(isInfected: boolean)
+setVaccinatedA(isVaccinatedA: boolean)
+setVaccinatedB(isVaccinatedB: boolean)
+isVaccinatedA(out isVaccinatedA: boolean)
+isVaccinatedB(out isVaccinatedB: boolean)
+isInMax(out isInMax: boolean)
+setInMax(out isInMax: boolean)

Figure 6.6: Class: SVertex

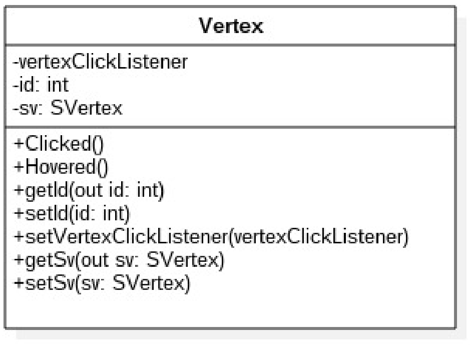


Figure 6.7: Class: Vertex

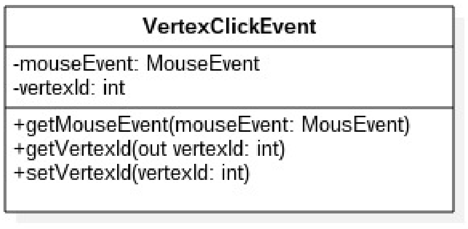

VertexClickEvent
-mouseEvent: MouseEvent
-vertexId: int
+getMouseEvent(mouseEvent: MousEvent)
+getVertexId(out vertexId: int)
+setVertexId(vertexId: int)

Figure 6.8: Class: VertexClickEvent

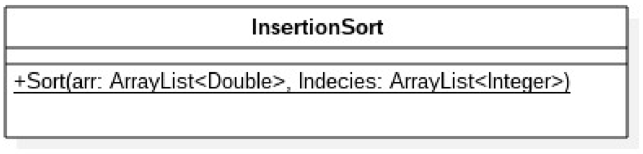


Figure 6.9: Class: InsertionSort

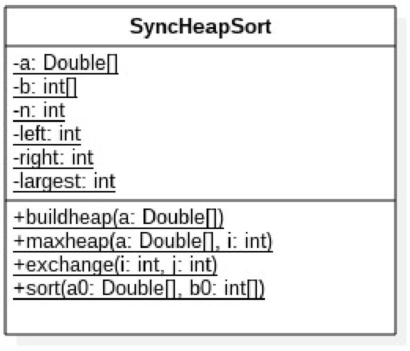


Figure 6.10: Class: SyncHeapSort

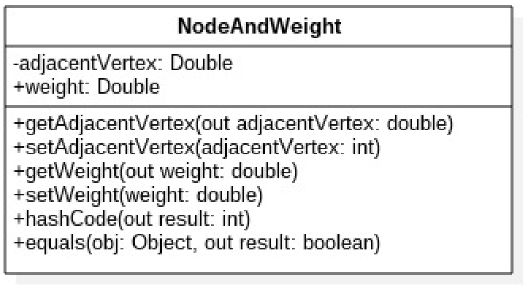

NodeAndWeight
-adjacentVertex: Double
+weight: Double
+getAdjacentVertex(out adjacentVertex: double)
+setAdjacentVertex(adjacentVertex: int)
+getWeight(out weight: double)
+setWeight(weight: double)
+hashCode(out result: int)
+equals(obj: Object, out result: boolean)

Figure 6.11: Class: NodeAndWeight

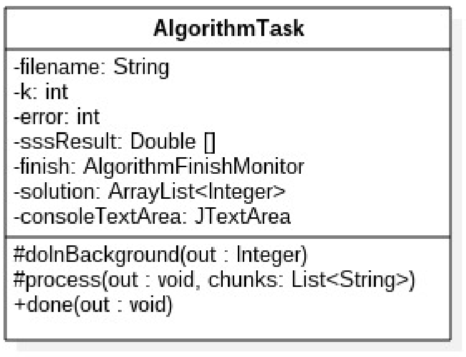
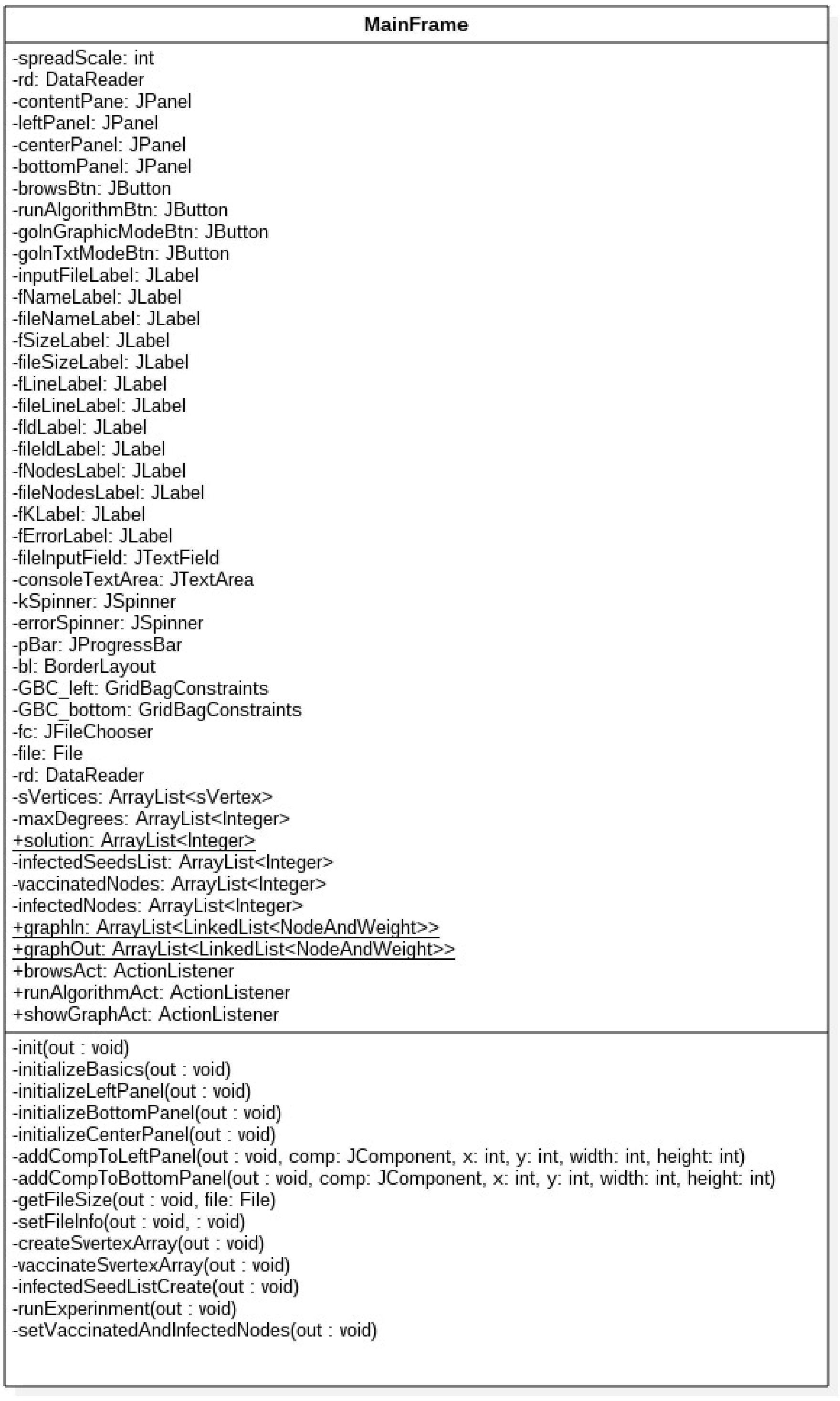

AlgorithmTask
-filename: String
-k: int
-error: int
-sssResult: Double []
-finish: AlgorithmFinishMonitor
-solution: ArrayList<Integer>
-consoleTextArea: JTextArea
#doInBackground(out : Integer)
#process(out : void, chunks: List<String>)
+done(out : void)

Figure 6.12: Class: AlgorithmTask



**MainFrame**

-spreadScale: int
-rd: DataReader
-contentPane: JPanel
-leftPanel: JPanel
-centerPanel: JPanel
-bottomPanel: JPanel
-browsBtn: JButton
-runAlgorithmBtn: JButton
-goInGraphicModeBtn: JButton
-goInTxtModeBtn: JButton
-inputFileLabel: JLabel
-fNameLabel: JLabel
-fileNameLabel: JLabel
-fSizeLabel: JLabel
-fileSizeLabel: JLabel
-fLineLabel: JLabel
-fileLineLabel: JLabel
-fIdLabel: JLabel
-fileIdLabel: JLabel
-fNodesLabel: JLabel
-fileNodesLabel: JLabel
-fKLabel: JLabel
-fErrorLabel: JLabel
-fileInputField: JTextField
-consoleTextArea: JTextArea
-kSpinner: JSpinner
-errorSpinner: JSpinner
-pBar: JProgressBar
-bl: BorderLayout
-GBC_left: GridBagConstraints
-GBC_bottom: GridBagConstraints
-fc: JFileChooser
-file: File
-rd: DataReader
-sVertices: ArrayList<sVertex>
-maxDegrees: ArrayList<Integer>
+solution: ArrayList<Integer>
-infectedSeedsList: ArrayList<Integer>
-vaccinatedNodes: ArrayList<Integer>
-infectedNodes: ArrayList<Integer>
+graphIn: ArrayList<LinkedList<NodeAndWeight>>
+graphOut: ArrayList<LinkedList<NodeAndWeight>>
+browsAct: ActionListener
+runAlgorithmAct: ActionListener
+showGraphAct: ActionListener

-init(out : void)
-initializeBasics(out : void)
-initializeLeftPanel(out : void)
-initializeBottomPanel(out : void)
-initializeCenterPanel(out : void)
-addCompToLeftPanel(out : void, comp: JComponent, x: int, y: int, width: int, height: int)
-addCompToBottomPanel(out : void, comp: JComponent, x: int, y: int, width: int, height: int)
-getFileSize(out : void, file: File)
-setFileInfo(out : void, : void)
-createSvertexArray(out : void)
-vaccinateSvertexArray(out : void)
-infectedSeedListCreate(out : void)
-runExperinment(out : void)
-setVaccinatedAndInfectedNodes(out : void)

Figure 6.13: Class: MainFrame

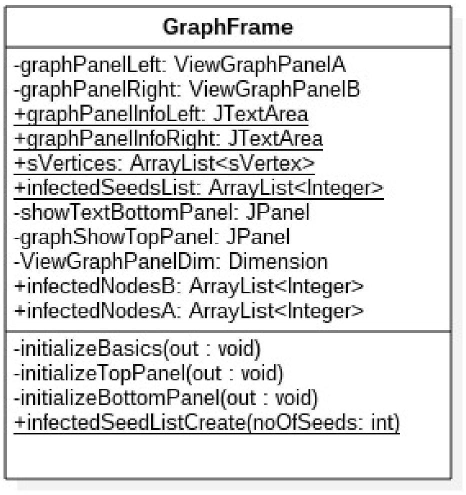

GraphFrame
-graphPanelLeft: ViewGraphPanelA
-graphPanelRight: ViewGraphPanelB
+graphPanelInfoLeft: JTextArea
+graphPanelInfoRight: JTextArea
+sVertices: ArrayList<sVertex>
+infectedSeedsList: ArrayList<Integer>
-showTextBottomPanel: JPanel
-graphShowTopPanel: JPanel
-ViewGraphPanelDim: Dimension
+infectedNodesB: ArrayList<Integer>
+infectedNodesA: ArrayList<Integer>
-initializeBasics(out : void)
-initializeTopPanel(out : void)
-initializeBottomPanel(out : void)
+infectedSeedListCreate(noOfSeeds: int)

Figure 6.14: Class: GraphFrame

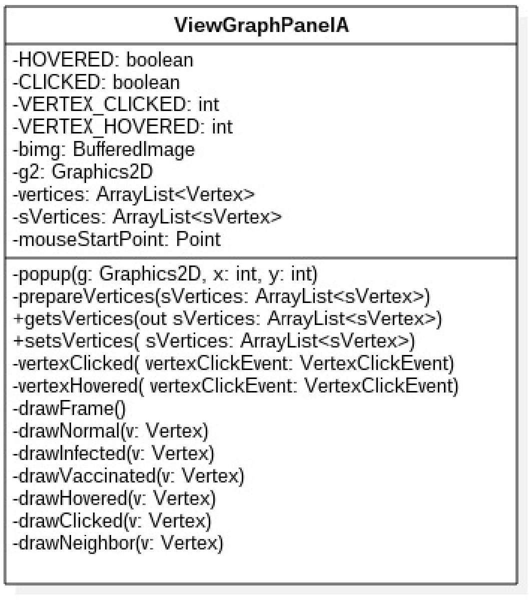

ViewGraphPanelA
-HOVERED: boolean
-CLICKED: boolean
-VERTEX_CLICKED: int
-VERTEX_HOVERED: int
-bimg: BufferedImage
-g2: Graphics2D
-vertices: ArrayList<Vertex>
-sVertices: ArrayList<sVertex>
-mouseStartPoint: Point
-popup(g: Graphics2D, x: int, y: int)
-prepareVertices(sVertices: ArrayList<sVertex>)
+getsVertices(out sVertices: ArrayList<sVertex>)
+setsVertices( sVertices: ArrayList<sVertex>)
-vertexClicked( vertexClickEvent: VertexClickEvent)
-vertexHovered( vertexClickEvent: VertexClickEvent)
-drawFrame()
-drawNormal(v: Vertex)
-drawInfected(v: Vertex)
-drawVaccinated(v: Vertex)
-drawHovered(v: Vertex)
-drawClicked(v: Vertex)
-drawNeighbor(v: Vertex)

Figure 6.15: Class: ViewGraphPanelA

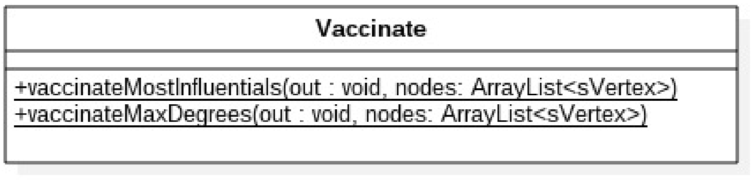

Vaccinate
+vaccinateMostInfluentials(out : void, nodes: ArrayList<sVertex>)
+vaccinateMaxDegrees(out : void, nodes: ArrayList<sVertex>)

Figure 6.16: Class: Vaccinate

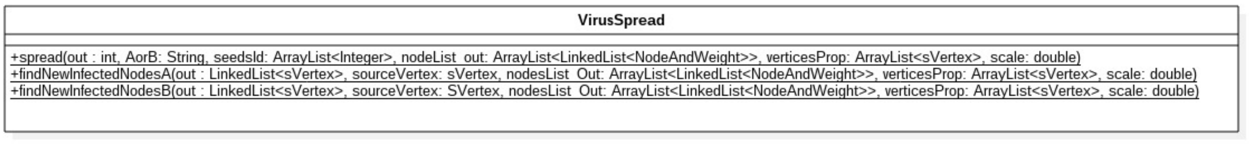

VirusSpread
+spread(out : int, AorB: String, seedsId: ArrayList<Integer>, nodeList_out: ArrayList<LinkedList<NodeAndWeight>>, verticesProp: ArrayList<sVertex>, scale: double)
+findNewInfectedNodesA(out : LinkedList<sVertex>, sourceVertex: sVertex, nodesList_Out: ArrayList<LinkedList<NodeAndWeight>>, verticesProp: ArrayList<sVertex>, scale: double)
+findNewInfectedNodesB(out : LinkedList<sVertex>, sourceVertex: SVertex, nodesList_Out: ArrayList<LinkedList<NodeAndWeight>>, verticesProp: ArrayList<sVertex>, scale: double)

Figure 6.17: Class: VirusSpread

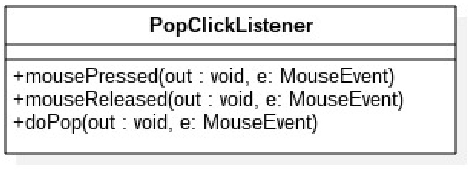

PopClickListener
+mousePressed(out : void, e: MouseEvent)
+mouseReleased(out : void, e: MouseEvent)
+doPop(out : void, e: MouseEvent)

Figure 6.18: Class: PopClickListener

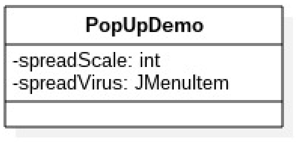


Figure 6.19: Class: PopUpDemo

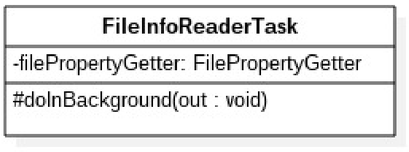

FileInfoReaderTask
-filePropertyGetter: FilePropertyGetter
#doInBackground(out : void)

Figure 6.20: Class: FileInfoReaderTask

# Appendix A

# Experiments Results

The details of the architecture of the experiment as well as the experiments' results are described in Chapter 4. In this appendix, we provide all the information regarding the **US Airports** experiments with more details.

## A.1. General Description

Here in figures A.1 and A.2, each pie chart is the result of 20 runs of the experiment with constant values for $K$ and $S$, where:

- **K** is the required number of most influential nodes which is equal to the required number of high-degree nodes, and
- **S** is the number of nodes which will be initially infected by the virus in the procedure of the spread of the virus in both societies A and B.

Since each experiment compares the two societies A and B, a pie chart is used to illustrate the relative frequency of the society with the higher infection rate. As it was previously explained in Chapter 4:

- **A** represents the society (network) in which the $k$ most influential nodes are vaccinated,
- **B** represents the society (network) in which the $k$ high-degree nodes are vaccinated, and
- **E** represents the situations in which A and B are equally infected.

In every row of these tables of pie charts, $K$ remains the same while $S$ varies. The purpose of such a sequence of experiments is to see if there exists any pattern in which either A or B always has the higher infection rate.

After these pie charts, details of each experiment are presented in tables and diagrams. In each table, there are 4 columns where:

- **N** keeps track of the experiment number,
- [**A**] shows the number of nodes which are eventually infected in the society A,
- [**B**] shows the number of nodes which are eventually infected in the society B,

- [**A V B**] shows the society with the highest infection rate after the virus spread, in a particular run.

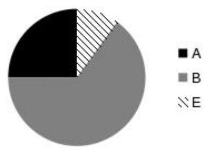
We carried out experiments on different data sets and since they all resulted in the same behavior, we provide the details of all experiments on only one data set. This data set represents a network of all the US Airports, originally provided in *Vladimir Batagelj*'s web page under the following link:

`http://vlado.fmf.uni-lj.si/pub/networks/data/`.

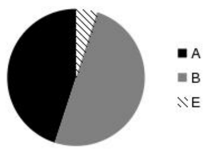
The network is weighted and directed, and contains 332 verticies, each of which represents one United States' airlines.

## A.2. Airlines: Data-set With Default Weights

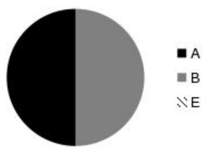
Figure A.1: Relative frequency of the experiment (A or B) with higher infection rate.
Data-set: US Airports
Total Nodes: 332
Default weight of the graph is maintained.



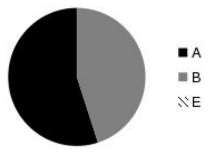
(a) K = 5, S = 5



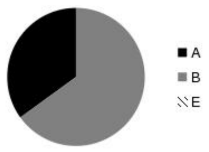
(b) K = 5, S = 10



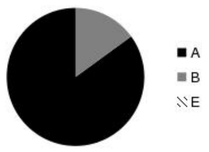
(c) K = 5, S = 20



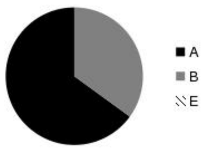
(d) K = 10, S = 5



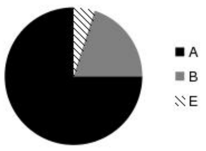
(e) K = 10, S = 10



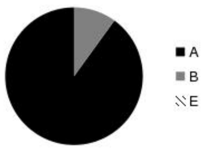
(f) K = 10, S = 20



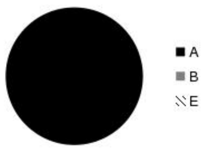
(g) K = 20, S = 5



(h) K = 20, S = 10



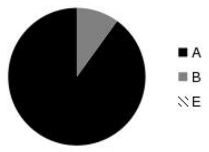
(i) K = 20, S = 20



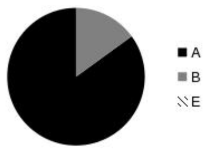
(j) K = 20, S = 30

## A.3. Airlines: Data-set With 5 X Default Weights

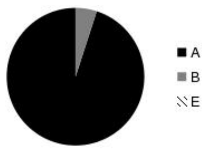
Figure A.2: Relative frequency of the experiment (A or B) with higher infection rate.
Data-set: US Airports
Total Nodes: 332
Weights of the graph is 5 times greater than its original.



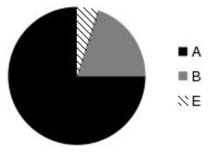
(a) K = 5, S = 5



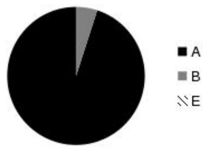
(b) K = 5, S = 10



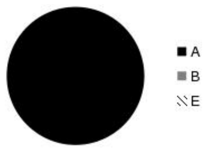
(c) K = 5, S = 20



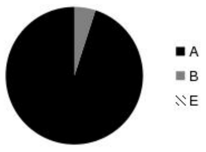
(d) K = 10, S = 5



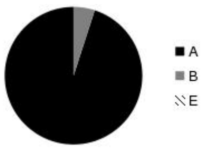
(e) K = 10, S = 10



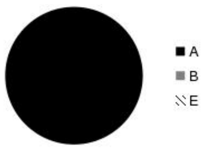
(f) K = 10, S = 20



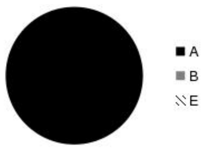
(g) K = 20, S = 5



(h) K = 20, S = 10



(i) K = 20, S = 20



(j) K = 20, S = 30

## A.4. Airlines: Details of Experiments for Default Weights

Table A.1:
`K = 5, S = 5`
$SSS$`(A):48.55`
$SSS$`(B):40.57`

| N | [A] | [B] | [A V B] |
|---|---|---|---|
| 1 | 11 | 39 | B |
| 2 | 10 | 21 | B |
| 3 | 37 | 37 | E |
| 4 | 24 | 38 | B |
| 5 | 49 | 78 | B |
| 6 | 17 | 39 | B |
| 7 | 28 | 5 | A |
| 8 | 29 | 41 | B |
| 9 | 24 | 43 | B |
| 10 | 14 | 18 | B |
| 11 | 5 | 6 | B |
| 12 | 20 | 18 | A |
| 13 | 6 | 36 | B |
| 14 | 26 | 49 | B |
| 15 | 5 | 32 | B |
| 16 | 6 | 5 | A |
| 17 | 10 | 5 | A |
| 18 | 11 | 38 | B |
| 19 | 6 | 6 | E |
| 20 | 25 | 5 | A |

Figure A.3: Number of infected nodes in each experiment after the spread of the virus - Total number of nodes : 332

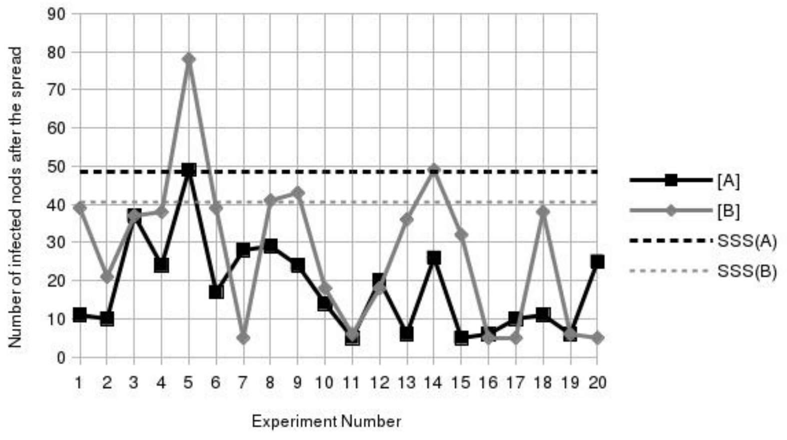


Table A.2:
`K = 5, S = 10`
$SSS$`(A):48.55`
$SSS$`(B):40.57`

| N | [A] | [B] | [A V B] |
|---|---|---|---|
| 1 | 15 | 18 | B |
| 2 | 10 | 11 | B |
| 3 | 35 | 29 | A |
| 4 | 39 | 40 | B |
| 5 | 30 | 30 | E |
| 6 | 44 | 55 | B |
| 7 | 53 | 30 | A |
| 8 | 42 | 35 | A |
| 9 | 44 | 47 | B |
| 10 | 17 | 26 | B |
| 11 | 64 | 51 | A |
| 12 | 24 | 21 | A |
| 13 | 19 | 72 | B |
| 14 | 58 | 24 | A |
| 15 | 25 | 20 | A |
| 16 | 45 | 32 | A |
| 17 | 14 | 25 | B |
| 18 | 30 | 48 | B |
| 19 | 56 | 44 | A |
| 20 | 59 | 14 | A |

Figure A.4: Number of infected nodes in each experiment after the spread of the virus - Total number of nodes : 332

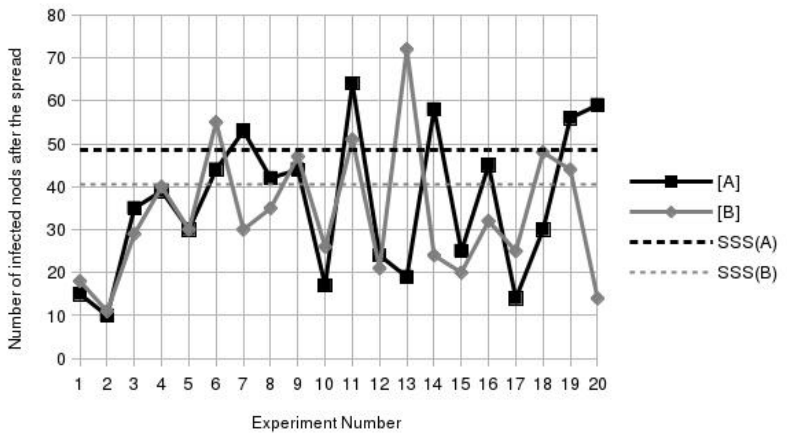


Table A.3:
`K = 5, S = 20`
$SSS$`(A):48.55`
$SSS$`(B):40.57`

| N | [A] | [B] | [A V B] |
|---|---|---|---|
| 1 | 39 | 40 | B |
| 2 | 64 | 61 | A |
| 3 | 71 | 75 | B |
| 4 | 73 | 63 | A |
| 5 | 63 | 45 | A |
| 6 | 71 | 66 | A |
| 7 | 49 | 57 | B |
| 8 | 64 | 61 | A |
| 9 | 35 | 43 | B |
| 10 | 82 | 47 | A |
| 11 | 48 | 29 | A |
| 12 | 49 | 76 | B |
| 13 | 66 | 68 | B |
| 14 | 55 | 60 | B |
| 15 | 70 | 71 | B |
| 16 | 69 | 73 | B |
| 17 | 64 | 60 | A |
| 18 | 60 | 62 | B |
| 19 | 61 | 49 | A |
| 20 | 80 | 67 | A |

Figure A.5: Number of infected nodes in each experiment after the spread of the virus - Total number of nodes : 332

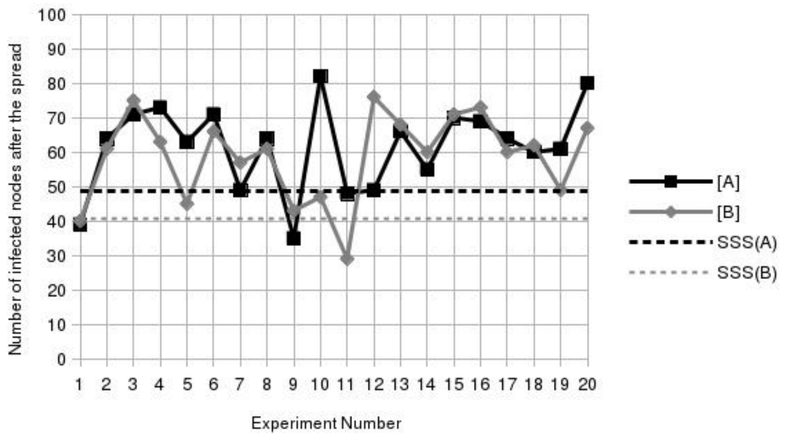


Table A.4:
`K = 10, S = 5`
$SSS$`(A):52.32`
$SSS$`(B):47.13`

| N | [A] | [B] | [A V B] |
|---|---|---|---|
| 1 | 27 | 16 | A |
| 2 | 18 | 30 | B |
| 3 | 15 | 28 | B |
| 4 | 5 | 30 | B |
| 5 | 14 | 5 | A |
| 6 | 41 | 40 | A |
| 7 | 6 | 32 | B |
| 8 | 6 | 5 | A |
| 9 | 41 | 17 | A |
| 10 | 25 | 8 | A |
| 11 | 24 | 5 | A |
| 12 | 32 | 34 | B |
| 13 | 5 | 6 | B |
| 14 | 38 | 6 | A |
| 15 | 21 | 29 | B |
| 16 | 17 | 5 | A |
| 17 | 42 | 41 | A |
| 18 | 44 | 5 | A |
| 19 | 5 | 10 | B |
| 20 | 38 | 43 | B |

Figure A.6: Number of infected nodes in each experiment after the spread of the virus - Total number of nodes : 332

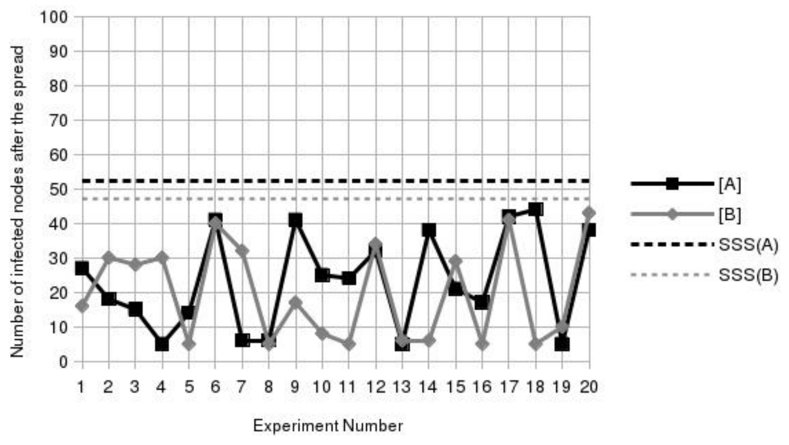


Table A.5:
`K = 10, S = 10`
$SSS$`(A):52.32`
$SSS$`(B):47.13`

| N | [A] | [B] | [A V B] |
|---|---|---|---|
| 1 | 10 | 11 | B |
| 2 | 21 | 17 | A |
| 3 | 23 | 39 | B |
| 4 | 32 | 45 | B |
| 5 | 26 | 23 | A |
| 6 | 14 | 11 | A |
| 7 | 25 | 16 | A |
| 8 | 41 | 47 | B |
| 9 | 58 | 16 | A |
| 10 | 34 | 28 | A |
| 11 | 40 | 41 | B |
| 12 | 15 | 49 | B |
| 13 | 16 | 22 | B |
| 14 | 14 | 28 | B |
| 15 | 25 | 38 | B |
| 16 | 49 | 53 | B |
| 17 | 28 | 26 | A |
| 18 | 14 | 35 | B |
| 19 | 12 | 45 | B |
| 20 | 14 | 17 | B |

Figure A.7: Number of infected nodes in each experiment after the spread of the virus - Total number of nodes : 332

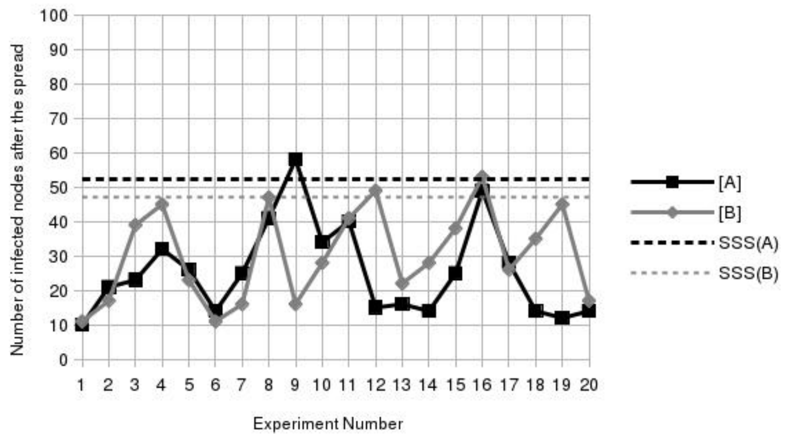


Table A.6:
`K = 10, S = 20`
$SSS$`(A):52.32`
$SSS$`(B):47.13`

| N | [A] | [B] | [A V B] |
|---|---|---|---|
| 1 | 50 | 48 | A |
| 2 | 54 | 72 | B |
| 3 | 60 | 52 | A |
| 4 | 68 | 59 | A |
| 5 | 59 | 37 | A |
| 6 | 51 | 46 | A |
| 7 | 52 | 45 | A |
| 8 | 47 | 54 | B |
| 9 | 65 | 41 | A |
| 10 | 59 | 53 | A |
| 11 | 81 | 36 | A |
| 12 | 51 | 21 | A |
| 13 | 29 | 24 | A |
| 14 | 33 | 24 | A |
| 15 | 58 | 45 | A |
| 16 | 52 | 50 | A |
| 17 | 46 | 53 | B |
| 18 | 68 | 52 | A |
| 19 | 84 | 71 | A |
| 20 | 73 | 50 | A |

Figure A.8: Number of infected nodes in each experiment after the spread of the virus - Total number of nodes : 332

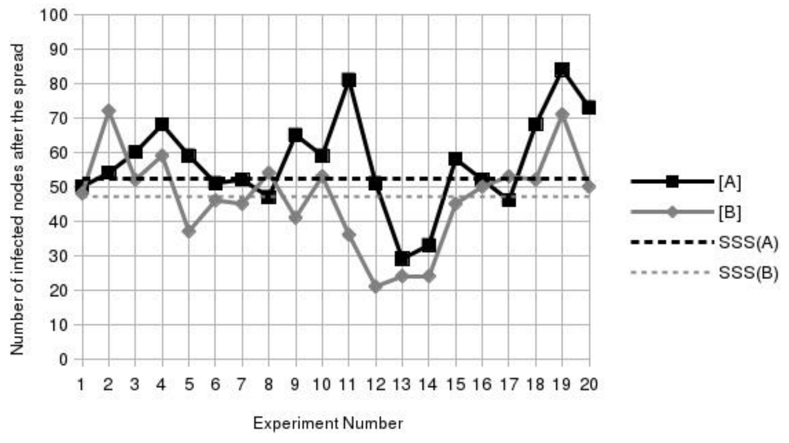


Table A.7:
`K = 20, S = 5`
*SSS*`(A):54.39`
*SSS*`(B):45.21`

| N | [A] | [B] | [A V B] |
|---|---|---|---|
| 1 | 46 | 14 | A |
| 2 | 27 | 30 | B |
| 3 | 6 | 8 | B |
| 4 | 13 | 5 | A |
| 5 | 8 | 5 | A |
| 6 | 18 | 5 | A |
| 7 | 5 | 6 | B |
| 8 | 6 | 5 | A |
| 9 | 5 | 10 | B |
| 10 | 16 | 8 | A |
| 11 | 6 | 12 | B |
| 12 | 5 | 11 | B |
| 13 | 30 | 11 | A |
| 14 | 6 | 5 | A |
| 15 | 13 | 12 | A |
| 16 | 6 | 5 | A |
| 17 | 51 | 5 | A |
| 18 | 35 | 5 | A |
| 19 | 40 | 7 | A |
| 20 | 5 | 12 | B |

Figure A.9: Number of infected nodes in each experiment after the spread of the virus - Total number of nodes : 332

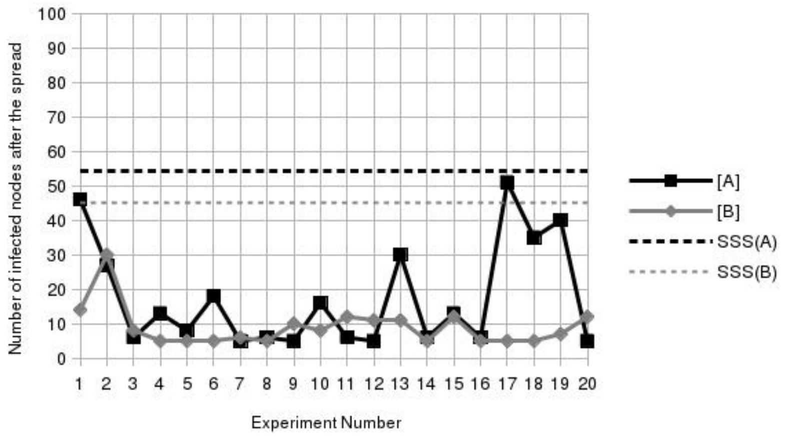


Table A.8:
`K = 20, S = 10`
*SSS*`(A):54.39`
*SSS*`(B):45.21`

| N | [A] | [B] | [A V B] |
|---|---|---|---|
| 1 | 28 | 11 | A |
| 2 | 27 | 17 | A |
| 3 | 49 | 32 | A |
| 4 | 33 | 18 | A |
| 5 | 19 | 20 | B |
| 6 | 25 | 12 | A |
| 7 | 54 | 10 | A |
| 8 | 27 | 18 | A |
| 9 | 38 | 18 | A |
| 10 | 57 | 11 | A |
| 11 | 16 | 21 | B |
| 12 | 16 | 17 | B |
| 13 | 37 | 27 | A |
| 14 | 10 | 11 | B |
| 15 | 61 | 27 | A |
| 16 | 24 | 24 | E |
| 17 | 32 | 19 | A |
| 18 | 59 | 24 | A |
| 19 | 35 | 20 | A |
| 20 | 31 | 10 | A |

Figure A.10: Number of infected nodes in each experiment after the spread of the virus - Total number of nodes : 332

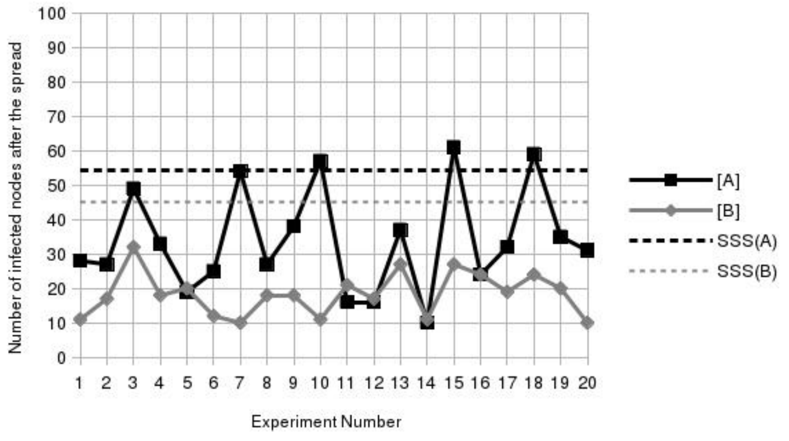


Table A.9:
`K = 20, S = 20`
*SSS*`(A):54.39`
*SSS*`(B):45.21`

| N | [A] | [B] | [A V B] |
|---|---|---|---|
| 1 | 66 | 32 | A |
| 2 | 41 | 24 | A |
| 3 | 35 | 39 | B |
| 4 | 58 | 25 | A |
| 5 | 64 | 35 | A |
| 6 | 43 | 20 | A |
| 7 | 63 | 22 | A |
| 8 | 62 | 34 | A |
| 9 | 59 | 35 | A |
| 10 | 63 | 33 | A |
| 11 | 43 | 38 | A |
| 12 | 65 | 29 | A |
| 13 | 25 | 28 | B |
| 14 | 42 | 27 | A |
| 15 | 50 | 26 | A |
| 16 | 37 | 34 | A |
| 17 | 70 | 28 | A |
| 18 | 26 | 25 | A |
| 19 | 46 | 33 | A |
| 20 | 40 | 36 | A |

Figure A.11: Number of infected nodes in each experiment after the spread of the virus - Total number of nodes : 332

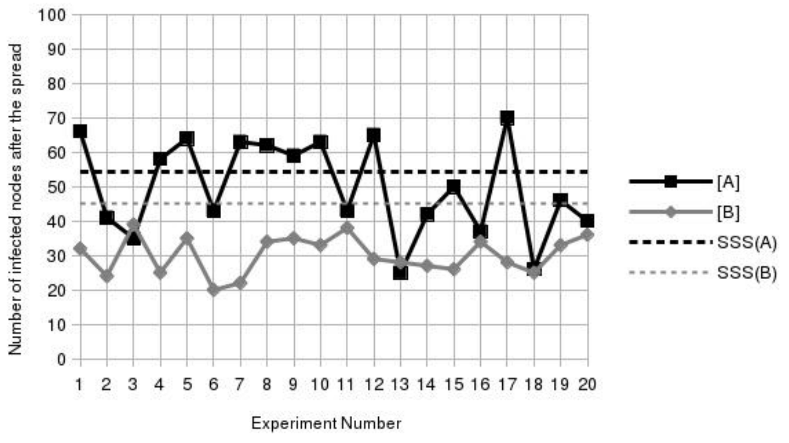
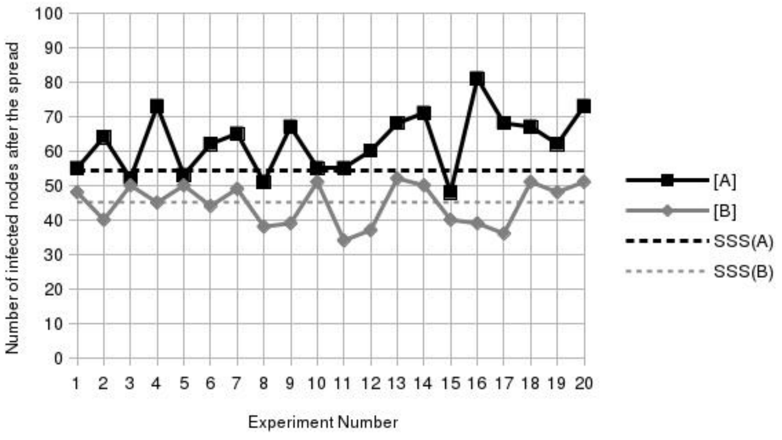


Table A.10:
`K = 20, S = 30`
$SSS$`(A):54.39`
$SSS$`(B):45.21`

| N | [A] | [B] | [A V B] |
|---|---|---|---|
| 1 | 55 | 48 | A |
| 2 | 64 | 40 | A |
| 3 | 52 | 50 | A |
| 4 | 73 | 45 | A |
| 5 | 53 | 50 | A |
| 6 | 62 | 44 | A |
| 7 | 65 | 49 | A |
| 8 | 51 | 38 | A |
| 9 | 67 | 39 | A |
| 10 | 55 | 51 | A |
| 11 | 55 | 34 | A |
| 12 | 60 | 37 | A |
| 13 | 68 | 52 | A |
| 14 | 71 | 50 | A |
| 15 | 48 | 40 | A |
| 16 | 81 | 39 | A |
| 17 | 68 | 36 | A |
| 18 | 67 | 51 | A |
| 19 | 62 | 48 | A |
| 20 | 73 | 51 | A |

Figure A.12: Number of infected nodes in each experiment after the spread of the virus - Total number of nodes : 332



## A.5. Airlines: Details of Experiments for 5 X Default Weights

Table A.11:
`K = 5, S = 5`
$SSS$`(A):128.79`
$SSS$`(B):106.15`

| N | [A] | [B] | [A V B] |
|---|---|---|---|
| 1 | 129 | 98 | A |
| 2 | 77 | 128 | B |
| 3 | 91 | 100 | B |
| 4 | 128 | 114 | A |
| 5 | 128 | 107 | A |
| 6 | 142 | 103 | A |
| 7 | 140 | 103 | A |
| 8 | 52 | 51 | A |
| 9 | 110 | 103 | A |
| 10 | 105 | 97 | A |
| 11 | 104 | 92 | A |
| 12 | 41 | 40 | A |
| 13 | 139 | 84 | A |
| 14 | 115 | 113 | A |
| 15 | 99 | 52 | A |
| 16 | 100 | 91 | A |
| 17 | 61 | 53 | A |
| 18 | 151 | 123 | A |
| 19 | 146 | 104 | A |
| 20 | 104 | 83 | A |

Figure A.13: Number of infected nodes in each experiment after spread of the virus - Total number of nodes : 332

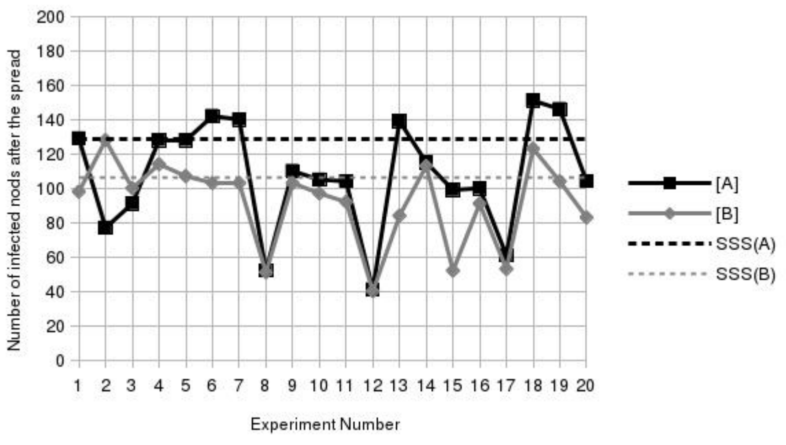


Table A.12:
`K = 5, S = 10`
$SSS$`(A):128.79`
$SSS$`(B):106.15`

| N | [A] | [B] | [A V B] |
|---|---|---|---|
| 1 | 143 | 78 | A |
| 2 | 157 | 97 | A |
| 3 | 113 | 112 | A |
| 4 | 58 | 55 | A |
| 5 | 82 | 131 | B |
| 6 | 131 | 110 | A |
| 7 | 157 | 124 | A |
| 8 | 116 | 98 | A |
| 9 | 150 | 127 | A |
| 10 | 161 | 125 | A |
| 11 | 138 | 103 | A |
| 12 | 136 | 112 | A |
| 13 | 148 | 102 | A |
| 14 | 146 | 112 | A |
| 15 | 159 | 133 | A |
| 16 | 149 | 122 | A |
| 17 | 154 | 133 | A |
| 18 | 77 | 82 | B |
| 19 | 68 | 97 | B |
| 20 | 146 | 93 | A |

Figure A.14: Number of infected nodes in each experiment after the spread of the virus - Total number of nodes : 332

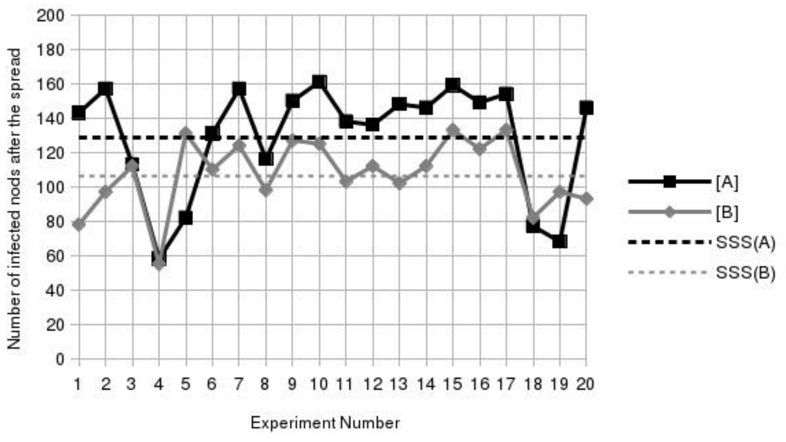


Table A.13:
`K = 5, S = 20`
$SSS$`(A):128.79`
$SSS$`(B):106.15`

| N | [A] | [B] | [A V B] |
|---|---|---|---|
| 1 | 152 | 124 | A |
| 2 | 163 | 127 | A |
| 3 | 159 | 148 | A |
| 4 | 154 | 112 | A |
| 5 | 139 | 119 | A |
| 6 | 159 | 141 | A |
| 7 | 162 | 126 | A |
| 8 | 141 | 121 | A |
| 9 | 149 | 141 | A |
| 10 | 130 | 145 | B |
| 11 | 165 | 145 | A |
| 12 | 141 | 125 | A |
| 13 | 136 | 115 | A |
| 14 | 116 | 107 | A |
| 15 | 158 | 125 | A |
| 16 | 154 | 115 | A |
| 17 | 126 | 117 | A |
| 18 | 148 | 119 | A |
| 19 | 109 | 71 | A |
| 20 | 155 | 151 | A |

Figure A.15: Number of infected nodes in each experiment after the spread of the virus - Total number of nodes : 332

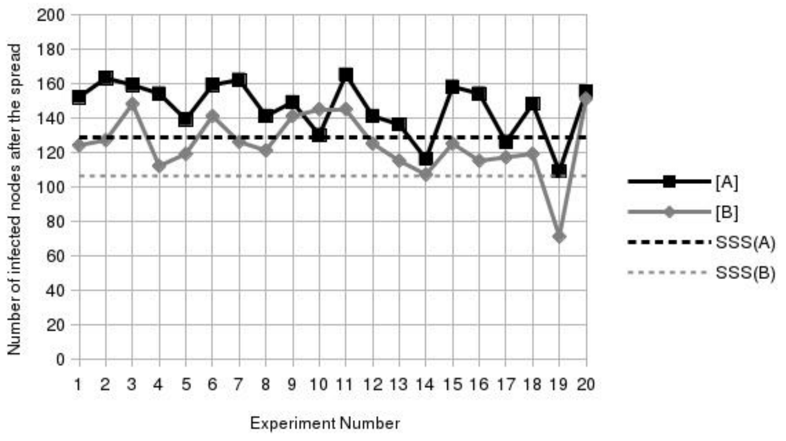


Table A.14:
`K = 10, S = 5`
$SSS$`(A):129.82`
$SSS$`(B):107.86`

| N | [A] | [B] | [A V B] |
|---|---|---|---|
| 1 | 164 | 8 | A |
| 2 | 154 | 50 | A |
| 3 | 159 | 88 | A |
| 4 | 157 | 73 | A |
| 5 | 25 | 48 | B |
| 6 | 125 | 63 | A |
| 7 | 139 | 91 | A |
| 8 | 60 | 51 | A |
| 9 | 79 | 61 | A |
| 10 | 109 | 87 | A |
| 11 | 63 | 63 | E |
| 12 | 51 | 29 | A |
| 13 | 168 | 108 | A |
| 14 | 138 | 78 | A |
| 15 | 69 | 76 | B |
| 16 | 72 | 106 | B |
| 17 | 139 | 81 | A |
| 18 | 103 | 68 | A |
| 19 | 58 | 71 | B |
| 20 | 133 | 82 | A |

Figure A.16: Number of infected nodes in each experiment after the spread of the virus - Total number of nodes : 332

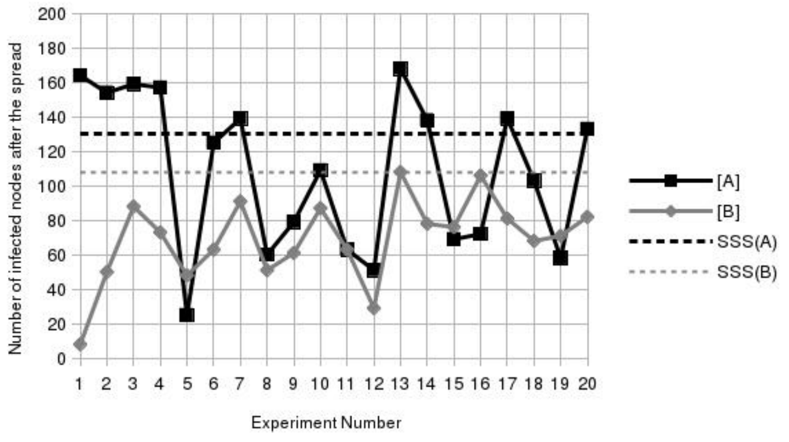


Table A.15:
`K = 10, S = 10`
$SSS$`(A):129.82`
$SSS$`(B):107.86`

| N | [A] | [B] | [A V B] |
|---|---|---|---|
| 1 | 151 | 100 | A |
| 2 | 139 | 79 | A |
| 3 | 156 | 108 | A |
| 4 | 153 | 75 | A |
| 5 | 101 | 102 | B |
| 6 | 91 | 72 | A |
| 7 | 154 | 71 | A |
| 8 | 138 | 103 | A |
| 9 | 134 | 67 | A |
| 10 | 152 | 104 | A |
| 11 | 147 | 105 | A |
| 12 | 147 | 88 | A |
| 13 | 114 | 75 | A |
| 14 | 99 | 77 | A |
| 15 | 44 | 28 | A |
| 16 | 162 | 112 | A |
| 17 | 141 | 60 | A |
| 18 | 105 | 49 | A |
| 19 | 138 | 69 | A |
| 20 | 140 | 87 | A |

Figure A.17: Number of infected nodes in each experiment after the spread of the virus - Total number of nodes : 332

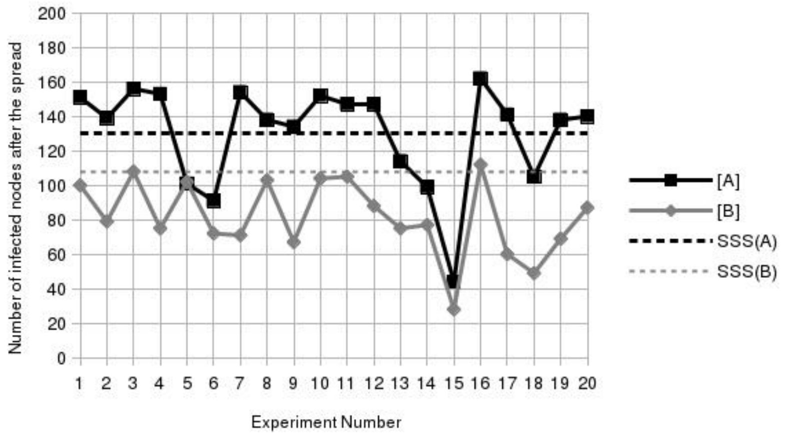


Table A.16:
`K = 10, S = 20`
$SSS$`(A):129.82`
$SSS$`(B):107.86`

| N | [A] | [B] | [A V B] |
|---|---|---|---|
| 1 | 113 | 58 | A |
| 2 | 168 | 119 | A |
| 3 | 113 | 92 | A |
| 4 | 144 | 96 | A |
| 5 | 155 | 90 | A |
| 6 | 141 | 102 | A |
| 7 | 170 | 111 | A |
| 8 | 145 | 99 | A |
| 9 | 165 | 68 | A |
| 10 | 152 | 126 | A |
| 11 | 141 | 89 | A |
| 12 | 124 | 104 | A |
| 13 | 162 | 100 | A |
| 14 | 147 | 86 | A |
| 15 | 142 | 103 | A |
| 16 | 167 | 112 | A |
| 17 | 162 | 100 | A |
| 18 | 156 | 119 | A |
| 19 | 146 | 109 | A |
| 20 | 158 | 88 | A |

Figure A.18: Number of infected nodes in each experiment after the spread of the virus - Total number of nodes : 332

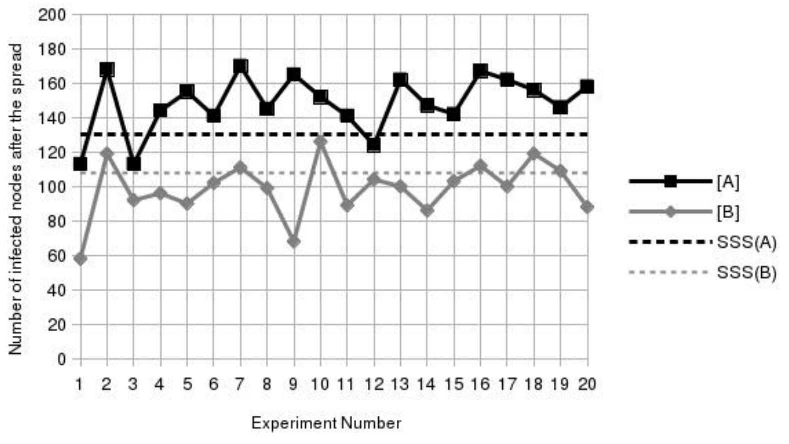


Table A.17:
`K = 20, S = 5`
*SSS*`(A):130.13`
*SSS*`(B):99.26`

| N | [A] | [B] | [A V B] |
|---|---|---|---|
| 1 | 126 | 39 | A |
| 2 | 88 | 70 | A |
| 3 | 148 | 44 | A |
| 4 | 136 | 34 | A |
| 5 | 109 | 47 | A |
| 6 | 53 | 39 | A |
| 7 | 147 | 14 | A |
| 8 | 20 | 16 | A |
| 9 | 90 | 54 | A |
| 10 | 130 | 52 | A |
| 11 | 100 | 52 | A |
| 12 | 35 | 6 | A |
| 13 | 150 | 40 | A |
| 14 | 133 | 53 | A |
| 15 | 68 | 48 | A |
| 16 | 11 | 14 | B |
| 17 | 104 | 29 | A |
| 18 | 142 | 52 | A |
| 19 | 67 | 40 | A |
| 20 | 114 | 5 | A |

Figure A.19: Number of infected nodes in each experiment after the spread of the virus - Total number of nodes : 332

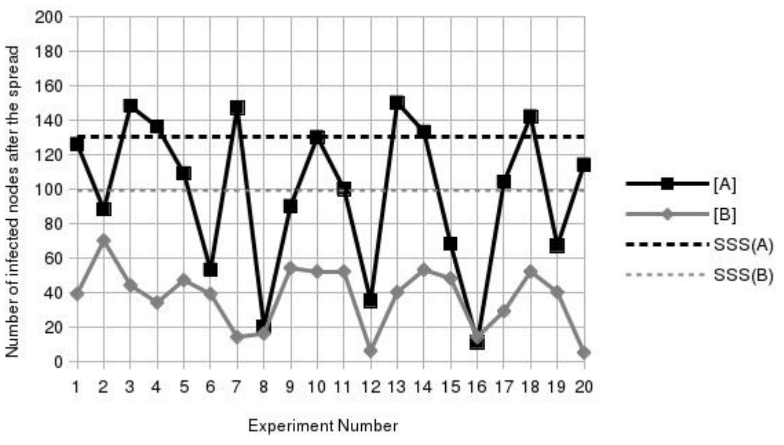


Table A.18:
`K = 20, S = 10`
*SSS*`(A):130.13`
*SSS*`(B):99.26`

| N | [A] | [B] | [A V B] |
|---|---|---|---|
| 1 | 150 | 32 | A |
| 2 | 121 | 59 | A |
| 3 | 85 | 41 | A |
| 4 | 119 | 60 | A |
| 5 | 11 | 12 | B |
| 6 | 135 | 52 | A |
| 7 | 135 | 44 | A |
| 8 | 146 | 44 | A |
| 9 | 149 | 51 | A |
| 10 | 143 | 67 | A |
| 11 | 45 | 26 | A |
| 12 | 139 | 54 | A |
| 13 | 135 | 45 | A |
| 14 | 168 | 67 | A |
| 15 | 110 | 134 | A |
| 16 | 165 | 52 | A |
| 17 | 143 | 55 | A |
| 18 | 145 | 59 | A |
| 19 | 141 | 48 | A |
| 20 | 92 | 64 | A |

Figure A.20: Number of infected nodes in each experiment after the spread of the virus - Total number of nodes : 332

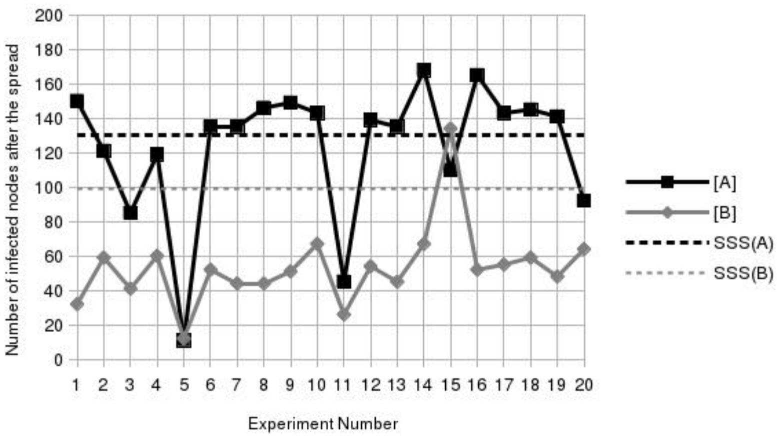


Table A.19:
`K = 20, S = 20`
*SSS*`(A):130.13`
*SSS*`(B):99.26`

| N | [A] | [B] | [A V B] |
|---|---|---|---|
| 1 | 143 | 78 | A |
| 2 | 160 | 61 | A |
| 3 | 165 | 69 | A |
| 4 | 161 | 78 | A |
| 5 | 144 | 70 | A |
| 6 | 158 | 76 | A |
| 7 | 157 | 77 | A |
| 8 | 166 | 77 | A |
| 9 | 124 | 76 | A |
| 10 | 161 | 63 | A |
| 11 | 150 | 38 | A |
| 12 | 147 | 72 | A |
| 13 | 106 | 42 | A |
| 14 | 161 | 74 | A |
| 15 | 157 | 66 | A |
| 16 | 138 | 83 | A |
| 17 | 167 | 81 | A |
| 18 | 133 | 68 | A |
| 19 | 171 | 79 | A |
| 20 | 157 | 65 | A |

Figure A.21: Number of infected nodes in each experiment after the spread of the virus - Total number of nodes : 332

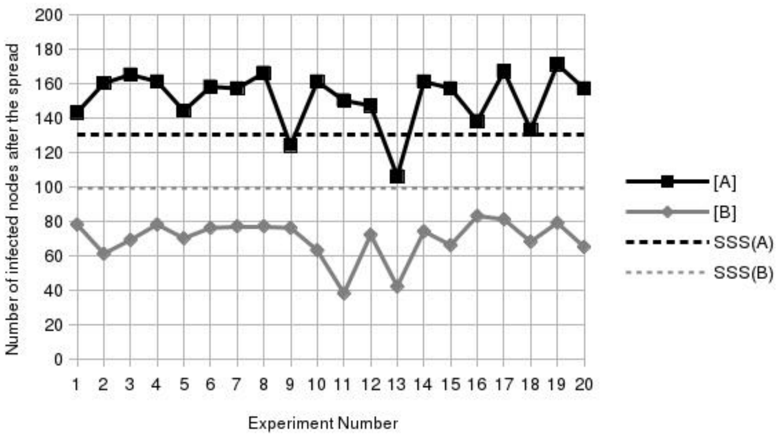
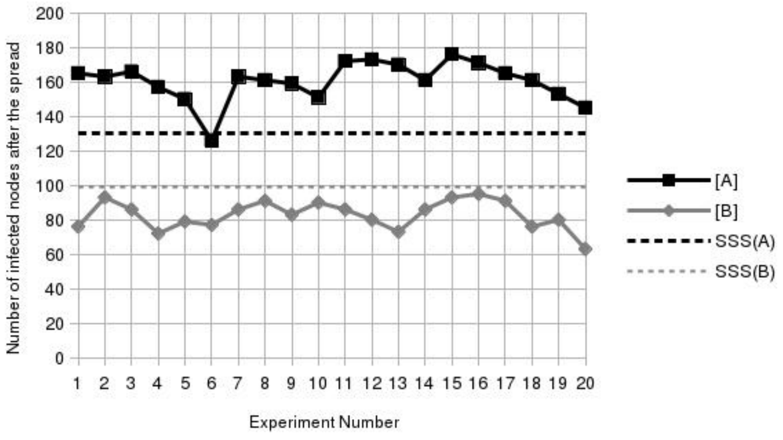


Table A.20:
`K = 20, S = 30`
$SSS$`(A):130.13`
$SSS$`(B):99.26`

| N | [A] | [B] | [A V B] |
|---|---|---|---|
| 1 | 165 | 76 | A |
| 2 | 163 | 93 | A |
| 3 | 166 | 86 | A |
| 4 | 157 | 72 | A |
| 5 | 150 | 79 | A |
| 6 | 126 | 77 | A |
| 7 | 163 | 86 | A |
| 8 | 161 | 91 | A |
| 9 | 159 | 83 | A |
| 10 | 151 | 90 | A |
| 11 | 172 | 86 | A |
| 12 | 173 | 80 | A |
| 13 | 170 | 73 | A |
| 14 | 161 | 86 | A |
| 15 | 176 | 93 | A |
| 16 | 171 | 95 | A |
| 17 | 165 | 91 | A |
| 18 | 161 | 76 | A |
| 19 | 153 | 80 | A |
| 20 | 145 | 63 | A |

Figure A.22: Number of infected nodes in each experiment after the spread of the virus - Total number of nodes : 332



Sajjad Hassanpour
Nr albumu 265728

Warszawa, 26th May 2016

## Statement of the author of thesis
*Oświadczenie autora pracy dyplomowej*

Being aware of my legal responsibility, I certify that this diploma:
*Świadom odpowiedzialności prawnej oświadczam, że przedstawiona praca dyplomowa:*

- has been written by me alone and does not contain any content obtained in a manner
  *została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób*
  inconsistent with the applicable rules,
  *niezgodny z obowiązującymi przepisami,*

- had not been previously subject to the procedures for obtaining professional title or
  *nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego*
  degree at the university.
  *lub stopnia naukowego w wyższej uczelni.*

Furthermore I declare that this version of the diploma thesis is identical with the electronic
*Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z wersją elektroniczną*
version attached.
*w załączeniu.*

........................................
Sajjad Hassanpour

# Bibliography

[1] Charu C. Aggarwal, Arijit Khan and Xifeng Yan
*"On Flow Authority Discovery in Social Networks"*
http://www.cs.ucsb.edu/ arijitkhan/Papers/gflow.pdf

[2] Aristides Gionis, Evimaria Terzi, and Panayiotis Tsaparas
*"Opinion maximization in social networks"*
http://cs-people.bu.edu/evimaria/papers/opinion-maximization.pdf

[3] Jon Kleinberg
*"Hubs, Authorities, and Communities",* 1999
http://cs.brown.edu/memex/ACM_HypertextTestbed/papers/10.html

[4] Chi Wang, Wei Chen, Yajun Wang
*"Scalable influence maximization for independent cascade model in large-scale social networks",* 2012

[5] Allan Borodin, Yuval Filmus, and Joel Oren *"Threshold Models for Competitive Influence in Social Networks", 2010*
http://www.cs.toronto.edu/ oren/Joel_Oren/About_Me_files/doc_1.pdf

[6] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze
*"Introduction to Information Retrieval",* 2008
http://nlp.stanford.edu/IR-book/html/htmledition/hubs-and-authorities-1.html

[7] *Insertion Sort*
https://en.wikipedia.org/wiki/Insertion_sort

[8] *Insertion Sort*
http://www.sorting-algorithms.com/insertion-sort

[9] *The NCSU Finite Math Book*, by *Lavon B. Page, Sandra O. Paur*
http://www.amazon.com/NCSU-finite-math-book/dp/9912293223