POLITECNICO DI TORINO

POLITECNICO DI TORINO

Master Degree course in Communications and Computer Networks Engineering

Master Degree Thesis

Simulations of Linear Threshold Model over large graphs.

Supervisor
Prof. Emilio Leonardi

Candidate
Davide Balzano

Academic Year 2017-2018

Acknowledgements

I would like to thank my professor Emilio Leonardi for the possibility to work on this experimental thesis. A special thank to my friend and colleague German that helped me on some C++ issues and for the support. Last but not least thank to the total support of my friends and my parents that gave me the possibility to study at Politecnico di Torino.

Abstract

Complex networks are recent case of study because the growing of Internet and the online social platforms (e.g. Instagram, Facebook, LinkedIn) that together with the always-on connectivity and availability of the mobile, has created the possibility of a rapid dissemination of information. In this work the interest is on epidemic processes on very large networks in the form of databases that have real and irregular structures. Hence with a well-known activation process called *Bootstrap percolation* we can understand the process of diffusion. Bootstrap percolation consists on activate a node only if it has at least r active neighbours. The purpose of this thesis is so to create a C++ program which can read common databases, load informations in a graph structure and then test bootstrap percolation and some variants on it (all done as new). Specifically are done simulations with bootstrap percolation starting from initial single and double seed of infection. Then we move on a modified version of the program that permits actions like disinfection and re-infection in the case of double initial seed. The main goal is to see the average trend of infections changing the starting point, so the initial seed (low, high, single, double and equal, etc.) and changing internal function parameters (threshold, neighbour threshold, limit in changes of infection, number of nodes, etc.). All these results are obtained experimentally and are very variable from simulation to simulation. We have tried to search not one shot behaviours but the common ones using different databases for the same simulation (also a synthetic one) to show that are really common results when we talk about Scale-free networks.

Contents

Chapter 1

Introduction

1.1 Complex networks

In recent years the research community has accumulated a large set of evidence for the emergence of complex and heterogeneous connectivity patterns in a wide range of biological and socio-technical systems. This situations can be described by dynamical processes, so an evolution in time of a network, to simulate many fundamental phenomena occurring in a various kind of complex systems like biological networks (e.g., neural, ecological, biochemical), technological networks (e.g., transportation, communication, energy) and social networks (e.g., group of people, Internet). A network is a set of nodes connected by links G=(V,E) and a very general and abstract representation is shown in figure 1.1.

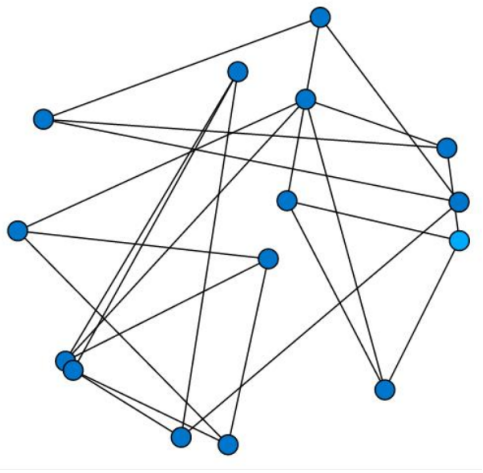
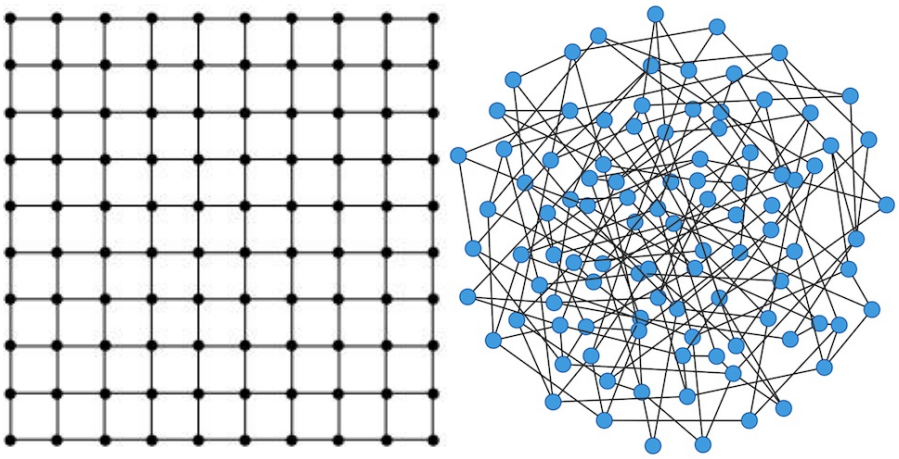


Figure 1.1: Graphical representation of a generic network.

For this reasons the study of complex networks become relevant, so networks with no-trivial topological features that do not occur in simple networks like lattices

(regular structure, grid) but often occur in graph modelling of real systems. To understand a graphical comparison is shown in figure 1.2.



(a) Lattice network (grid). (b) Complex network.

Figure 1.2: Graphical comparison between lattice and complex network.

Such features include a heavy tail in the degree distribution, a high clustering coefficient, assortativity or disassortativity among vertices, community structure, and hierarchical structure. Now are explained better all these characteristics. The degree tell us the number of links connected to a node in a certain network, and the degree distribution is the probability that a node selected uniformly at random has a certain number of links. The heavy tail distribution is a characteristic that brings to a tail heavier than the exponential, so we have value of degree far from the mean of the distribution with non-zero probability (higher degree). Better explanation looking the figure 1.3.

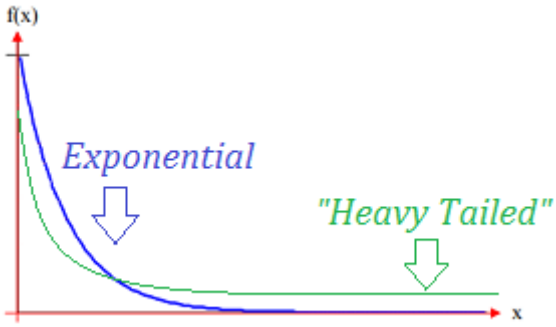
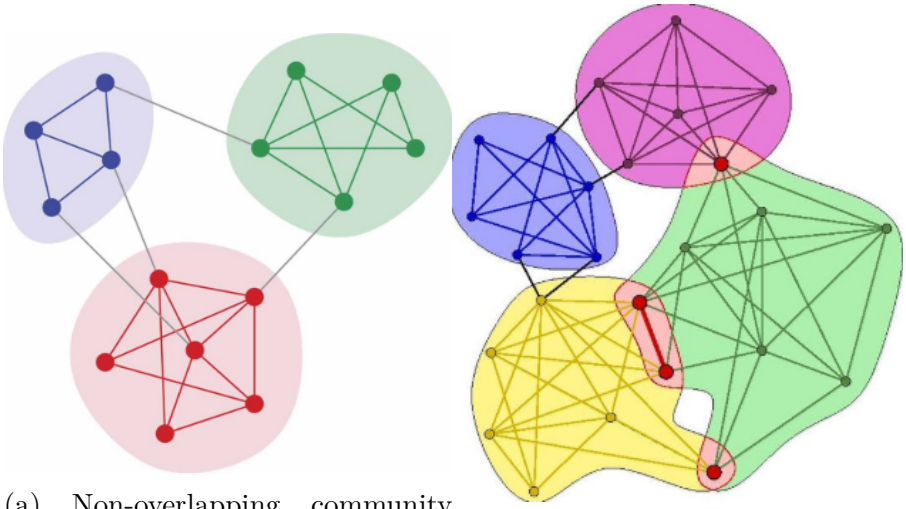


Figure 1.3: Comparison between exponential and heavy tailed.

A real example is visible looking the figure 1.5 in which a little group of nodes have a very high degree. The clustering coefficient is a measure of degree to which

nodes tends to cluster together. In our case the experiments are done on social network and the evidence tells us that nodes tend to create tightly knit groups, so high density of ties that brings to high clustering coefficient. Assortativity is the trend of nodes to attach to similar ones, similar in some way (can be with respect to the degree), and disassortativity means the contrary. It can happen to have instead community structure, so internally to a complex network nodes can be easily grouped in sets densely connected. Two possible case are non-overlapping community in which we have sparse connections between formed groups, and overlapping community in which this characteristic is not present. Example in figure 1.4.



(a) Non-overlapping community structure.

(b) Overlapping community structure.

Figure 1.4: Graphical representation of the two cases of community structure.

Lastly there is the hierarchical structure in which nodes can be placed in some way at different level, above, below or at the same level, given very often by the degree evaluation in social networks.

Two well-known and much studied classes of complex networks are *scale-free networks* and *small-world networks* [8], whose discovery and definition are canonical case-studies in the field, both are characterized by power-law degree distribution. Power law is a general term, but focusing on degree distribution means that a high number of nodes have a low degree and a small percentage have a very high degree. An example of power law is shown in figure 1.5 that is the case of the Orkut database used in next chapters for main simulations.

Instead examples of these networks are shown in figure 1.6.

In a network with a scale-free degree distribution, some vertices have a degree that is orders of magnitude larger than the average, these vertices are often called

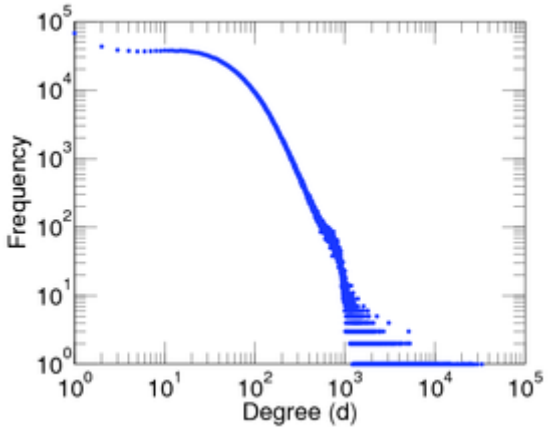
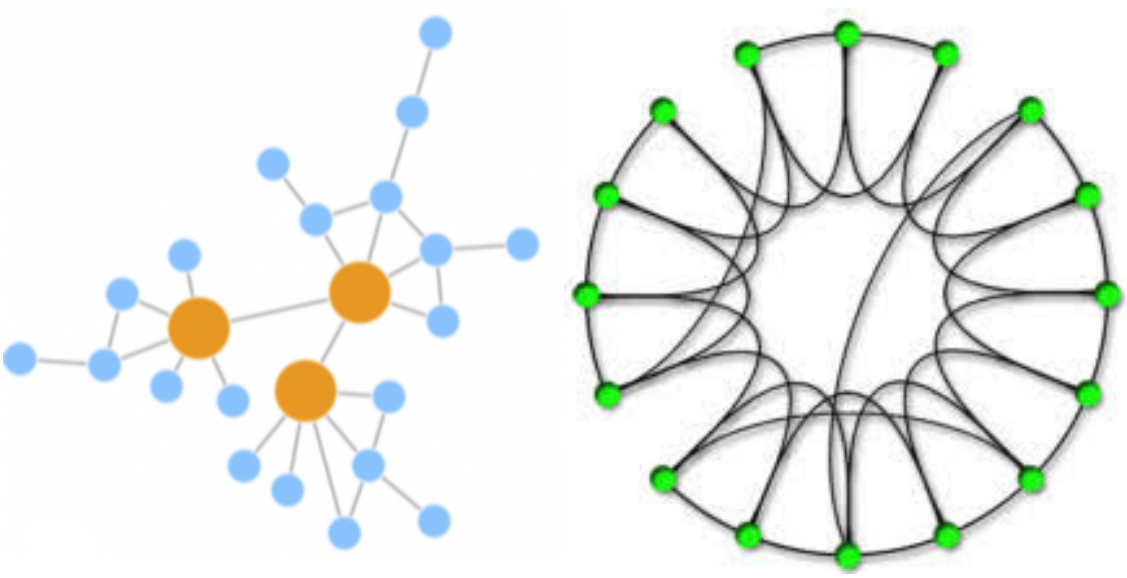


Figure 1.5: Power law degree distribution of Orkut network [2].



(a) Scale-free network.

(b) Small world network.

Figure 1.6: Graphical representation of two very studied classes of complex network.

hubs. This type of network grows exponentially linking preferentially to a hub node, so we are in a situation in which "*the rich becomes ever richer*". The presence of these hubs is the base for the small-world network and the effect of "*6 degrees of separation*" [9], that is the theory according to which two random individuals are in mean connected by 6 levels of knowledge. In this sense the hubs have the role to link certain areas of the graph that otherwise they would not be connected. The small-world phenomenon is so based on the fact that the diameter of the corresponding graph of social connections is not much larger than six (6 degrees of

separation). The diameter is by definition the longest path of the shortest paths between any two nodes in the network. To create a small-world is demonstrated that the addition, on a regular graph, of only a small number of long-range links can create this phenomenon. Doing this the average number of edges between any two vertices is very small and the clustering coefficient stays large. A wide variety of graphs exhibits this kind of property, e.g. random graphs, scale-free networks and real world networks like World Wide Web.

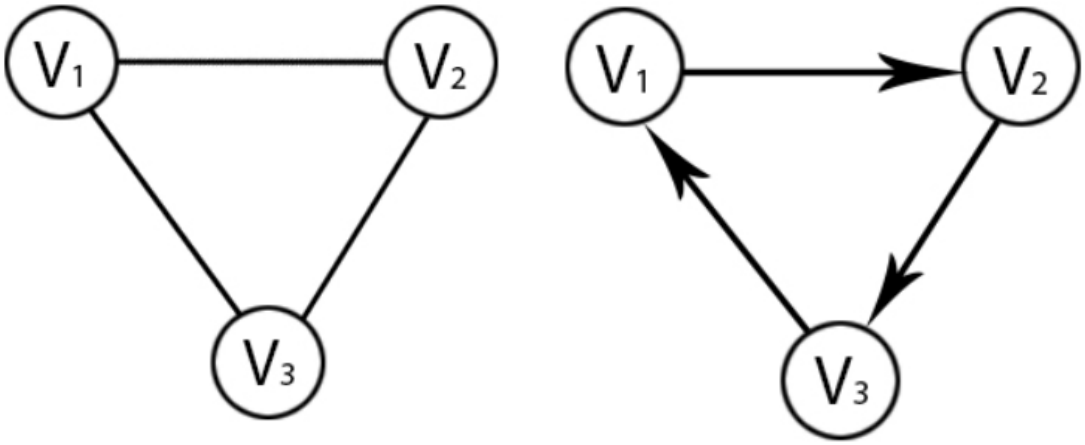
Focusing on these last years it is evident that the social networks have a fast growing, for example Facebook and Instagram. The structure of the network is obviously random because every person that subscribes on the social can follow or become friend of every page or person that it want, so they are creating a completely random network. Internally some particular characteristics are found, for example the higher "directional" degree of famous people, or famous company, etc., and the presence of directed (with pages on Facebook or with a generic page on Instagram) and undirected links (with friends on Facebook or with a generic page on Instagram). The famous pages play the role similar to hubs, in both social networks, because have obviously higher degree (even if we talk about "directional" degree). In structures like these are done the simulations, in the form of databases and supposing all undirected links (directed and undirected features explained better in next section).

1.2 Epidemic process on graphs

A widely studied case of dynamical process is the epidemic process that is a process, so a set of activities, that spreads some information in the general form of the term (e.g., signals, ideas, social media contents, trends, fake news). Physicists, mathematicians, epidemiologists, computer engineers and social scientists share a common interest in studying epidemic spreading and rely on similar models for the description of their particular processes.

The analysis of an epidemic process requires the development of a network framework, that produce results of conceptual and practical relevance. So the simulation take place over a graph representing the system structure. A graph is a discrete mathematical structure that are of interest in mathematics, topology, in certain chapters of computer technology and in engineering fields. A graph data structure consists in a finite (and possibly mutable) set of vertices, nodes, or points, together with a set of unordered pairs of these vertices for an undirected graph or a set of ordered pairs for a directed graph. These pairs are known as edges, arcs, or lines for an undirected graph and as arrows, directed edges, directed arcs, or directed lines for a directed graph, shown in figure 1.7.

The vertices may be part of the graph structure, or may be external entities represented by integer indices or references. A graph data structure may also



(a) Undirected graph. (b) Directed graph.

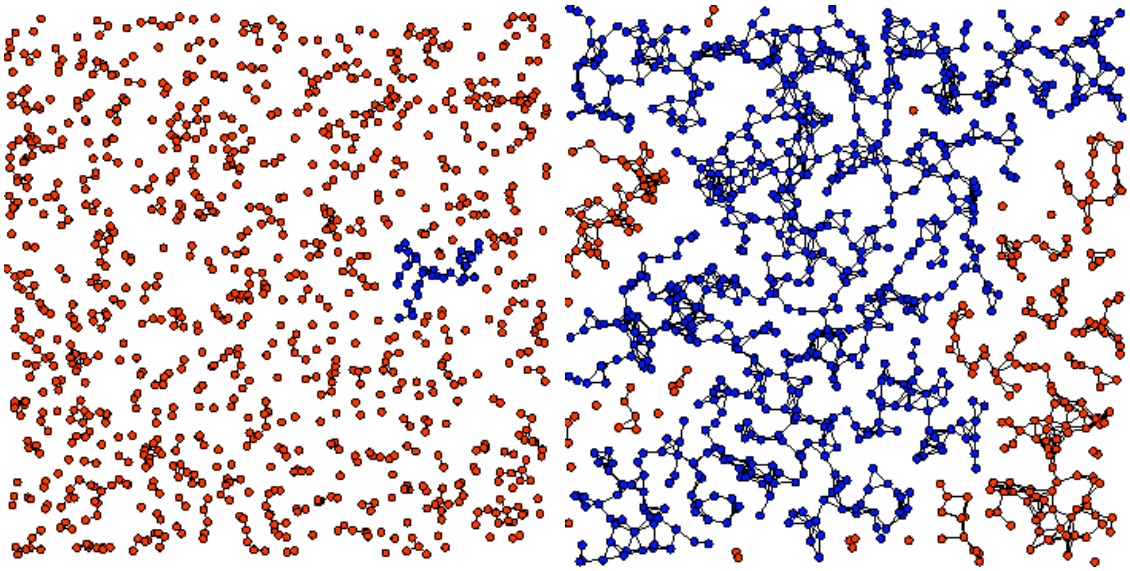
Figure 1.7: Graphical representation of a directed and undirected graph.

associate to each edge some edge value, such as a symbolic label or a numeric attribute (cost, capacity, length, weight, etc.). In the simulations we are talking about undirected graphs, initially unitary weight in all the edges and we have nodes (devices or data points on a larger network) with certain characteristics. The processes modify over time the internal state of nodes and spread across the network following the edges (links between nodes) of the graph. The spreading mechanism is controlled also by different algorithms that are described in subsequent chapters.

1.3 A particular case of study: Bootstrap percolation

This work is started with the analysis of a special case of epidemic process, that is bootstrap percolation process. Initially a random set of nodes are already infected, the seed, that can be chosen deterministically or random (in this case of study is chosen always random). From this initial seed you try to spread the infection and go searching for 'inactive' nodes not already infected. It is possible to see a visual example in figure 1.8.

This is a *threshold-based*] epidemic process, so the nodes are not infected at the first try but they have a threshold r (integer, r$\geq$2) and each edge exerts an influence equal to one (weight, information stored in the edges), so a node can be infected only when has at least r infected neighbours. Then the nodes that have to do infection are successively removed from the 'active' set until the system stabilizes, and the order in which this removal occurs make no difference on the final stable state.



(a) Initial seed (blue).

(b) Spreading of infection.

Figure 1.8: Graphical example of bootstrap percolation on a random graph.

Bootstrap percolation can be interpreted as a cellular automaton (a discrete model studied in computer science, theoretical biology, etc.), resembling *Conway's Game of Life* [6], in which live cells die when they have too few live neighbours. This is a "*zero-player game*", meaning that its evolution is determined by its initial state, requiring no further input. The manner to interacts with the Game of Life is by creating an initial configuration and observing how it evolves, or, for advanced players, by creating patterns with particular properties. However, unlike Conway's Life, cells that become dead never become alive again in this case. Another view is an epidemic model in which inactive cells are considered as infected and active cells with too many infected neighbours become infected themselves. So in this particular case when a node become infected cannot return in the previous state, so we are in the case of time-fix network and permanent decision.

Bootstrap percolation has a rich story because has been proposed initially in the area of statistical physics and has been primarily studied over the year in the case of regular structures like lattice, grids, trees, etc., most notably in a series of papers by Balogh and Bollobás. More recently, bootstrap percolation has been investigated also in the field of random graphs, like in my reference paper, that is the starting point for my work. Random graph is the general term to refer to probability distributions over graphs, and may be described simply by a probability distribution or by a random process which generates them (lies between graph theory and probability theory). A probability distribution is a mathematical function that provides the probabilities of occurrence of different possible outcomes in an experiment. From a mathematical perspective random graphs are used to answer

and find typical properties of complex networks that needs to be modelled.

1.4 Next chapters

In this work the interest is on epidemic processes occurring on very large graphs that have real and irregular structures, trying different techniques of infection with time-fix networks, in the case of permanent decision (Bootstrap percolation) and not permanent decision. These cases are tested with different databases, not only with Orkut, to show that is possible to obtain some general results from these simulations. In next chapters some theory on G(n,p) graphs is explained, also with the edge-based reformulation for G(n,M) [4] in which bootstrap percolation is applied. It is also included a brief theory on Barabasi-Albert model, because our databases are constructed following this model, and a more technical description of bootstrap percolation. Then all the details on the developed C++ program, the algorithms and the structures used. Lastly the most important one, a chapter with all the simulations done and the discussion of the results.

A work that can be done following the purposes of this thesis is to focus the attention on time-varying networks. This case is complex and interesting because is closer to real case studies given by the fact that social networks are in continuous change.

Chapter 2

Theory

2.1 Theory on G(n,p) graphs

Talking about real network structures, we need a mathematical abstraction, this allow to work analytically with simplified reality. We can have deterministic topological structure in which properties and topological characteristics are deterministic, and random topological structure defined by a random process.

The G(n,p) model is a classical model for random graphs, introduced by Erdos and Renyi into '50 [5] (Binomial model, Erdos-Renyi model), each possible edge of the graph exists with probability p independently (undirected graph with no loops). This model has two deterministic parameters: n, number of vertices, p, probability of existence of each potential edge (can be fixed or variable), and the outcome is random with many possible samples. An example of G(n,p) graph is showed in figure 2.1. (add details to explain the example)

So the structure is random and depends on the realization but we can still characterize its topological properties probabilistically.

We have some simple properties of the G(n,p) graphs [7]. The number of different graphs that can be created is $|S| = 2^{\binom{n}{2}}$, this binomial coefficient because each edge can or cannot be present. The probability to generate a specific G(n,p) graph with n vertices, depends on the given set of edges $E = ei, e2, e3, .., e_m$, and is $P(togenerateE) = p^{|E|}(1-p)^{\binom{n}{2}-E}$. All graphs are equiprobable when $p = 1/2$. If we want to know how many edges a G(n, p) graph has we can use the random variable M and the distribution is $P(M = m) = \binom{\binom{n}{2}}{m} p^m (1-p)^{\binom{n}{2}-m}$. For the expect value of M instead $E[M] = \binom{n}{2} p = [n(n-1)p]/2$.

Any vertex of the graph has certain characteristics, a common one is the degree, so the number of edges that exit from the node. Each potential edge exists with probability p, $G(n,p)$ has on average $\binom{n}{2} p$ edges, so we can obtain the distribution of the degree D in this manner $P[D = k] = \binom{n-1}{k} p^k (1-p)^{n-1-k}$, with $k = 0,1, .., n-1$

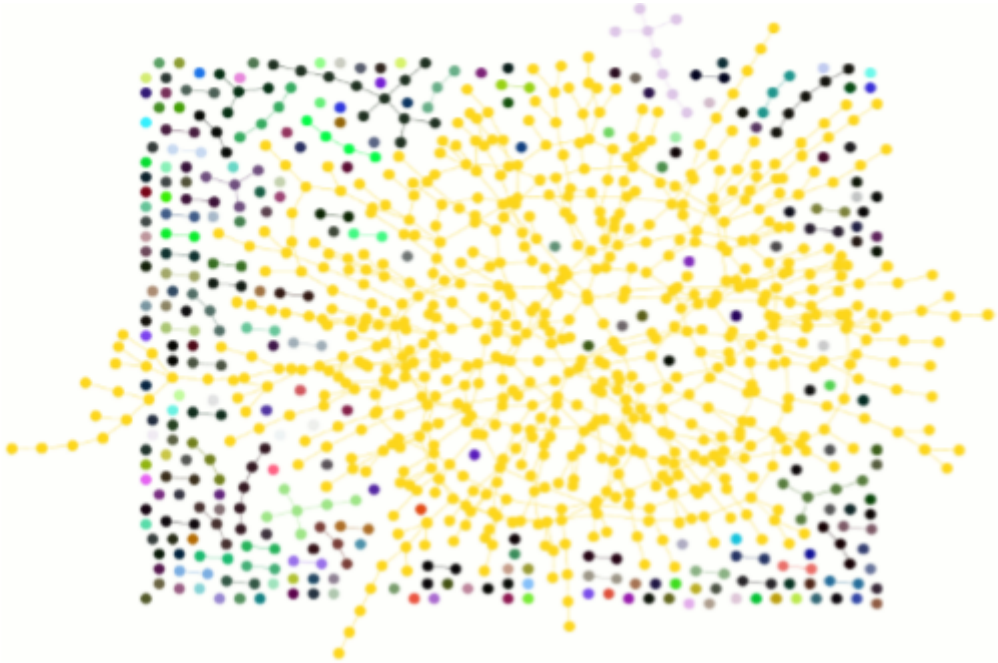


Figure 2.1: Example of a G(n,p) graph with n=1000 and p=1.5/1000 [7].

and the expected value is $E[D] = (n-1)p$. When we talk about graphs with large number of nodes the formulation become $p_k \approx \frac{E[D]^k}{k!} e^{-E[D]}$, for large n and $k << n$.

So we are interested on topological properties of $G(n,p)$, that can be satisfied by all the graph or a subset, satisfied probabilistically. We look at the asymptotic behaviour when n is very large, so while n$\to\infty$ the all structure of the graph depends on p, that can be fixed and not variable with n or can be function of n ($p(n)$) and so varies with n.

This model is mentioned but not used in this work, it is anyway the basic theory needed to go further and understand the process globally.

2.2 Theory on Barabasi-Albert model

This is an algorithm for random graphs based on PA (*Preferential Attachment*) mechanism [3]. This model permits to construct a *Scale-free network*, so the type of real networks that are studied in this paper. The principal characteristic of a real network, focusing on social networks, is that there are always nodes that tends to be more "famous" with respect to others. This translated algorithmically means nodes with an high degree and so high connection. The PA exploits this characteristic of *"Rich gets richer"* to construct synthetically the network. An example showed in figure 2.2.

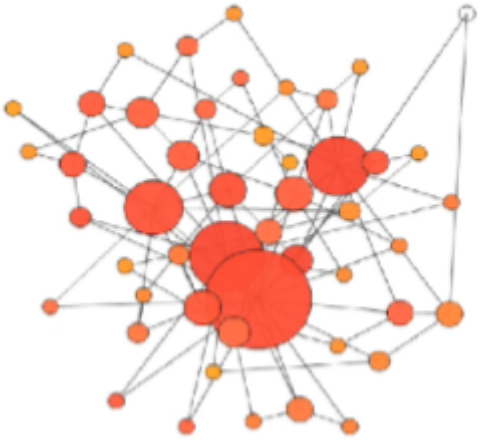


Figure 2.2: Example of a network constructed with BA model.

The process consists initially to have a small clique of nodes interconnected, then at each step a node is added with a degree m and interconnected with edges to random nodes. The interconnection is completely random in the set of nodes already present in the network but some are more likely, with a probability proportional to their degrees. It is possible to reach different results choosing different values for m, so increasing this value we obtain behaviours like in figure 2.3.

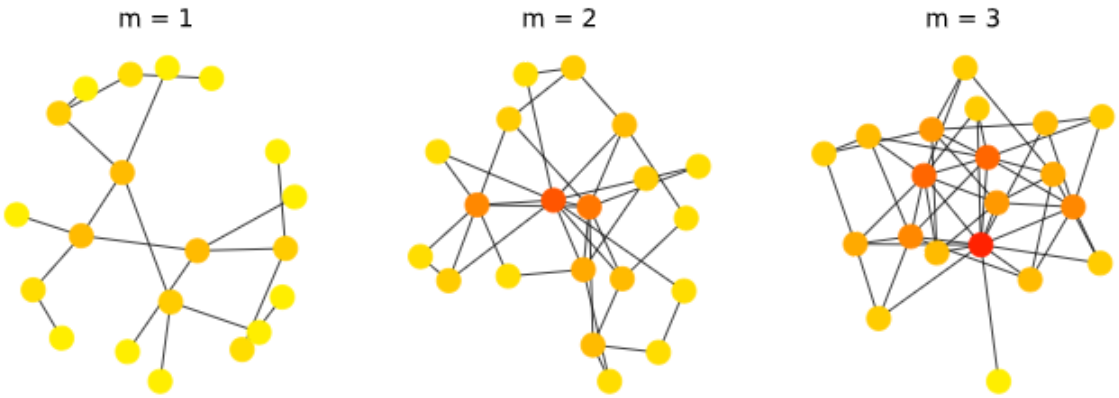


Figure 2.3: Example of behaviours changing the value m.

To obtain the *Preferential definition* we consider a graph G=(V,E) at the time t=1, 2, etc. with $d_u(t)$ that is the degree of vertex u at time t. Considering the arrival of a new vertex at time t it is possible to note that brings m edges (value decided a priori, for example m=2). Hence it is possible to obtain this formula:

$p_u(t) = \frac{d_u(t)}{\sum_{v \in V} d_u(t)} = \frac{d_u(t)}{2mt}$ in which $p_u(t)$ is the probability for a vertex u of being incident to a new edge at time t and $2mt$ is the number of edges in the graph at time t. The vertex u is inserted at time t_u and initially $d_u(t_u) = m$, so by construction older nodes are the best nodes in terms of greater degree. The power law arises because the training process follow the PA principle, but not all the power law phenomena can be explained by PA. As we have said older nodes are the ones that can exhibit higher degree and new nodes have no chance of being "popular". Moreover nodes can create edges only at the instant in which a node enter for the first time in the network and no further evolutions of the network are permitted. These last considerations are some limitations of the BA model.

2.3 Theory on Bootstrap percolation

In bootstrap percolation the nodes start with an own integer threshold $r \geq 2$, and we start by choosing uniformly at random the initial seed of nodes $A(0)$ that have a certain cardinality a.

We proceed with this particular epidemic process by set all these initial nodes as 'active' (from standard terminology provided by Erdos-Rényi graphs) by means of a variable, so we can also say that these nodes are 'infected' (other terminology that can be used). In the program while the nodes are chosen randomly these are added sequentially into a stack, a data structure that works as LIFO (Last In First Out), and all their edges are set as 'usable', meaning that by these edges we can try to spread the infection in the next steps of the algorithm through the graph reaching the 'inactive' nodes. An inactive node can become active if at least r neighbours are active, and when a node is active it never revert to the state of inactive, so the set of active nodes grows monotonically. This is useful for the observation of the distribution of infected nodes. However an used edge can be reused by another node to 'infect back' a node that have already used the same edge for the same purpose. This "error" does not matter since it has no impact on the percolation process.

The process evolves automatically through generations with this algorithm trying to infect other vertices using all the edges of the nodes present in the stack, so the first generation is composed of all the vertices activated by the initial seed, the second generation of active nodes is made of all the node activated by the joint effect of the seed and the first generation nodes, etc. When all the edges of an active node are 'used' the node is removed from the active set, and when a new node is found and turned to active it is added to the active set.

The process stops when all the nodes of the stack are investigated, so when the stack results empty, and in this manner we include also the case in which an empty generation is obtained and the case of all nodes active.

However, by doing this, the generations remain simply a theoretical aspect and

we can practically forget about these, because what we have adopted is a reformulation of the bootstrap percolation (with respect of what was initially proposed) in which we change the time scale by introducing a virtual (discrete) time step $t>N$, such that a single active node is 'explored' at each time step. By doing so we obtain a process equivalent to the original one but with an algorithm more generic and reusable.

Hence for this formulation we need to introduce some notation [4]. The set $A(t)$ is the set of nodes active at time t, another set $Z(t) < A(t)$ is the set of used vertices, which is the subset of active vertices, of cardinality t, explored up to time t. The first initialized set is the $A(0)$, so the seed set, and the set of used vertices is initialized to the empty set $Z(0)=void$. Each node have a counter $Mi(t)<N$ initialized to 0 at time $t=0$ (these counters are independent from node to node). At time $t=1$ we arbitrarily choose a node $z(1)<A(0)$ and we try to spread the infection to its edges, incrementing by one the counter of all its neighbours. In our case, programmatically, we simply do a pop from the stack so we choose the last node pushed in, that is the same in terms of results, because is also random the last node. By doing so we use node $z(1)$ and it is added to the set of used nodes $Z(1)=z(1)$. We continue recursively in this manner and at each time t we pop from the stack an active node that as not been already used. Every time we increase the $Mi(t)$ of a node we check if this is an inactive node, denoted with ΔA(t), that can become active for effect of the distributed marks at time t. Such newly activated vertices are added to the set of active nodes: $A(t)=A(t\text{-}1)+\Delta A(t)$. Note that no vertices can be activated at time 1, because $r\geq 2$. The process stops when $Z(t)=A(t)$, so when all active nodes has been used.

Practically, we have used for our program the *Edge-base reformulation G(n, M)*. For this reformulation, looking at the edges point of view, we can add also this notation: when a node becomes active, all edges connecting this node to other nodes which are still not active are denoted as 'usable', and added to the set B of usable edges. At a given time step t, one usable edge is selected uniformly at random from $B(t\text{-}1)$, adding one mark to the endpoint that was inactive (when the edge become usable), only if this endpoint is still inactive. The selected edge is then removed from $B(t)$. Set $B(0)$ is initialized with the edges connecting seeds to non-seeds. Following this construction at most one node can become active at each time instant t, and having $A(t)$ we can write $A(t)<=a+t$.

The dynamics of the epidemic process are determined by the behaviour of the number $A(t)$ of active nodes, so active nodes which have not been already used, hence radically from the random seed chosen initially and its cardinality a. Note that we assume r$\geq$2 because the case in which a node can be infected by just a single neighbour is degenerate and lead us to the trivial fact that a single seed is enough to infect the entire connected component it belongs to. Therefore this is of no interest because many real systems are connected by construction.

Chapter 3

Description of the program

3.1 How to build the basic structure and details

We have to prepare our C++ program to store and create step by step a connected graph looking all the links/edges present in a generic database. Initially we have tested all the functionality with the Orkut database. These databases are generically made by a list of couple of numbers spaced, the couple together is the link to create and the single numbers are the identifiers of the nodes to connect (e.g., 3 2, 7 10). The program can read line by line from databases structured in this manner (that are the common ones) and save all the data in appropriate classes, constructing step by step our graph structure.

For this structure I have created three classes: *Node*, *Edge* and *Graph*.

The *Node* class have its features inside such as identifier number (unique number readable from the database), flag to signal the infection (0 -> not infected, or 1 -> infected), threshold for the infection, colour, information on neighbourhood colours, and for last contains the pointers to the next node and to the first edge in the node list, that is the list of neighbours. Furthermore to speed up the simulation it has been added in the node structure a further pointer to the last edge connected to the chain. In this manner it is avoided the scan of all the list of edges to add a new one. This trick is very relevant because we work with huge databases as Orkut that can have nodes with more than 30k connections.

The *Edge* class is structured similar to a node, so have some features inside like state, weight, and have two pointers, one to the next edge in a certain node list and the other to the connected node.

Finally the *Graph* class contains only the pointers to the head node and to the tail node, to simplify some low level operations done internally in the program and to speed up the simulation.

Graphically we have the kind of structure showed in figure 3.1.

Now are described better the main features. The *identifier number* is used to identify uniquely a node in the graph, when a specific node number is read for

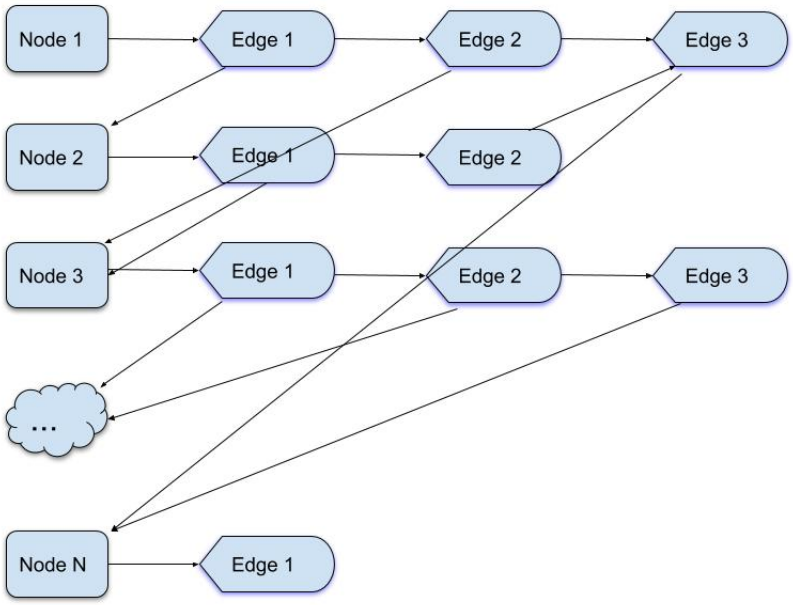


Figure 3.1: Graphical representation of graph structure.

the first time from the database a specific Node element is created and connected suitably. Furthermore the pointer to this new node is saved in the *nodesArray* vector, that have the dimension of the graph, this is useful to reach the element with a complexity of *O(1)* in a second moment, without scan all the graph. Hence in this manner the complexity is lower, but mostly the time needed to search for a specific node number is very short, this is a hinge point for our simulations.

Next important characteristic of a node is the *infection flag*, that allow us to understand if a node is already infected (value 1) or not (value 0). A node can be infected if it is part of the initial seed that start to spread the infection or can become infected by effect of spreading mechanism, only if it is reached the infection threshold.

The *infected threshold* is set normally at value 2 for the majority of simulations. This means that are needed two infected neighbours that spread their infection to the inspected node.

The colour instead is not used for initial simulations because we fall in the case of initial single seed. Subsequently it is used in the case of double seed to check what colour succeed to spread its infection.

It is possible to say something more about the *state* of an edge. It can assume values *0→not usable*, initial condition in which the infection not yet passed through,

1→usable, so a node that is connected to this ready for inspection and then try to spread infection through it if get infected and *2→used* if an infected node has spread through this link.

For last we have the *weight* that remains of value 1 and it represent how much an edge counts for infection. An example to understand is an edge with weight 2 that can infect alone a node reaching in one shot the infection threshold.

3.2 Construct the initial seed

Talking about initial seed we have created a function that can choose randomly a node one by one from the total number of nodes present in the graph and add these to the seed, this is done simply adding to the stack of integer *infectedStack*. In the stack we have saved in this manner all the nodes initially infected, the active set, setting in their nodes class the infected flag equal 1 and the threshold of infection already equal to the maximum value. Moreover for all the nodes added to the stack, the usable flag in all their edges is set to 1, the meaning is that we can inspect the edges of a node trying to spread the infection, and then the node is popped from the stack. This discussion is valid when we talk about single initial seed, the configuration used for the first experiments, so it is enough to think about coloured and not coloured nodes for these cases.

In fact we have done also a second function that initialize a double seed. The method for this one is the same, so we generate one by one a random number in the node's set but also randomly the node is marked as colour 1 (1→RED) or colour 2 (2→BLACK) thanks to the infected flag that can assume three values (also 0→WHITE, no infection). As we have done previously these node are added all in the *infectedStack*, the active set, ready to be inspected to try the spreading of infection, and all their variables are set opportunely, like we have explained before.

I have constructed two different functions for the double seed. The first one is "not an equo function" in the sense that are found all the seeds of the colour 1, so added in the *infectedStack*, and then are found the seeds of the colour 2, so added in queue and extracted first because the structure of the stack. The second one instead is an "equo function" because choose randomly seeds of colour 1 and 2 and consequently fill the *infectedStack* mixing colours, so the effect of the stack is overshadowed when I do pop.

3.3 Spread the infection with Bootstrap percolation mechanism

3.3.1 Bootstrap percolation on single seed

Now we can talk about the functions that spread the epidemic.

The first is named simply *epidemicProcess*, a basic function that is characterized by the use of a single seed with a given amount of repetitions to provide a better study of the epidemic process and understand in mean how it works. For this function we have permanent decision in terms of infection, so if a node is infected cannot be disinfected in some manner. This single seed is changed with a higher value every time we finish the given repetitions, because we want to see the trend of infection changing these parameters. Hence to run correctly the function needs the vector containing all the seeds to test and the variable containing the number of repetitions to run for every seed. In this manner we obtain not a one shot simulation but average values that are very useful in terms of average behaviour (so we are talking about Montecarlo method). For every repetition, while we have nodes in the *infectedStack*, we pop from the stack and we try to propagate the infection throw neighbours of actual node under test. Remember that a generic node can be already infected because is part of the seed or can be infected if the number of infections on it exceed the *infectionThreshold*. If a node is infected with this procedure is pushed into the *infectedStack*, ready to be analyzed at next step. This procedure is done continuously until we have the empty *infectedStack* condition. The results that we have are the number of infected nodes for every repetitions, but more important the average number of infections looking all the repetitions, very useful to plot this parameter with respect to the initial seed used to show the average behaviour.

3.3.2 Bootstrap percolation on double seed

A natural evolution is the same function but with double initial seed, named *epidemicProcessDouble*. Also in this function we have permanent decision. This epidemic function works in the same manner as before, the difference is that now we can infect with two different colours (for example in our simulations *RED* and *BLACK*), so for the infection we have not the *infectionThreshold* but a so called *neighboursThreshold* that works in different manner. When a node is popped from the *infectionStack* we look for all its neighbours trying to spread the infection to them. Every neighbour can be infected by a colour only if in its neighbourhood the specific colour reaches or exceeds the *neighboursThreshold* when compared to the other colour. If yes the actual neighbour node in exam is infected by the specific colour, if no the node remains *WHITE* so no infection. The results that we want are the same of the previous function, so average number of infected nodes (given

by subsequent repetitions) with respect to the initial seeds. The difference is that now we have two trend of the two colours.

3.4 Spread the infection with a modified version of Bootstrap percolation mechanism

Next step was to remain in the case of double seed, but with a change of implementation that bring not to a permanent decision about the infection, so more realistic and interesting case. This change is related to our interest on real cases like opinions in a social network or ideas, something that is not static but can change with the time maybe by a cultural or external influence. This is translated in a modification of the last epidemic function we talked about, so this new function is called *epidemicProcessDoubleMod*. The skeleton of the algorithm remains the same but obviously we have to take into account more all the cases of disinfection and reinfection with a different colour. Hence if a node is white can be coloured, then can change colour, return white and can also return coloured, all is always dependent by the neighbourhood and the change occur if the rules of the *neighbourThreshold* are reached. Since the decision to infect a node with a colour or disinfect is not permanent, when a change in a node happens, the mentioned node is scanned and all the neighbours are pushed into the *infectedStack* to be subsequently re-analyzed because the change happens also in their neighbourhood. Thinking about the termination condition of empty *infectedStack* we can realize that is almost never reached because the continuous changes that bring to re-fill the stack, so we have to add another variable that can stop the process in a suitable manner. For this purpose a new variable called *limit* is insert in every node counting all the changes that a specific node suffered. When for a generic node this variable reach a specific value the mentioned node cannot return in the stack and this bring the function, after a certain amount of time and steps, to the principal termination condition we talked about. The simulations were done testing different possible values for the *limit*. Tests were done first with the *maximum degree* as upper limit, because working with networks characterized by hubs this value it is very high. It was possible to see that using this value the simulations give us optimum results in terms of behaviour but in too much time, so not completely good. To search for a compromise in results and time we have seen that using the *average degree* as limit value are obtainable same results in much less time.

To run this function is needed like before the vector containing all the initial seed of infection to use and the number of repetitions to test each single seed. For this case the use of the seeds is a little different because is doubled, so are created two different initial seed distinguishable by the colour set internally to nodes.

Furthermore to lighten up the algorithm and speed up the simulations other two variables are used internally to every node to count the number of red and black

neighbours, *neighRed* and *neighBlack*, so not every time is necessary to scan all the neighbourhood. This long scan is in any case present, but only when a change in a node happens and we have to reach all the neighbours to push one by one into the stack to re-check the neighbourhood.

3.4.1 Synthetic graph

In this section it is explained how to run an epidemic process on a synthetic graph, so not a real one but constructed from zero. The trick is to not construct the database and then read line by line like before but immediately construct the graph that we want. Obviously the construction must be accordant to a *Scale-free network* so it is used the BA model explained in theory. Hence it was created the function *syntheticBAGraph* to which is passed the *initialClique* and the number of m of initial edge that a new node have when starts to be part of the network. Internally was set a parameter *numNodes* that tell us how much nodes we want inside the network. The hinge point of the function is the *probabilityVector* in which all node numbers are repeated inside as many time as they gain a new edge, so initially m times. This is done because the PA (Preferential Attachment) principle, so in this manner we can link with more probability to a *hubs* because the repetitions. Through this procedure it is possible to create the graph/network that we want. At the start the *initialClique* is fully connected and new nodes are generated with casual identifier in the set of *numNodes*. These nodes are connected suitably at random with other nodes already present and the created connection are added to the *probabilityVector* to increase the probabilities of future connections and create *hubs*.

Next step was simply to run on this synthetic graph our previous algorithms.

Chapter 4

Simulation details and results

The created functions has been used to simulate different cases of epidemic processes, not all with bootstrap percolation mechanism. The main used database is the Orkut one, a part of an old social network designed by Google on 2004. Internally we have for every line of database a couple of number referring to users connected.

4.1 Bootstrap percolation

4.1.1 Single initial seed

The first simulations was done on the Orkut database, using the function *epidemicProcess*, so the basic one. The results are showed in figure 4.1.

As we can see from the figure the choice of seeds is appropriate because we can reach almost total infection. Looking the effect of the seeds from 70 to 100 we can say that are completely superfluous, complete infection is already reached at seed 60.

Analyzing the first part of the graph, from very low seed to 18, we can see that the spreading of infection grows very fast in an exponential way and we are able to infect practically half of the entire network structure. This is what we expect by a network structured like this, social network are placed in the set of scale-free networks, so some nodes are strongly connected to the network with many edges and touching these nodes initially leads to an obvious huge spreading of infection.

Next behaviour that we can notice is a little drop of infection at seed 20, this is not significant but show us that choosing every time a random initial seed we can fall also in this case. Simply what happens is that we have chosen for this seed in average nodes poor connected that brings to less spreading of infection, compared to the previous case in which we have less initial node in our seed but these are more connected.

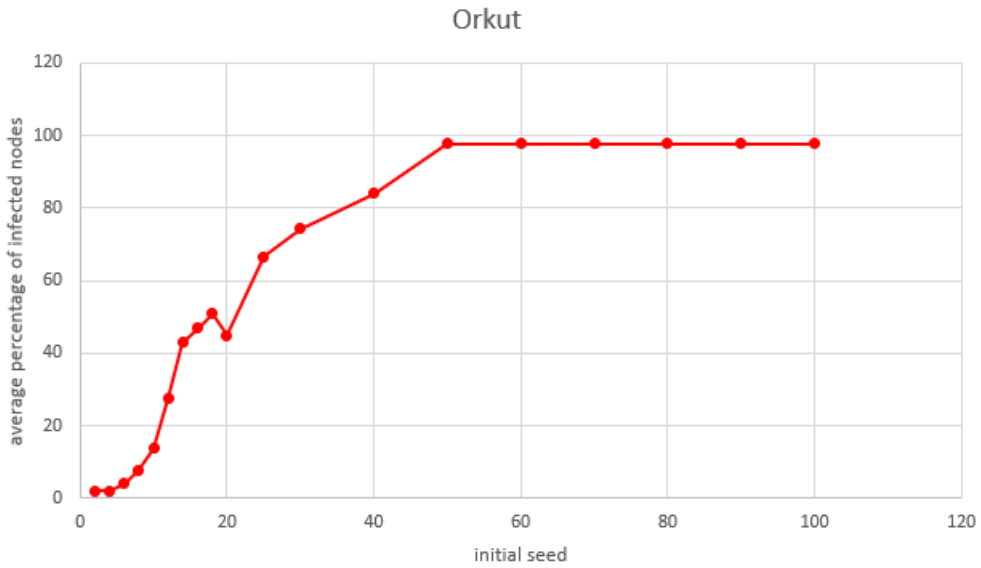


Figure 4.1: Simulation results of a bootstrap percolation on Orkut network, single initial seed, permanent decision, infection threshold=2, 50 repetitions per seed.

After this drop the curve of infection continue to grow up like before for a short distance, so with an exponential or better super linear behaviour, for the same previous motivations.

Then the slope change and become linear because it has been reached almost the 3/4 of total infection. Through more connected nodes the infection has been spreaded and remain only other nodes we can say "more difficult to reach" because of their low connection. Hence the slope of infection is slowed down in the interval from seed 25 to seed 50.

In the end we can observe a flat behaviour at practically 100% of infection due to the fact that the infection has reached all the possible nodes. The nodes excluded by the infection process can be isolated nodes, so with no connections, nodes with only one connection, so no possibility to raise the basic infection threshold due to network structure issues, and consequently all the parts of network isolated by this kind of nodes. These last particular nodes act as a bottleneck.

The nodes for the initial seed are chosen with a pseudo-random number generator, so if we start again all the simulation taking the results of all the average infection values we will obtain the same trend as the figure 4.1. Therefore to do a further averaging we have used in the program the *srand()* function that allows to change the seed used by the pseudo-random number generator algorithm.

Doing this for three times more randomness and smoother curves was obtained as showed in figure 4.2. It was also avoided in this manner the drop of infection occurred before at seed 20.

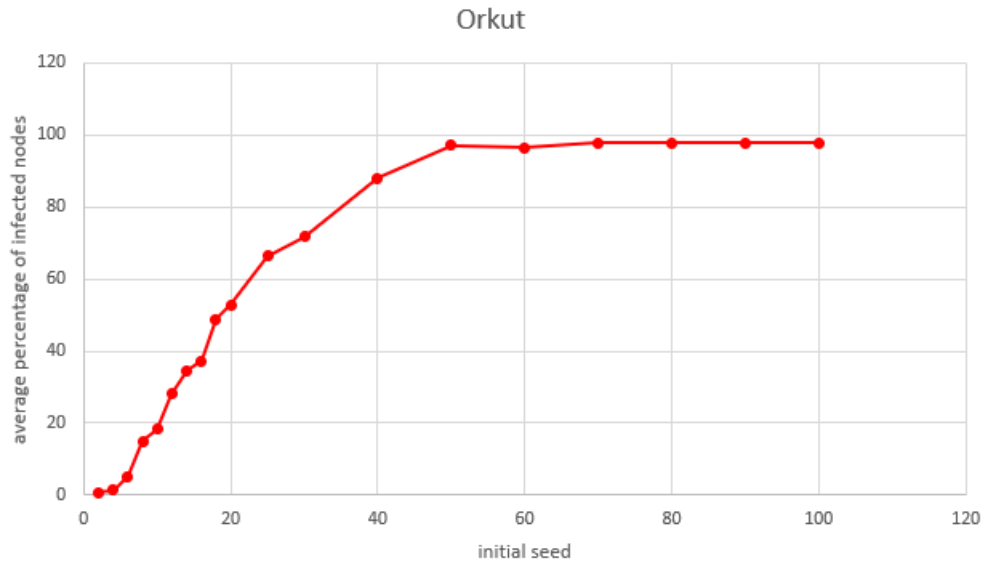


Figure 4.2: Simulation results of a bootstrap percolation on Orkut network, single initial seed, permanent decision, infection threshold = 2, 50 repetitions per seed, average of 3 level of randomness

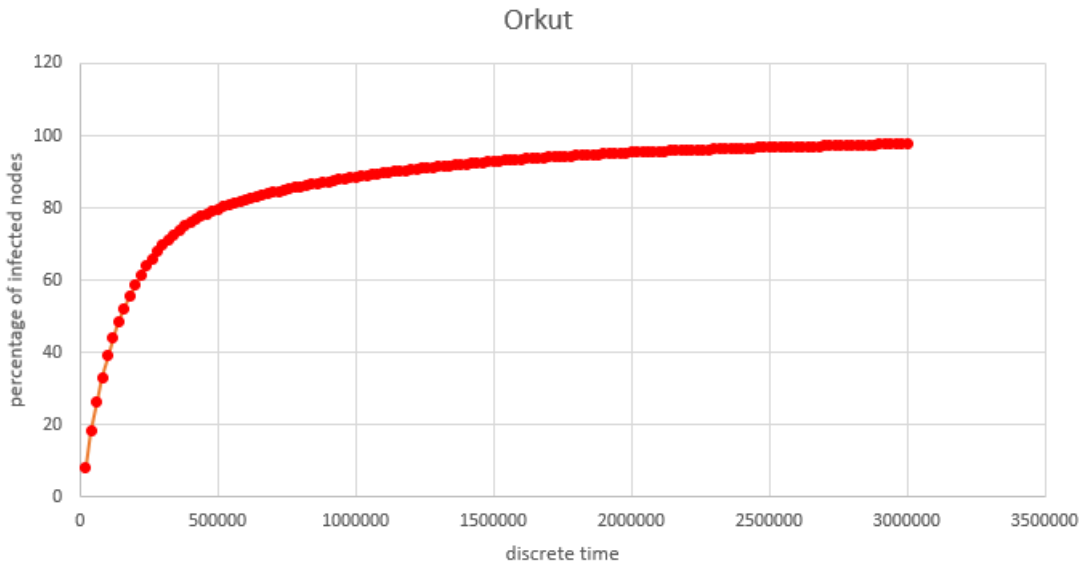


Figure 4.3: Simulation results of a bootstrap percolation on Orkut network, single initial seed, permanent decision, infection threshold = 2, "one shot" with initial seed 60.

Next thing showed is a "one shot" simulation in which it is done only one repetition of one seed that can reach at the end the total infection (initial seed 60 used). This is showed with respect to a discrete time set as the number of extractions done from the *infectedStack*. The values of infection was stored every 25000 extractions. The graphical result is visible in figure 4.3.

4.1.2 Double initial seed

Now are showed instead the results of bootstrap percolation done over two initial seeds. The function used is the *epidemicProcessDouble*. The results are visibile in figure 4.4.

Figure 4.4: Simulation results of a bootstrap percolation on Orkut network, double initial seed, permanent decision, infection threshold = 2, 50 repetitions per seed, seed with equo function

As we can see the infection for both colours grows almost linear with respect to the increasing of initial seed. In reality the graph is only stretched and the behaviour is also in this case exponential. The curves have the same behaviour and infects with equal probability the network, characteristic visible from the graphic contention. It is noticeable a spike at seed 90 for the red curve with respect to the black but it is not relevant because the randomness. In fact after this we can see another time a certain stabilization. The two curves goes to an almost total infection, we reach with seed 150 a 94% on the total number of nodes. Increasing

more the values of initial seed surely it is not possible to reach 100% of infection now. The fact is that we have double seed and the rules of infection are changed, so spreading and touching a node one time or two times is not sufficient. Now we start to follow the rule of the *neighbourThreshold*, so unless one colour raise or exceeds this value with respect to the other colour in the neighbourhood of a node it cannot be infected.

4.2 Modified Bootstrap percolation with double initial seed

Another more complex simulation was done using the function *epidemicProcess-DoubleMod*. This function is characterized instead by a double initial seed and not permanent decision for what concern the infection, so change are permitted continuously.

Talking about this particular case we have initial *BLACK* and *RED* equal seeds, so the infection is carried on for these two colours. Internally to this simulation were done two variants: the first one with the use of the *initialSeedNotEquo* function and the second with the *initialSeedEquo* function. Results are showed in figure 4.5 and figure 4.6 so it is possible to compare and explain the differences.

4.2.1 Initial double seed with the "not equo function"

In figure 4.5 we have shown the results of the first variant in which the initial seeds are the same but they are not distributed equally for what concern the position in the *infectedStack*.

From the simulation we can observe that the behaviour of the red and black curve are practically the same initially and then starts to step away one from each other, this is caused by the internal structure of the function used to initialize the seeds.

The explanation is that all the seeds infected are initially pushed into the stack waiting to be analyzed but this function put at first all the red seeds and then all the black ones. In this manner it begins to infect starting from black seeds because the pop mechanism offered by the stack, so more probability to infect with the black colour.

Initially the two curves have almost the same behaviour because we can come up against black seed as red seed and looking the neighbourhood we can infect by one colour new nodes with the same probability for red and black. The difference between the two colours is visible when a black node with a high degree so high connected succeeds to infect first with its colour a part of a network that is completely white. For this type of event the probability for black nodes to hit is higher

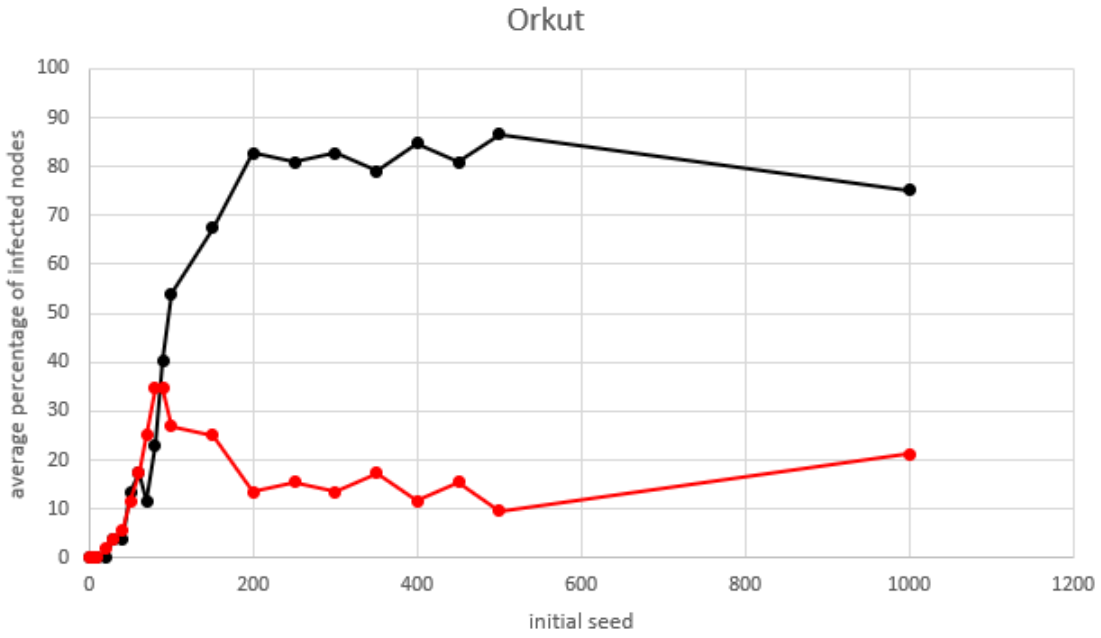


Figure 4.5: Simulation results of a bootstrap percolation on Orkut network, double and equal initial seed, not permanent decision, neighbour threshold = 2, 50 repetitions per seed, seed with not equo function.

because from the stack continuously are extracted black nodes first and then after all the black seeds the red ones.

Also the fact that we are in not permanent decision cannot change nothing about this trend in which black nodes win. The only thing that we can notice about is what we see at seed 200, from here are visible some oscillations but the trend remains the same anyway. Therefore the percentage of red infected nodes go down as the percentage of infected black nodes go up, and looking at long run (seed 1000) in which all the possible continuous changes of infection colour or disinfection happens the behaviour remains the same.

4.2.2 Initial double seed with the "equo function"

Now is explained instead the second variant of the same simulation showed in figure 4.6.

The peculiarity of the second variant is that the seeds are assigned in equal random manner, so in the stack we have the possibility to put first a red or a black seed with equal probability and so on in this manner. This procedure stops when all the seeds are assigned to the respective colours.

This change on the assignment of seeds do not change the first part of the

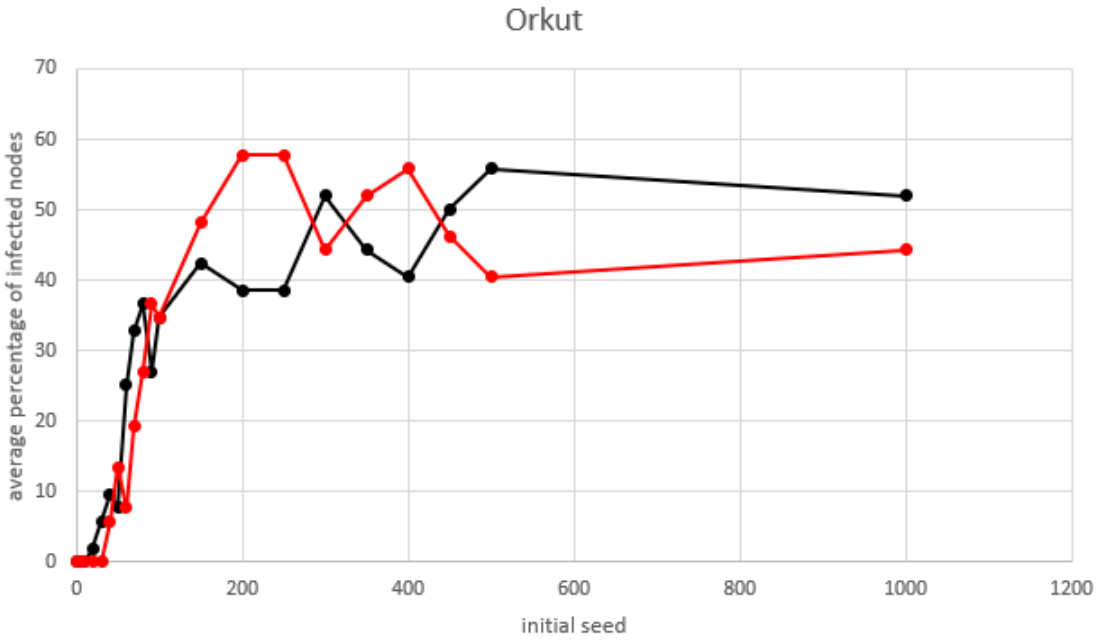


Figure 4.6: Simulation results of a bootstrap percolation on Orkut network, double and equal initial seed, not permanent decision, neighbour threshold = 2, 50 repetitions per seed, seed with equo function.

infection in which both colours raise a certain level of infection together, like before in an exponential manner. The difference on the trend take place when most of the high connected nodes are infected and happens that a certain colour can touch a white "marginal" area of the social network. As we can notice initially the red curve of infection predominate on the black one, that go down in the same manner because the portion of network that they contend is the same probably. The same portion is always the last because at seed 200 we can say that we are almost at the maximum infection looking the percentage, almost 60% and 40% for respectively the red and the black colour.

After this point there is a specular oscillation between the infections caused by the possibility to change the state of infection in this area in which nodes are not well connected, so a little change of infection can bring to a consistent change.

Thanks to this we have the graphic competition showed in figure 4.6 in which sometimes the infection is won by the reds and sometimes by the blacks. At long run was tested with a large seed of 1000 and as you can see the behaviour remains the same.

4.2.3 Same experiment but changing the *neighbourThreshold*

Now it is repeated the same experiment as before but changing a parameter that influence the infection, the *neighbourThreshold*. The results of the simulation are showed in figure 4.7.

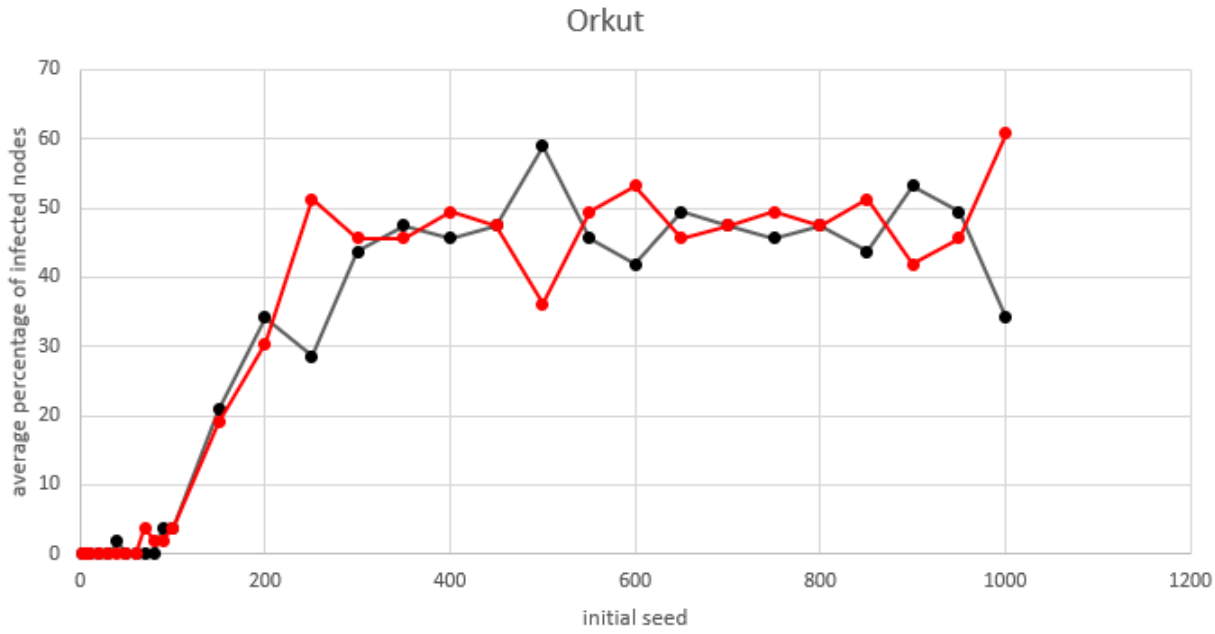


Figure 4.7: Simulation results of a bootstrap percolation on Orkut network, double and equal initial seed, not permanent decision, neighbour threshold = 3, 50 repetitions per seed, seed with equo function.

We are in the same case as before but here all the results of simulation are stretched. This is due by the fact that now a node have a lower probability to be infected or we can say that are needed more neighbours infected because *neighbourThreshold=3*. As before we have an exponential increase of infection but in a bigger range of seeds until seed 300. Then we have a little linear increase of infection until seed 450. This behaviour is almost the same as we can notice before but in the previous case is all pressed until seed 180 because the lower value of *neighbourThreshold*. Hence it is possible to see in the figure that the main difference is that to reach the maximum possible infection we need an higher initial seed. Here is around seed 450 instead previously was around seed 200. It follows the graphic competition like the first case with the same explanation.

4.2.4 Initial double seed with the "equo function" and different database

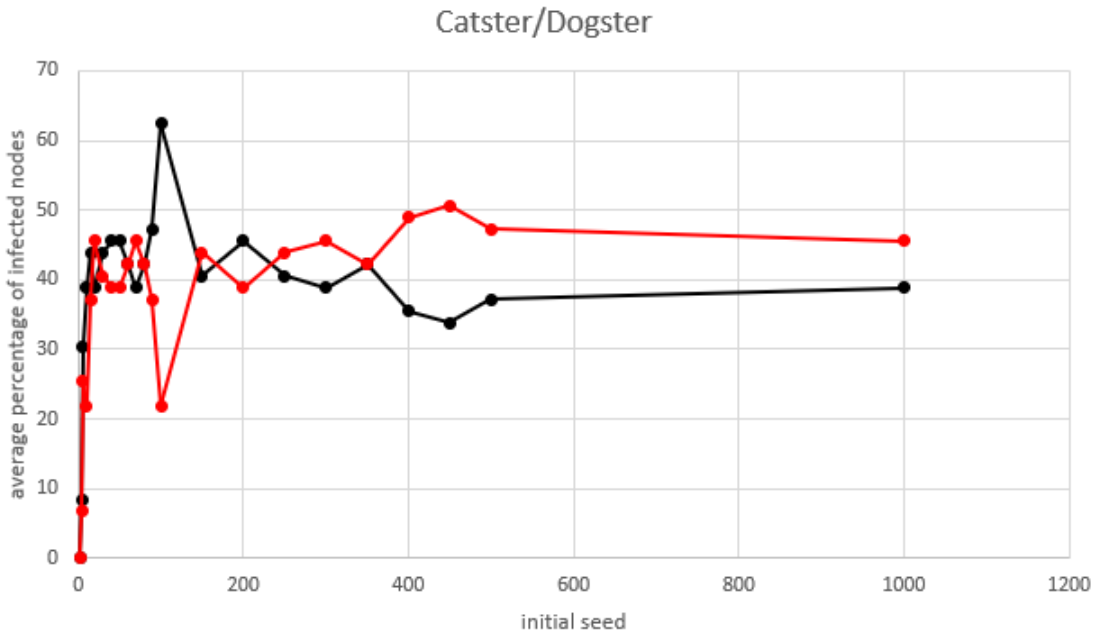


Figure 4.8: Simulation results of a bootstrap percolation on Catster-Dogster network, double and equal initial seed, not permanent decision, neighbour threshold = 2, 50 repetitions per seed, seed with equo function.

All these previous simulations are done with the Orkut database as we can notice. Now the results obtained using Catster-Dogster database [1] are showed, to prove that these are general for every social database. The mentioned one is a network contains family links between cats and cats, cats and dogs, as well as dogs and dogs from the social websites *catster.com* and *dogster.com* (also included are cat-cat and dog-dog friendships).

In figure 4.8 are showed the results of the last simulation done before with Orkut, the second variant of *epidemicProcessDoubleMod* with *initialSeedEquo* now done with Catster-Dogster. This database is not large as the Orkut, it has less nodes (about 623,776) and less edges (about 15,699,276), but is structured in the same manner meaning that is a *Scale-free network*. Hence we expect the same behaviour.

As you can see the result is the same, so exponential grow of infection for both colours and then they contend the same and last part of the network, subject to constant changes. Can be a weak unique part as a multitude of little parts in which a colour cannot predominate permanently, that is why the fluctuation.

Therefore it can be affirmed that this is a general result if the simulation takes place on social networks or anyway networks structured in *Scale-free* manner.

4.2.5 Initial different double seed with the "equo function"

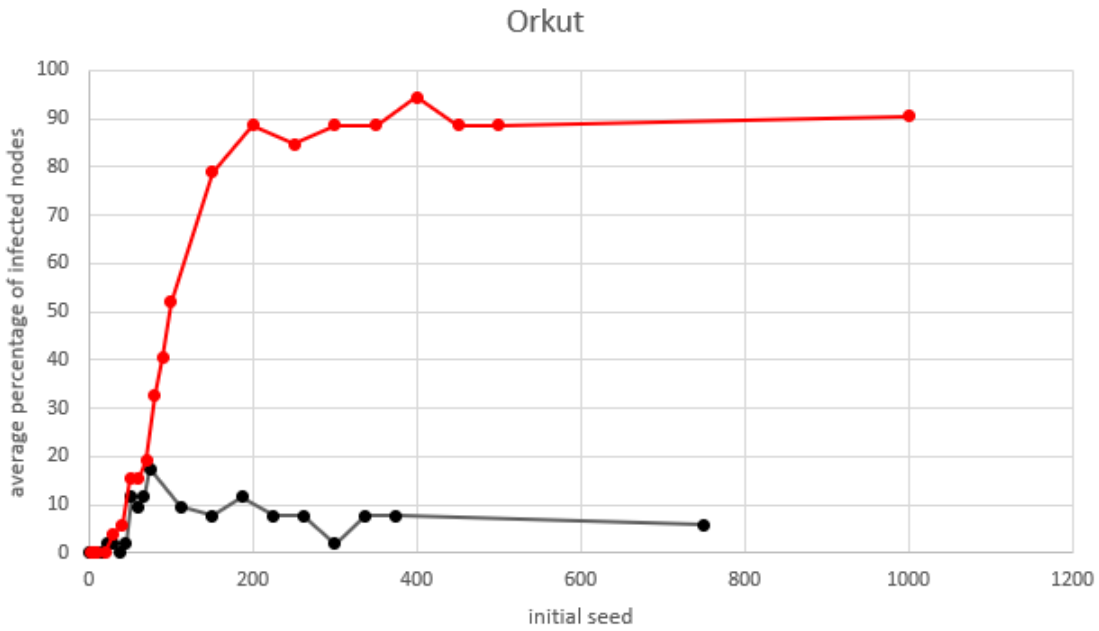


Figure 4.9: Simulation results of a bootstrap percolation on Orkut network, double and not equal initial seed, not permanent decision, neighbour threshold = 2, 50 repetitions per seed, seed with equo function.

Another variant tested is the one in which it is maintained all like before, so the second variant with the difference of not equal initial seeds. For this simulation we are returned to Orkut database and the seed's rule is: $seed_{black} = 3/4 * seed_{red}$. The results are visible in figure 4.9.

From what we can notice initially the behaviour of the two curves of infections seems to be almost the same, so for a little time, until they reach both together almost the 20% of infection, the growth is exponential. At 20% of infection there is a breaking point in which the red colour with higher initial seed starts to get better and continue the exponential growth of its infection. The black colour instead starts to suffer consequently about this aspect, all caused by a significant difference in the initial seed. In this situation the red infection have a higher probability to raise faster the high connected nodes, so it spreads very fast because the absence of a "worthy rival" with enough infection power and no fluctuations between the two competitors happen. This behaviour remains unchanged also in long run.

4.2.6 Initial double seed on a synthetic graph

Another test that was done is to repeat the experiment with double seed and equo function on a synthetic network, constructed like explained in the technical part. The number of nodes present in this network are the same as Orkut (3072441), useful for comparison reasons. For what concern the edges cannot be the same because the fixed construction mechanism of *PA* (Preferential Attachment), so they can be less or more depending which values of m is chosen. Hence what we expect is more or less a similar behaviour as Orkut, depending on the choice of m and *initialClique*. In figure 4.10, figure 4.11 and figure 4.12 it is possible to see the resulting simulations with different values of *initialClique* and m.

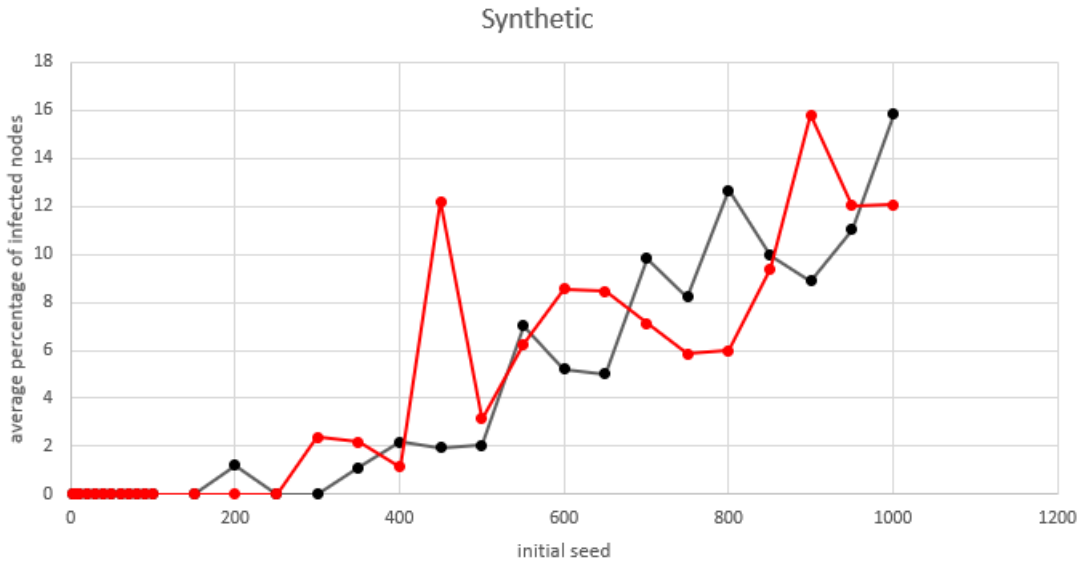


Figure 4.10: Simulation results of a bootstrap percolation on a synthetic network, double and equal initial seed, not permanent decision, neighbour threshold = 2, 50 repetitions per seed, seed with equo function, m=3 and *initialClique*=4.

These three simulations was done with increasing values of m and consequently *initialClique*. How it is immediately possible to notice for all simulations is that we have the usual behaviour in which the two colours try to contend the majority of infection. Hence the result that we can see graphically is the same as previous simulations done with real databases. Also some spikes of infection are possible like in figure 4.10 because the infection mechanism is random and there is a probability that also with low seed we can touch high connected nodes and infect much more than we expect. Looking to the results we observe a not total infection in figure 4.10 because number of edges created ($numNodes * m * 2$) is not sufficient to spread everywhere also using initial seed 1000. Instead in next two simulations increasing the values m and *initialClique* it is possible to reach in both cases practically the

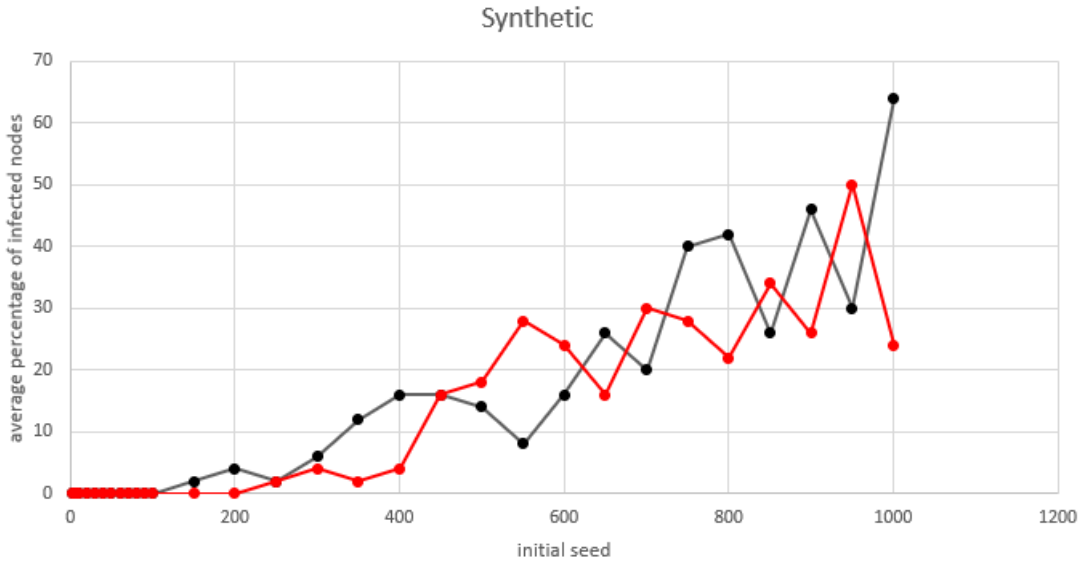


Figure 4.11: Simulation results of a bootstrap percolation on a synthetic network, double and equal initial seed, not permanent decision, neighbour threshold = 2, 50 repetitions per seed, seed with equo function, m=4 and *initialClique*=5.

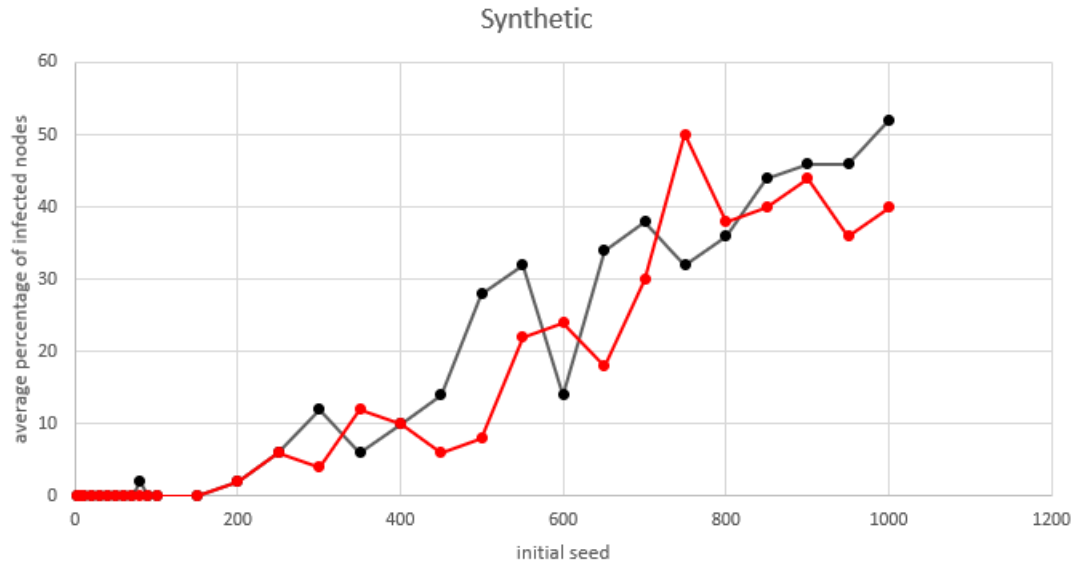


Figure 4.12: Simulation results of a bootstrap percolation on a synthetic network, double and equal initial seed, not permanent decision, neighbour threshold = 2, 50 repetitions per seed, seed with equo function, m=5 and *initialClique*=6.

90-93%. Same results as real databases. A particularity of a synthetic graph is that it is constructed with a fixed mechanism. So observing the results at first impression it is possible to say that it is not present the initial part on the graph in which the infection is spreaded exponentially thanks to very high connected nodes. Hence probably there are hubs which are the most connected of the network like previous simulations, but the the hierarchy of degree is constructed in a pyramidal way so the effect of spreading infection in hubs is not drastic as before. Instead what we can really say is that using same parameters as Orkut, so same number of nodes and edges, we can obtain same results and a 100% of infection, as it possible to observe in figure 4.13.

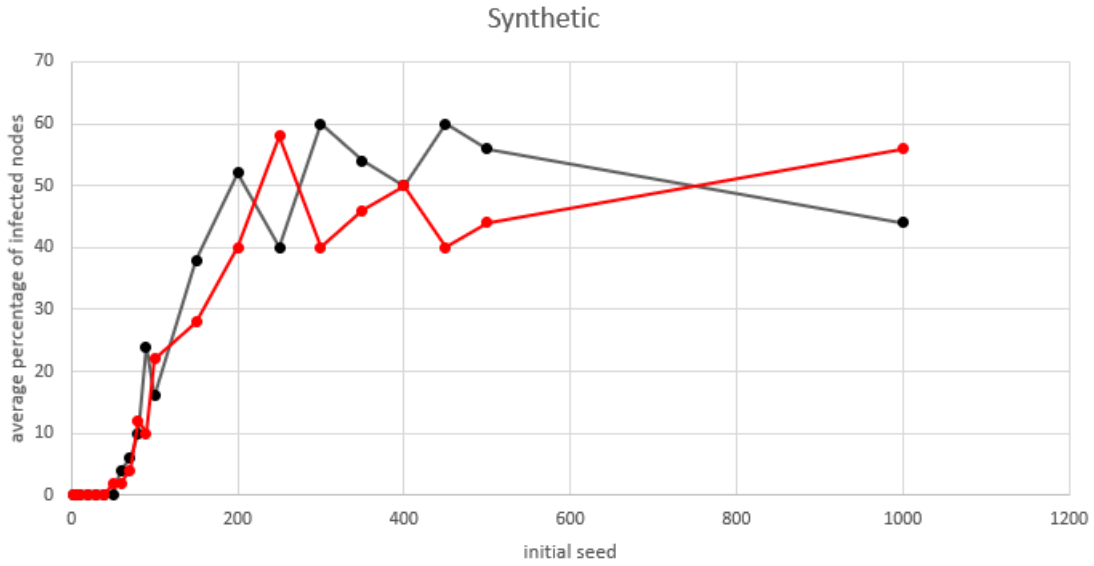


Figure 4.13: Simulation results of a bootstrap percolation on a synthetic network, double and equal initial seed, not permanent decision, neighbour threshold = 2, 50 repetitions per seed, seed with equo function, m=19 and *initialClique*=20.

Only in this case we can reach total infection because we construct the database with a well-know method and all the nodes are connected properly with no isolated nodes or strange behaviours.

Bibliography

[1] Catster/dogster familylinks/frindships network dataset. KONECT [http://konect.uni-koblenz.de/networks/orkut-links], April 2017.

[2] Krishna P. Gummadi Peter Druschel Alan Mislove, Massimiliano Marcon and Bobby Bhattacharjee. Orkut network dataset. In *Measurement and analysis of online social networks. In Proc. Internet Measurement Conf.* KONECT [http://konect.uni-koblenz.de/networks/orkut-links], 2007.

[3] Albert-Laszlo Albert, Reka; Barabasi. Statistical mechanics of complex networks. In *Reviews of Modern Physics*, pages 47–97, 2002.

[4] Albert-Laszlo Albert, Reka; Barabasi. In *Generalized Threshold-Based Epidemics in Random Graphs: the Power of Extreme Values*, 2016.

[5] P. Erdos and A. Renyi. On random graphs. In *Publicationes Mathematicae Debrecen*, pages 290–297, 1959.

[6] Martin Gardner. In *Mathematical Games - The fantastic combinations of Jhon Conway's new solitaire game "life"*, pages 120–123. Scientific American, 1970.

[7] Emilio Leonardi. G(n,p) model. In *Complex network, lecture 12*, page 4. Politecnico di Torino, 2017.

[8] S. Miligram. Pshycology today. In *The Small World Problem*, pages 60–67, 1967.

[9] D. J. Watts. In *Six Degrees: The Science of a Connected Age*. W. W. Norton and Company, 2003.