Monte Carlo simulations of epidemic spreading in the SIS model using Complex Networks

Pablo Higueras

May 12, 2021

The aim of this practise is to develop a Monte Carlo (MC) simulation to study the behaviour of epidemics using complex networks. We will explain the logic that we have followed in our approach and we provide the code developed for the hard computing part in C++ and the light part (network generation and result visualization) in Python.

1 Design of the simulation

1.1 Introduction

The idea of this whole study is simple: create a network of nodes representing people that are connected through different edges, which would usually be contact between two people. Once we have the network, we simulate a whole process in which the nodes evolve in time switching states from infected to susceptible. We will therefore be applying the logic of the SIS epidemic model, widely studied in epidemiology.

1.2 Network generation

To simulate different processes, we have chosen the Erdos-Renyi model as well as scale-free (SF) networks [2]. To generate SF networks we have on the one hand used certain degree distributions to generate particular exponents in the resultant degree distribution, and on the other hand the Barabasi-Albert model, which generates random SF networks with a theoretical exponent γ=3 using preferential attachment. Using these 3 methods, we will create various plots changing different network parameters.
We have selected the Complex Network package NetworkX from the Python language due to simplicity. This package, among many other useful tools,

allows to generate complex random networks with just a couple commands. The different generated networks can be found in the file `networksGeneration.py`. They are saved in Pajek format (useful for many complex network software tools) and .txt format for personal programming.

1.3 Data structures used

The process of running MC simulations is very time consuming, so we have a chosen a compiled language like C++, which can achieve very high computing speeds in comparison to other interpreted high-level languages like Python. We can load all the information of a network from the previously generated .txt files, which contain the number of nodes, edges, and all the node pairs defining all of the edges. Due to compatibility issues with different C++ compiler versions, we define the number of nodes as a constant in the program, though this number can be easily loaded from the files.

There are many data structures useful that can be used, but for our approach we have chosen to use a **map** (which has nothing to do with the functional programming paradigm *map-reduce*). In the C++ Standard Template Library (STL), the **std::map** container works like some sort of Python dictionary, i.e., each element in the data structure is composed of a key-value pair. The key will be an integer, which will be representing each one of the nodes of the network, whereas the value will be a list containing all the adjacent nodes. In other words, the map data structure will work as an **adjacency list**, which is the best way to represent the information of the network.

Furthermore, we create a boolean array which will represent the state of every node, namely true or false values meaning infected or susceptible. The graph stored in the map data structure will remain unaltered and will provide all the information of the network (apart from the binary state of the nodes), whereas all the epidemic dynamics will occur "inside" the binary array. We also create another file containing the information of the adjacency matrix, which will be useful for the theoretical approach. We will explain this later.

1.4 The simulation process

The steps we have taken to simulate the epidemic dynamics are as follows. We define an initial fraction of infected nodes (this will be a simulation parameter, β_0) and then use random number generation (RNG) to infect a particular number of nodes. Then we start the temporal simulation. Each discrete time step has two phases. The first phase consists of iterating trough

every node, checking whether it has any infected neighbours. If there are, then we use RNG to decide if the node will be infected. This is heavily affected by the main parameter of the model β, which is the spreading rate per unit time. The second phase consists of iterating again through every node and using RNG to determine whether the node will change its state (if infected) to susceptible. This is modeled by the recovery parameter per unit time μ. After a transient time of several discrete time steps, the average fraction of infected nodes ρ reaches a stable phase. It is important to note that an auxiliar array is used to preserve the information of the network state at time t without the changes affecting it, so that the infection process depends only on t, and not on $t + 1$.

In order to avoid statistical drawbacks due to RNG, the MC simulation comes in handy, since we can repeat the temporal simulation as many times as necessary to ensure trustworthy results. Each repetition will provide an average fraction of infected nodes, so the final average ρ will be an average (of MC simulation iterations) of averages (of discrete time steps in the stable phase of a single iteration). Therefore, our results after this process has ended will be saved in a text file containing the β values, the average $< \rho >$ and its standard deviation.

This is an expensive and time consuming process, especially when we use large networks and a lot of repetitions. For this reason, we have parallelized sections of the code using the open source tool OpenMP available for different compilers in C/C++/Fortran. To compile our program, one can type in a terminal the command `g++ -fopenmp MC.cpp -o MC` and it will generate an executable file named `MC`.

1.5 Theoretical framework: the MMCA approach

Following [1] and [2], we can use the heterogeneous Mean Field approximation (HMF) prescription to consider that all nodes share the same dynamical properties in the network. The Microscopic Markov-Chain Approach (MMCA) introduces a set of equations that are satisfied by the microscopic variables $p_i(t)$, the probabilities of each node of being infected at a time t. With β the infection rate, μ the recovery probability and $q_i(t)$ the probability of node i not being infected by any neighbour, the SI MMCA equations are

$$p_i(t+1) = (1 - p_i(t))(1 - q_i(t)) + (1 - \mu)p_i(t)$$

$$q_i(t) = \prod_{j=1}^{N} (1 - \beta r_{ij} p_j(t))$$

where r_{ij} is 1 for connected nodes and 0 otherwise. Note that we have taken out the last term of the equations of [1], since we are not considering reinfection in our model in the same time step [2]. We can solve these equations for different networks, iterating for several time steps until a stable phase is reached, just like we do in the simulation. This theoretical approach to the problem gives us the capability of "measuring" how good our implementation of the MC simulation actually is.

2 Results of our model

In this section we present our results, from the generated random networks we have worked with to the different plots obtained using data from both the MC simulations and the theoretical framework.

2.1 General parameters

The parameters that are shared among the different networks are the next:

- $\rho_0 = 0.05$ is the initial probability of a node being infected
- $T = 1000$ is the total number of time steps used for both MC simulations and theoretical results
- $T_{stable} = 500$ is the time step that defines the beginning of the stable phase and therefore sets the start of taking information to compute the average fraction of infected nodes. In general, the stability is reached way before, so 500 is a conservative choice.
- $\mu = 0.1, 0.5, 0.9$ we have chosen these 3 recovery parameters to use in each of the models
- $MC = 50$ is the number of repetitions to run in the MC simulations
- β is the infection probability, each model generated has been run with a set of values ranging from 0.01 to 1, with 100 values in between

2.2 Results for the ER networks

We have generated a network with 500 nodes and p=0.012, i.e., $< k >= 6$; and a network with 2000 nodes and p=0.005, i.e., $< k >= 10$.

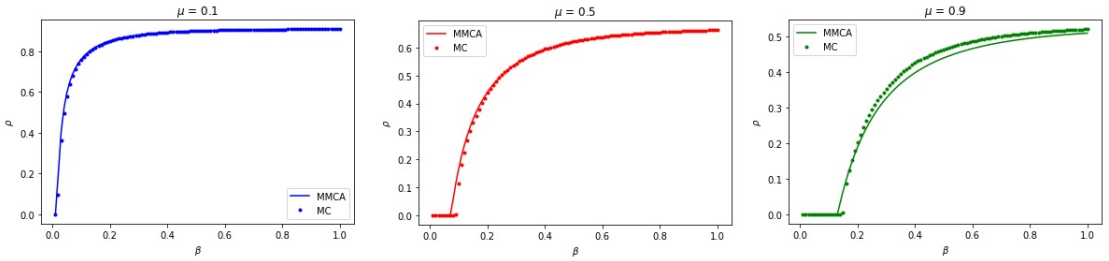

Figure 1: N=500

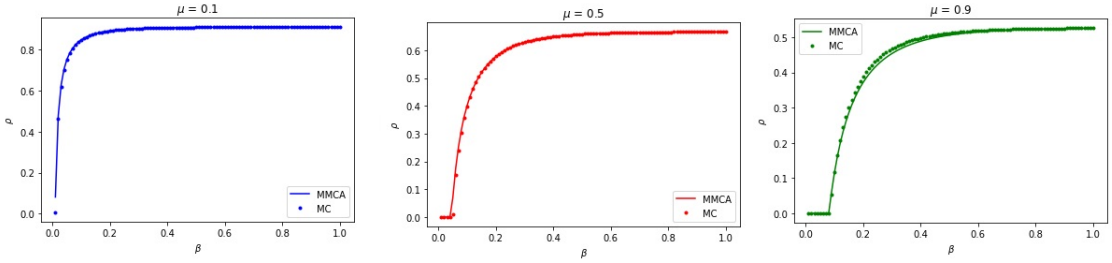

Figure 2: N=2000

2.3 Results for the SF networks

We have generated a network with 500 nodes and $\gamma = 2.7$; and a network with 2000 nodes and $\gamma = 2.5$.

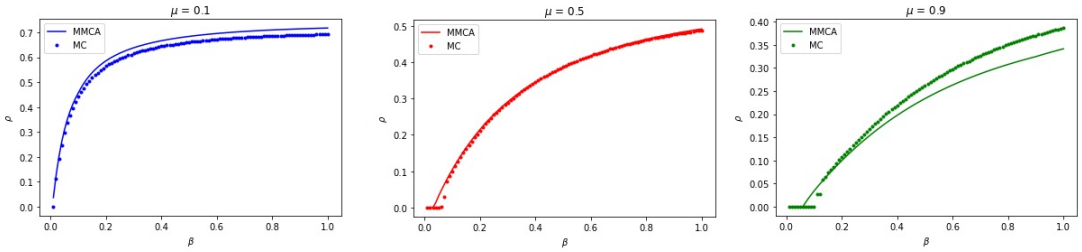

Figure 3: N=500

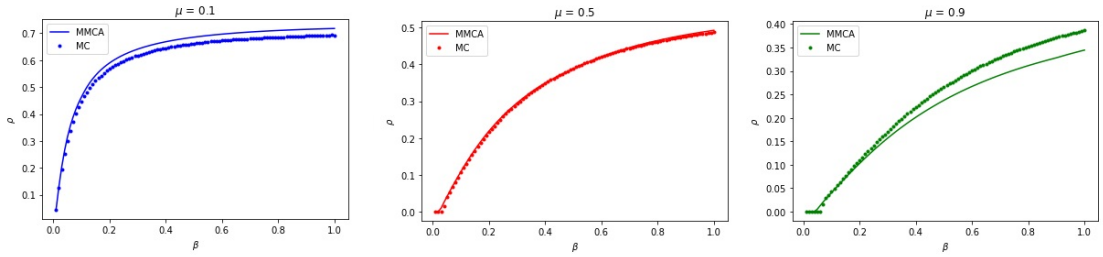

Figure 4: N=2000

2.4 Results for the BA networks

We have generated a network with 500 nodes and m=3; and a network with 2000 nodes and m=2

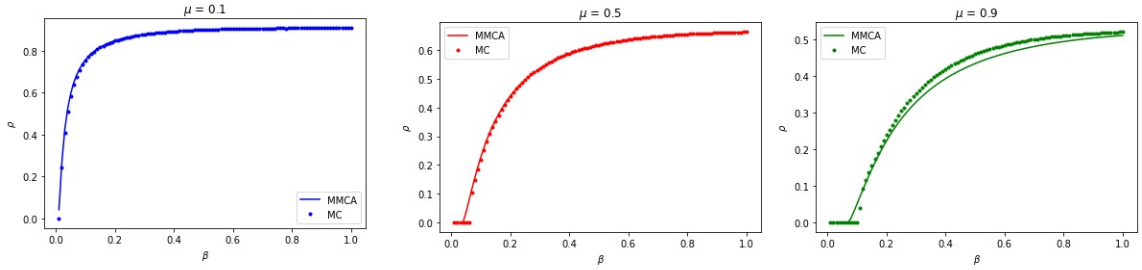

Figure 5: N=500

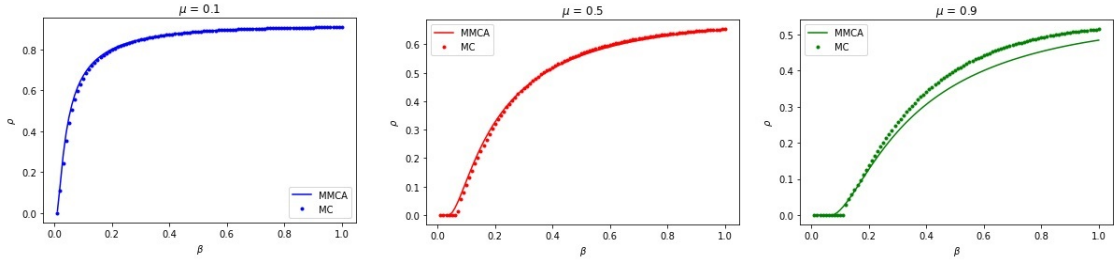

Figure 6: N=2000

2.5 Results for a real network

To close out the study we have used the real network `airportsUW`, which contains information about 3618 airports (nodes) around the world and 14142

edges among them. We have also run the simulation on top of this network using the same 3 recovery parameters.

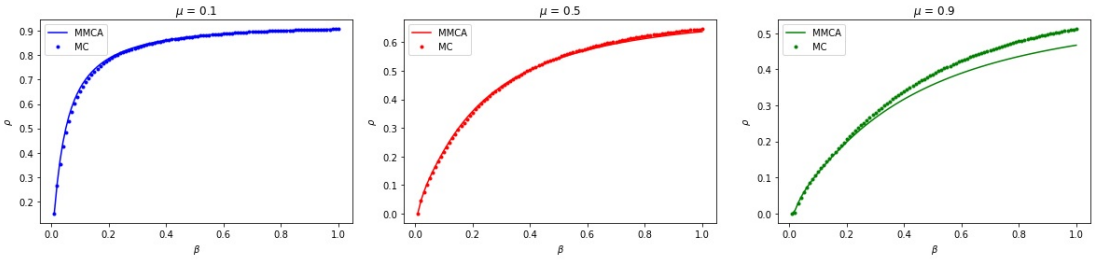

Figure 7: N=2000

3 Conclusions

The results obtained for most networks are completely aligned between theory and simulation. We do have to acknowledge some plots where the differences between theory and simulation are particulary large, as can be seen when the recovery parameter μ is taken close to 1. In general though, both approaches seem to be in agreement in almost all of the β range, except for μ=0.9, which seems to cause a little deviation. This fact might be caused by the elimination of the last term in the MMCA equations, which takes into account reinfection in the same time step.

4 References

[1] Gomez, Sergio & Arenas, Alex & Borge-Holthoefer, Javier & Meloni, Sandro & Moreno, Yamir. (2009). Discrete-time Markov chain approach to contact-based disease spreading in complex networks. EPL. 89. 10.1209/0295-5075/89/38009.
[2] Gomez, Sergio & Arenas, Alex & Borge-Holthoefer, Javier & Meloni, Sandro & Moreno, Yamir. (2011). Probabilistic framework for epidemic spreading in complex networks. Int J Complex Syst Sci. 1. 47-51.