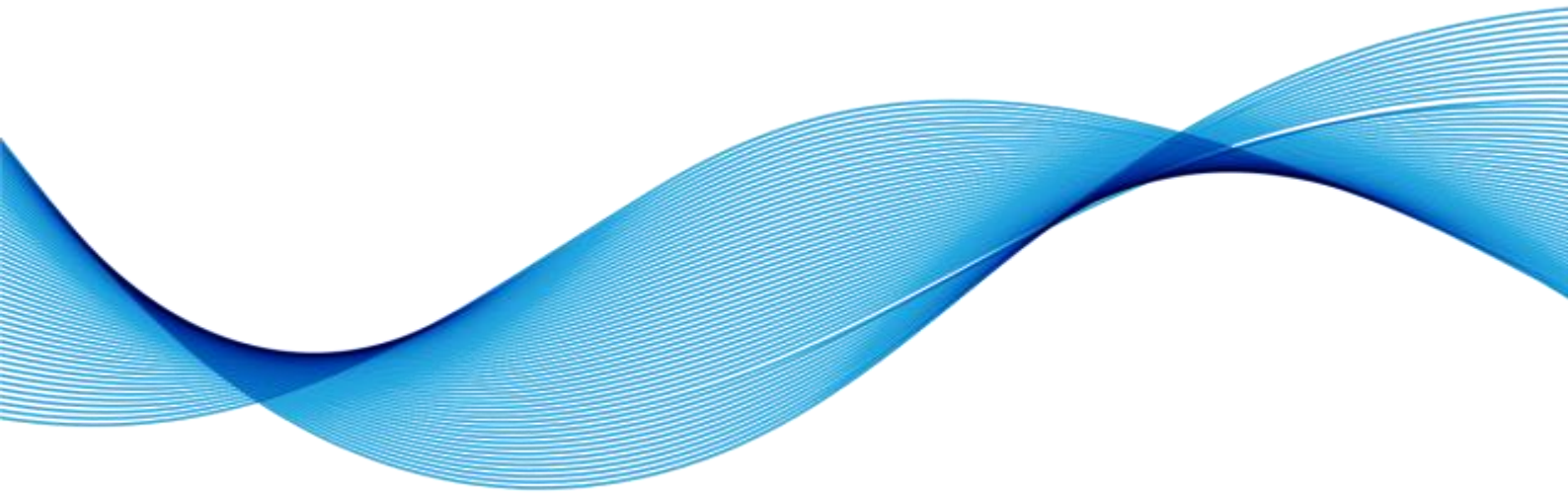
Προσομοίωση και Μοντελοποίηση Συστημάτων



ΤΑΣΟΥΛΑΣ ΘΕΟΦΑΝΗΣ

Ιανουάριος 2019

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΣΑΓΩΓΗ

Η προσομοίωση και μοντελοποίηση είναι η διαδικασία δημιουργίας ενός νοητού ή φυσικού μοντέλου, σχετικά με την λειτουργία ενός συστήματος (πραγματικού ή υπό κατασκευή), καθώς επίσης και η εύρεση της εξίσωσης εξέλιξής του. Αυτό έχει ως σκοπό την κατανόηση της λειτουργίας αυτού και την λήψη αποφάσεων. Η περιγραφή της λειτουργίας των συστημάτων γίνεται με την εισαγωγή κατάλληλων μεγεθών, όπου μπορούν να ποσοτικοποιηθούν. Αυτές λέγονται καταστατικές μεταβλητές, οι οποίες μετράνε μια ποσότητα και αντιστοιχούν μονοσήμαντα σε μια παράμετρο του συστήματος. Για παράδειγμα, προκειμένου να προβλεφθεί η τροχιά της σελήνης, αρκεί να είναι προσδιορίσιμη και γνωστή η θέση και η ταχύτητά της ανά χρονική στιγμή. Επομένως χρειάζονται δυο καταστατικές μεταβλητές, μια που αντιστοιχεί μονοσήμαντα στη θέση ανά χρονική στιγμή και μια στην ταχύτητα ανά χρονική στιγμή της σελήνης.

ΠΕΙΡΑΜΑ 1 – ΜΕΤΑΔΟΣΗ ΙΟΥ ΣΕ ΔΙΚΤΥΟ

Έστω ιός, ο οποίος μπορεί να μεταδοθεί σε ένα δίκτυο με N μοναδικούς κεντρικούς υπολογιστές. Από αυτούς, μόνο οι μη μολυσμένοι κόμβοι S θα μπορούσαν να μολυνθούν. Σε δεδομένο χρόνο t ≥ 0, το σύνολο των πιθανών N διαιρείται σε μολυσμένους I και μη μολυσμένους S, που αντιπροσωπεύονται από I (t) και S (t), αντίστοιχα. I (t) είναι ο αριθμός των κόμβων που έχουν μολυνθεί σε χρόνο t, και S (t) είναι ο αριθμός των κόμβων, που δεν έχουν μολυνθεί την ώρα t. Ο χρόνος στο παρόν πείραμα αναγράφεται με τον αριθμό των επαναλήψεων, που γίνονται στις λούπες του κώδικα. Τουτέστιν, ο χρόνος μπορεί να λάβει οποιαδήποτε λογική μονάδα μέτρησης (δευτερόλεπτα, λεπτά, ώρες, κλπ.). Για χρόνο t = 0 θεωρείται ότι (S (0), I (0)) = (S_0, I_0), όπου I_0 = 1 είναι το αρχικά μολυσμένο πλήθος. Λόγω της τυχαίας συμπεριφοράς διάδοσης ιού για t > 0, οι S (t) και I (t) είναι τυχαίες μεταβλητές. Οι μολυσμένοι κόμβοι διαδίδουν τη μόλυνσή τους, στέλνοντας μολυσμένα πακέτα σε άλλους τυχαία επιλεγμένους κόμβους με σταθερό ρυθμό B, όπου 1 ≥ B ≥ 0. Για όλες τις χρονικές στιγμές t ≥ 0, ισχύει I (t) + S (t) = N. Όταν ένας μολυσμένος κόμβος επιχειρεί να διαδώσει τον ιό με τυχαιότητα B, θεωρείται ότι οι στόχοι του έχουν επιλεγεί με ομοιόμορφη κατανομή. Το πείραμα τελειώνει, όταν μολυνθούν όλοι οι κόμβοι. Επομένως, η διαφορική εξίσωση που επιλύει το πρόβλημα είναι η:

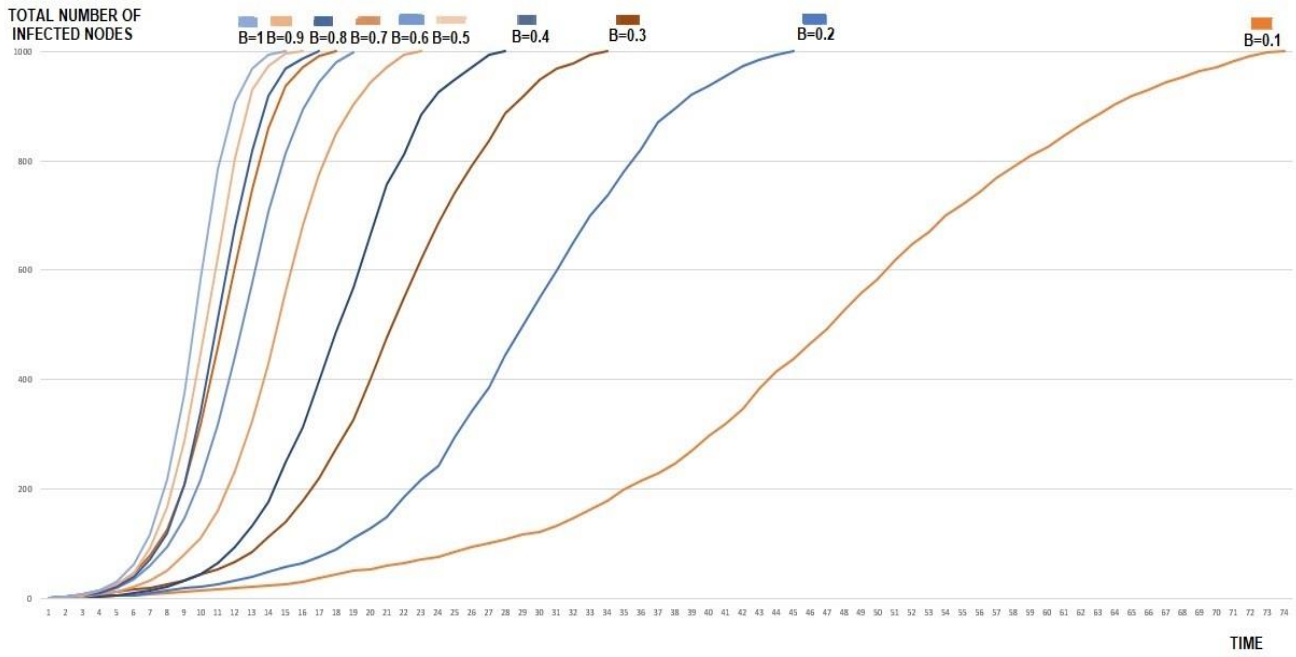
$$\frac{dI}{dt} = \beta \cdot I\left(N - I\right)$$

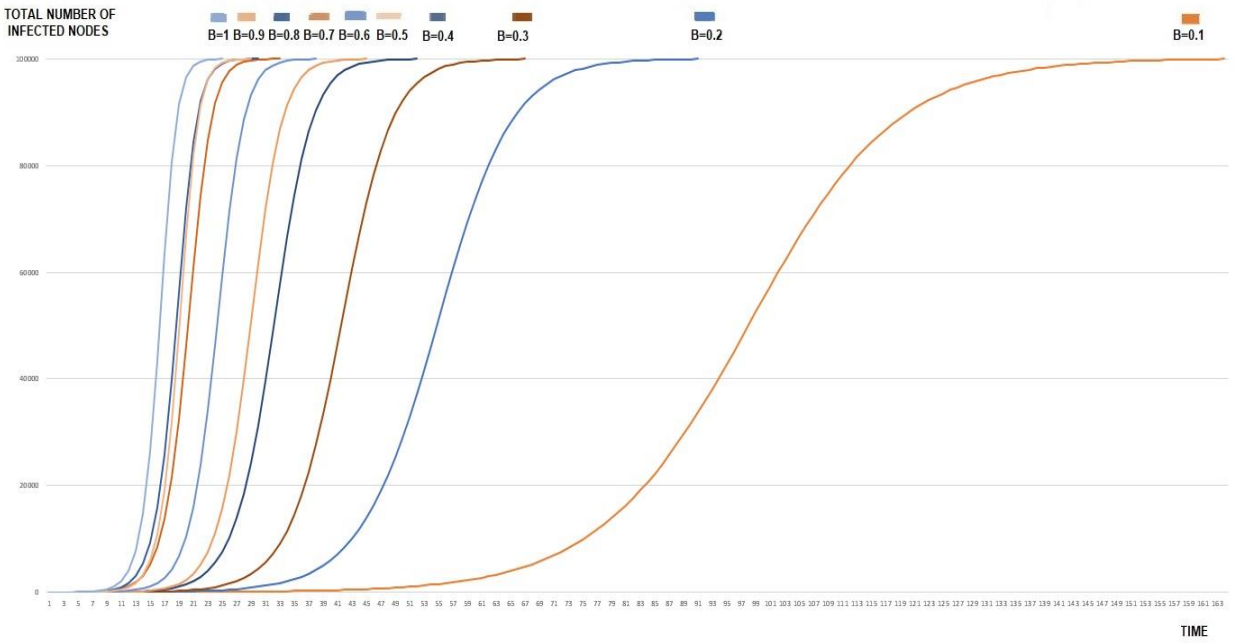
Το παραπάνω πρόβλημα αναπαριστά ο παρακάτω κώδικας, γραμμένος σε γλώσσα C.

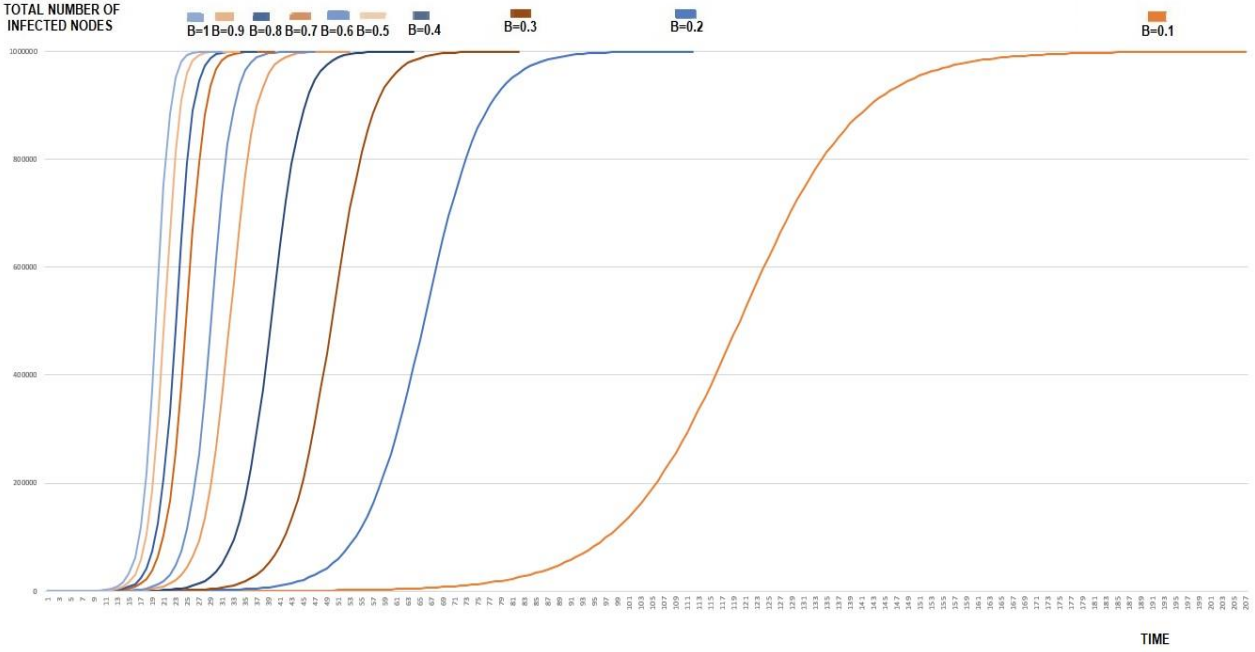
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define NODES 1000000
#define INFECTED 1
#define HEALTHY 0

int main(void)
{
srand(time(NULL));                                  // Randomize the time
FILE *FilePointer = fopen("Result.csv", "w");       // Make a file, to write results on it
float B = 1;                                        // Randomness variable
for (B = 1; B<=10; B++){
   int TotalInfected=1;                             // Num of infected
   int network[NODES]={HEALTHY};                    // All network nodes are healthy
   network[rand() % 1000000] = INFECTED;            // One  network node is infected (radomly)
   int j = 1;                                       // Variable used on the "while" loop
   int time = 0;
   int random;
   float rnd;
   printf("\n\nRadomness B = %0.1f\n", B/10);
   int NITT = 0;                                    // Number of infected this time

   while (TotalInfected < NODES && TotalInfected > 0){
      for (int i = 1; i<=j; i++){
         random = rand() % 1000001;                 // Choose a random network
         rnd = (float) rand() / (float) RAND_MAX; // Choose a random number
         if ( rnd  <=  B/10){                       // If the random number is less than B (where 0.1 < B < 1)
```

```
            if (network[random] == HEALTHY){  // then, if the random network is healthy
               network[random] = INFECTED;      // then make it infected
               TotalInfected++;                 // also add 1 to the Number of infected Networks
               NITT++;                  // also add 1 to the Number of infected Networks this time
            }
         }
      }
      j = TotalInfected;
      time++;
      printf("Time: %d \nNum of Infected: %d \n", time, TotalInfected);
      fprintf(FilePointer, "%d; %d; %d; \n", time, TotalInfected, NITT);
      NITT = 1;                           // re initialize Number of infected Networks this time
   }
}
fclose(FilePointer);
return 0;
}
```

Τρέχοντας τον κώδικα, εξάγονται σε ένα αρχείο .csv οι τιμές των συνολικά μολυσμένων κόμβων I ανά χρονική στιγμή t και των κόμβων που μολύνονται ανά χρονική στιγμή I(t) για τις διάφορες σταθερές B = {0.1, 0.2, ..., 1}. Εάν ξεχωριστούν οι τιμές ανά σταθερά B, τότε μπορούν να δημιουργηθούν οι γραφικές παραστάσεις του συνολικού αριθμού των μολυσμένων κόμβων N ανά χρονική στιγμή t με αριθμό κόμβων N ίσο με 1.000, 100.000, 1.000.000 αντίστοιχα:

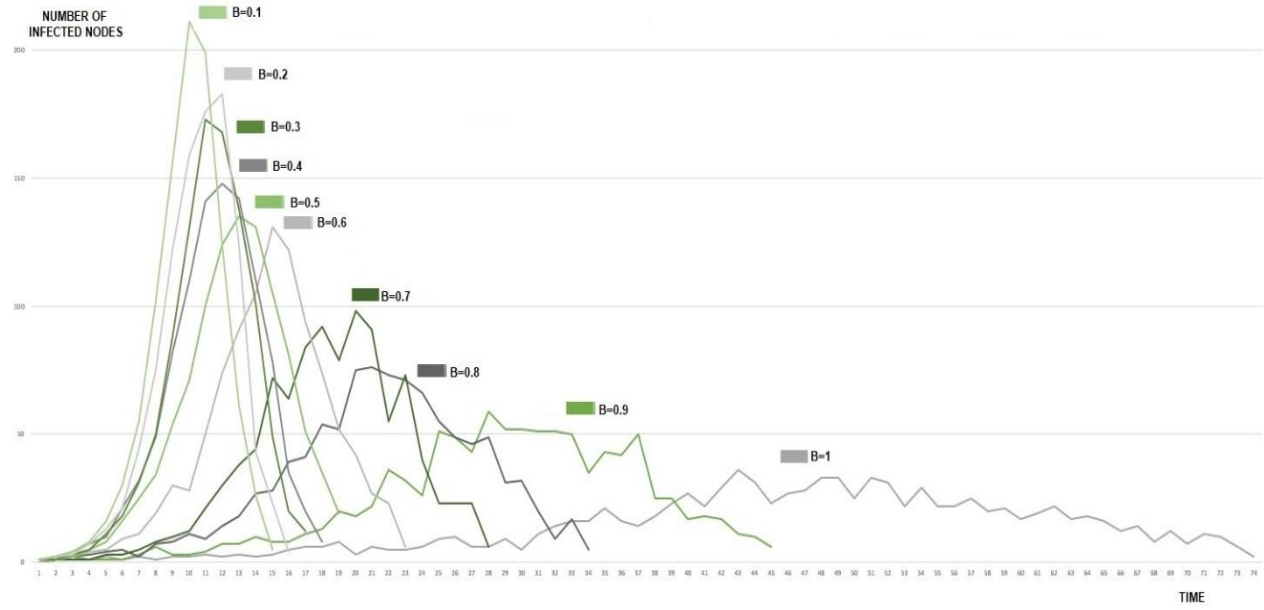


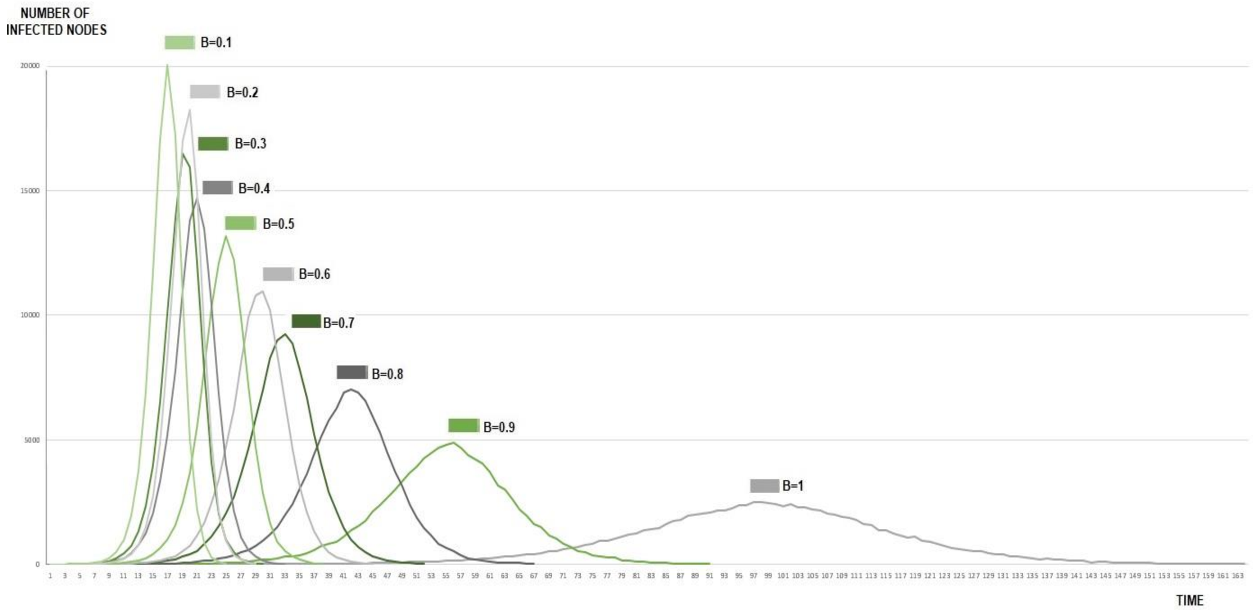


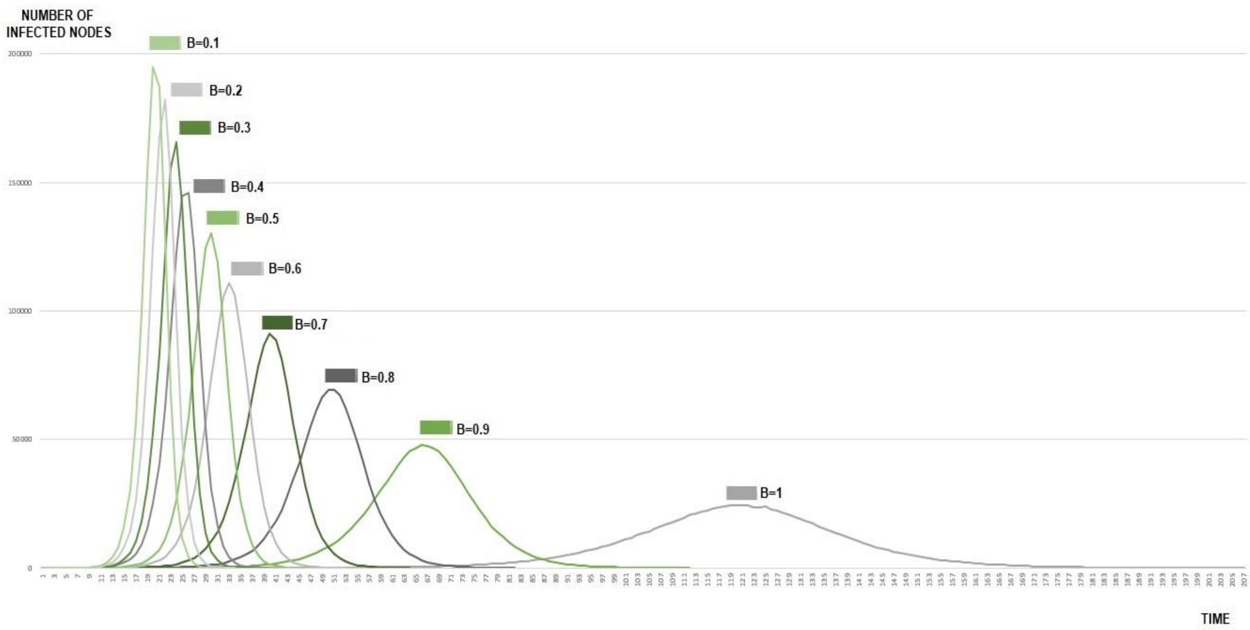


Παρατηρεί κανείς από τις γραφικές παραστάσεις ότι ο αριθμός των μολυσμένων κόμβων, ανεξαρτήτως του ρυθμού B, αρχικά για πολύ μικρό χρόνο t δεν έχει ιδιαίτερα μεγάλη αύξηση, καθώς δεν υπάρχουν ήδη πολλοί μολυσμένοι κόμβοι. Ωστόσο, πολύ σύντομα οι κόμβοι θα αρχίζουν να μολύνονται με απότομο ρυθμό. Και αυτό, επειδή πλέον έχει μολυνθεί ένα σύνολο κόμβων (περίπου το 2.5% με 5% του N), μεγάλο αρκετά για να μολύνει όλους τους υπόλοιπους υγιείς κόμβους. Τέλος, όταν έχουν μολυνθεί σχεδόν όλοι οι κόμβοι, οι υγιείς είναι πολύ λίγοι (περίπου το 2% με 4% του N) και κατά συνέπεια, είναι πιο πιθανό να λάβει το μολυσμένο πακέτο κάποιος μολυσμένος κόμβος, παρά ένας υγιής, αφού η επιλογή κόμβου γίνεται τυχαία. Για αυτό και ο ρυθμός μόλυνσης μειώνεται. Συνεπώς, είναι πολύ λογικό να δημιουργείται μια σιγμοειδής γραφική παράσταση.

Παρακάτω, εμφανίζονται οι γραφικές παραστάσεις των κόμβων που μολύνονται ανά χρονική στιγμή I(t), με αριθμό κόμβων N ίσο με 1.000, 100.000 και 1.000.000 αντίστοιχα.


NUMBER OF
INFECTED NODES
B=0.1
B=0.2
B=0.3
B=0.4
B=0.5
B=0.6
B=0.7
B=0.8
B=0.9
B=1
TIME


NUMBER OF
INFECTED NODES
B=0.1
B=0.2
B=0.3
B=0.4
B=0.5
B=0.6
B=0.7
B=0.8
B=0.9
B=1
TIME


NUMBER OF
INFECTED NODES
B=0.1
B=0.2
B=0.3
B=0.4
B=0.5
B=0.6
B=0.7
B=0.8
B=0.9
B=1
TIME

Στην αρχή ο αριθμός των κόμβων που μολύνονται ανά χρονική στιγμή I(t) (περίπου για το 27%-32% του χρόνου t) είναι μικρός, καθώς λίγοι είναι οι ήδη μολυσμένοι I και δεν είναι τόσο εύκολο να εξαπλώσουν λίγοι κόμβοι τον ιό. Μετά όμως, αυξάνει απότομα ο I(t) (περίπου για το 26%-32% του χρόνου t), αλλά ως ένα σημείο. Μετά από αυτό το σημείο, ο ρυθμός αύξησης μειώνεται με ίδιο τρόπο όπως ο αμέσως προηγούμενός του ρυθμός (περίπου για το 26%-32% του χρόνου t), σχηματίζοντας έτσι μια καμπάνα στη γραφική παράσταση. Το σκαμπανέβασμα αυτό είναι λογικό για τους ίδιους λόγους με της προηγούμενης περίπτωσης, όπως επίσης και το τέλος. Στο τέλος, ο I(t) μειώνεται με αργό ρυθμό (περίπου το 7%-12% του χρόνου t).

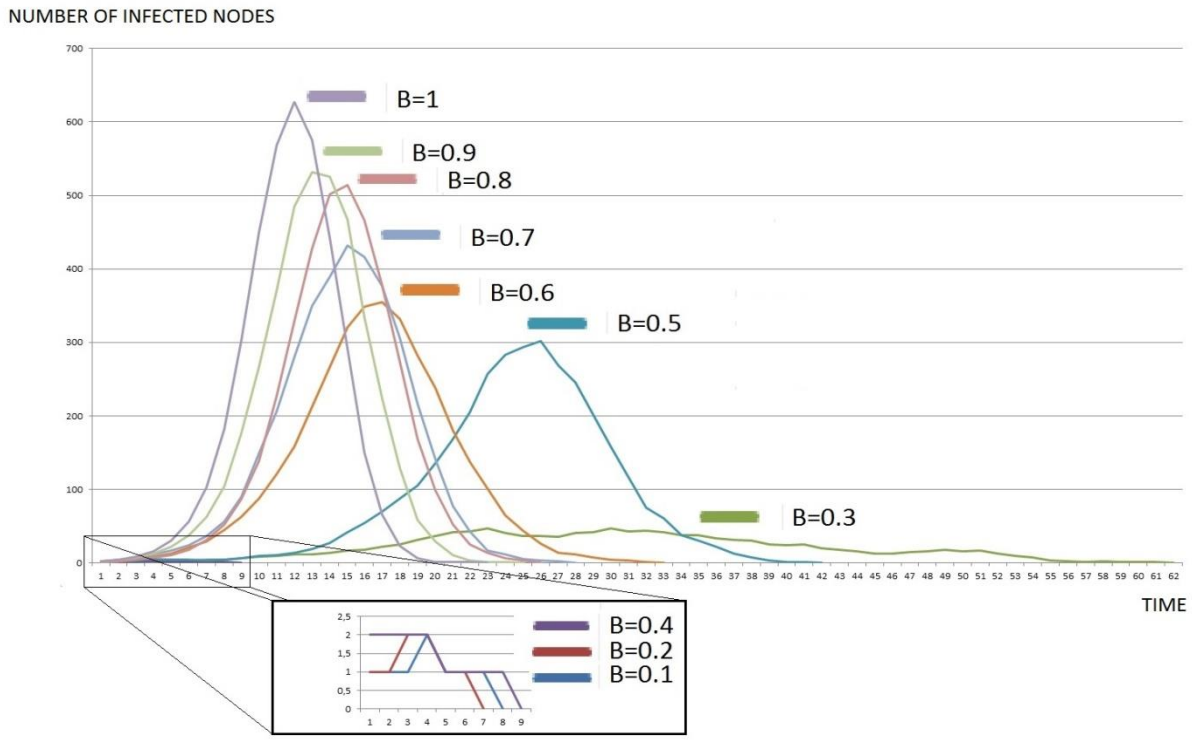
ΠΕΙΡΑΜΑ 2 – ΜΕΤΑΔΟΣΗ ΙΟΥ ΚΑΙ ΙΑΣΗ ΜΟΛΥΝΣΜΕΝΩΝ ΚΟΜΒΩΝ ΣΕ ΔΙΚΤΥΟ

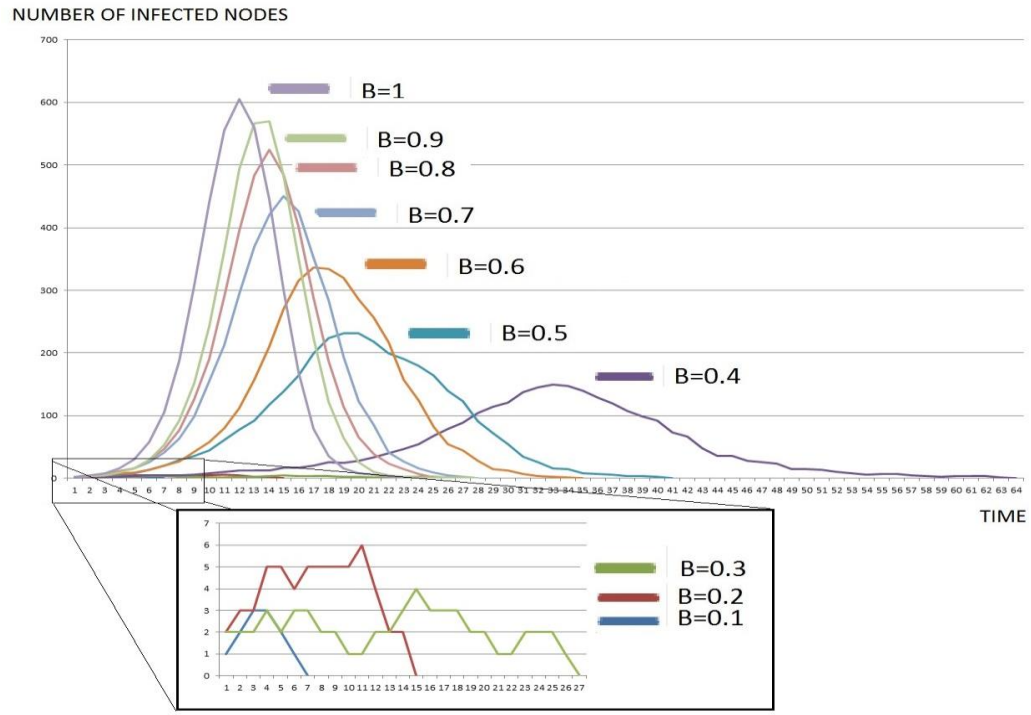
Τώρα η πολυπλοκότητα του πειράματος αυξάνεται με την προσθήκη του ανθρώπινου παράγοντα. Πλέον, κάθε μολυσμένος κόμβος, μετά από χρόνο t=5 θα ανοσοποιείται και δε θα μπορεί να μολυνθεί ξανά (ένας το πολύ ανά χρονική στιγμή). Το πείραμα τελειώνει, όταν μολυνθούν ή ανοσοποιηθούν όλοι οι κόμβοι και υλοποιείται ως εξής:

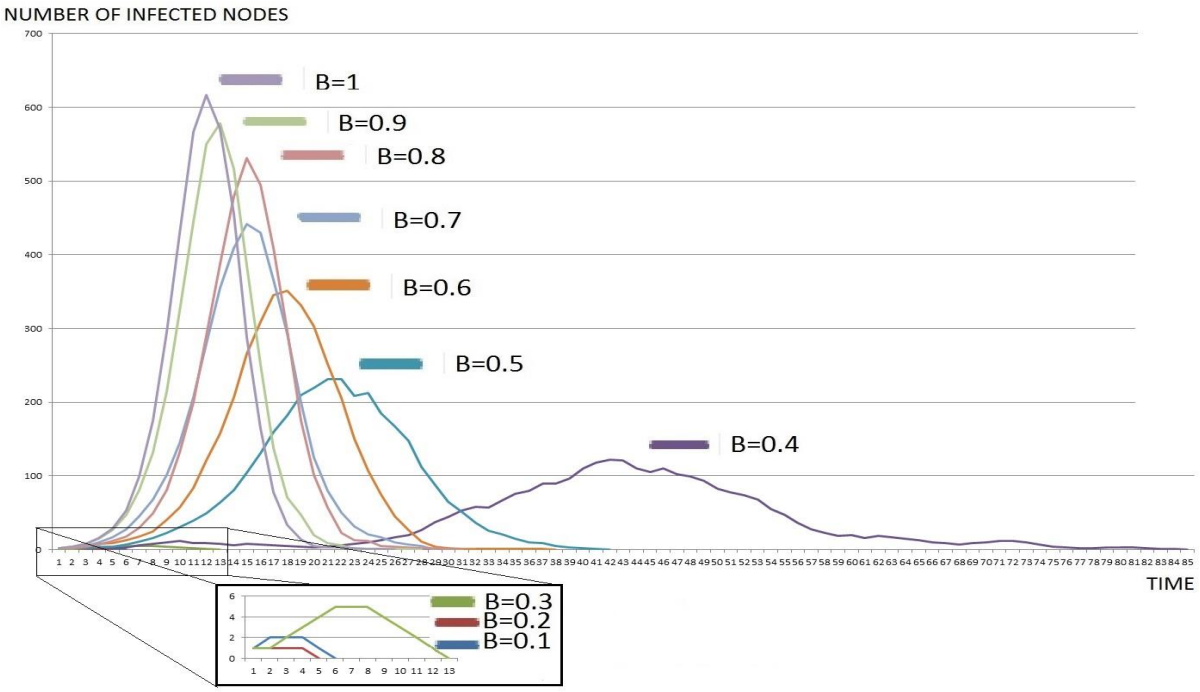
```
typedef struct NODESTRUCT{
int state;
int healingTime;
} Node;

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define NODES 1000000
#define HEALTHY  0
#define INFECTED 1
#define IMMUNED  2

int main(void){
srand(time(NULL));                                  // Randomize the time
FILE *FilePointer = fopen("Result2.csv", "w");     // Make a file, to write results on it
float B = 1;                                       // Randomness variable

for (B = 1; B<=10; B++){
   Node network[NODES];
   for (int i = 0; i<NODES; i++){                   // Initialize
      network[i].state = HEALTHY;                   // All network nodes are healthy and have healing
time=5
      network[i].healingTime = 5;
   }
   network[rand() % NODES].state = INFECTED;// One  network node is infected (radomly)
   int TotalInfected=1;                             // Num of Total infected
   int TotalImmunded=0;                             // Num of immuned
   int j = 1;                                       // Variable used on the "while" loop
   int time = 0;
   int random;
   float rnd;
   printf("\n\nRadomness B = %0.1f\n", B/10);
   int NITT = 0;                                    // Number of infected this time

   while (TotalInfected < NODES && TotalInfected > 0){

      for (int i = 1; i<=j; i++){
       random = rand() % 1000;                            // Choose a random network
            rnd = (float) rand() / (float) RAND_MAX;  // Choose a random number
            if ( rnd  <=  B/10){                 // If the random number is less than B (where 0.1 < B < 1)
               if (network[random].state == HEALTHY){  // then, if the random network is healthy
                  network[random].state = INFECTED;      // then make it infected
                  TotalInfected++;           // also add 1 to the Number of infected Networks
                  NITT++;                   // also add 1 to the Number of infected Networks this time
```

```
                    //if (TotalInfected==NODES){break;}  // If all networks get infected, exit the loop
                }
            }
        }
        for (int node=0; node <= NODES; node++){
           if (network[node].state == INFECTED){
              network[node].healingTime--;
              if(network[node].healingTime==0 ){
                 network[node].state=IMMUNED;
                 TotalImmunded++;
                 TotalInfected--;
              }
           }
        }
        j = TotalInfected;
        time++;
        printf("Time: %d \nInfected: %d Immuned: %d\n", time, TotalInfected, TotalImmunded);
        fprintf(FilePointer, "%f; %d; %d;\n", B, TotalInfected, TotalImmunded);
        NITT = 1;                              // re initialize Number of infected Networks this time
    }
}
fclose(FilePointer);
return 0;}
```

Τρέχοντας το 2ο κώδικα εξάγονται οι τιμές των συνολικά μολυσμένων ανά χρονική στιγμή t, για τις διάφορες σταθερές B = {0.1, 0.2, ..., 1} με αριθμό κόμβων N=1.000, N=10.000 και N=1.000.000 αντίστοιχα.

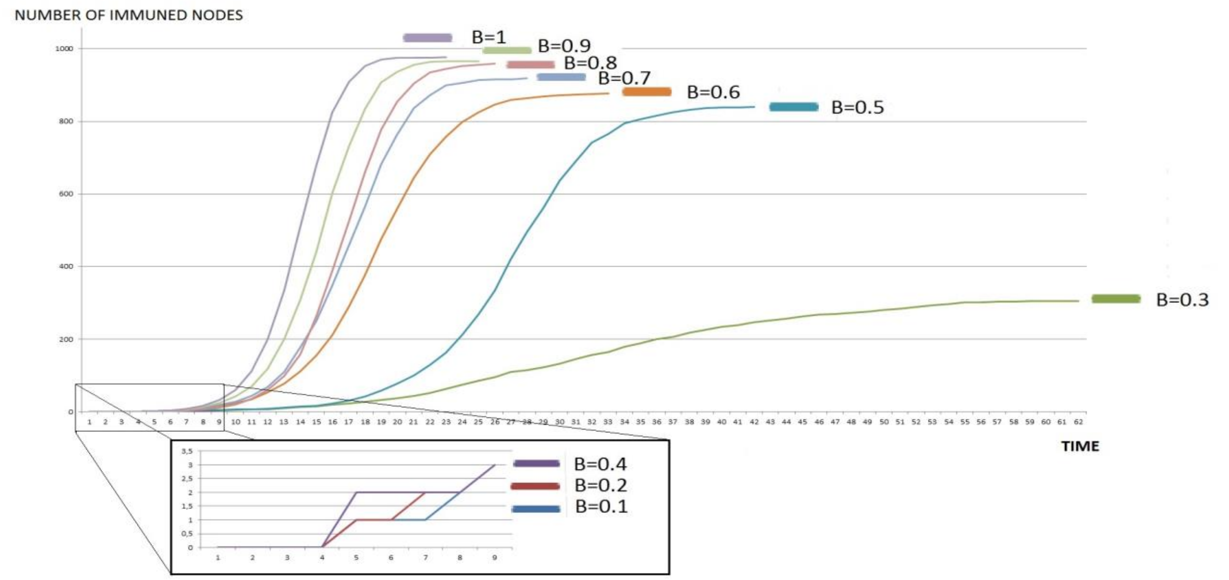


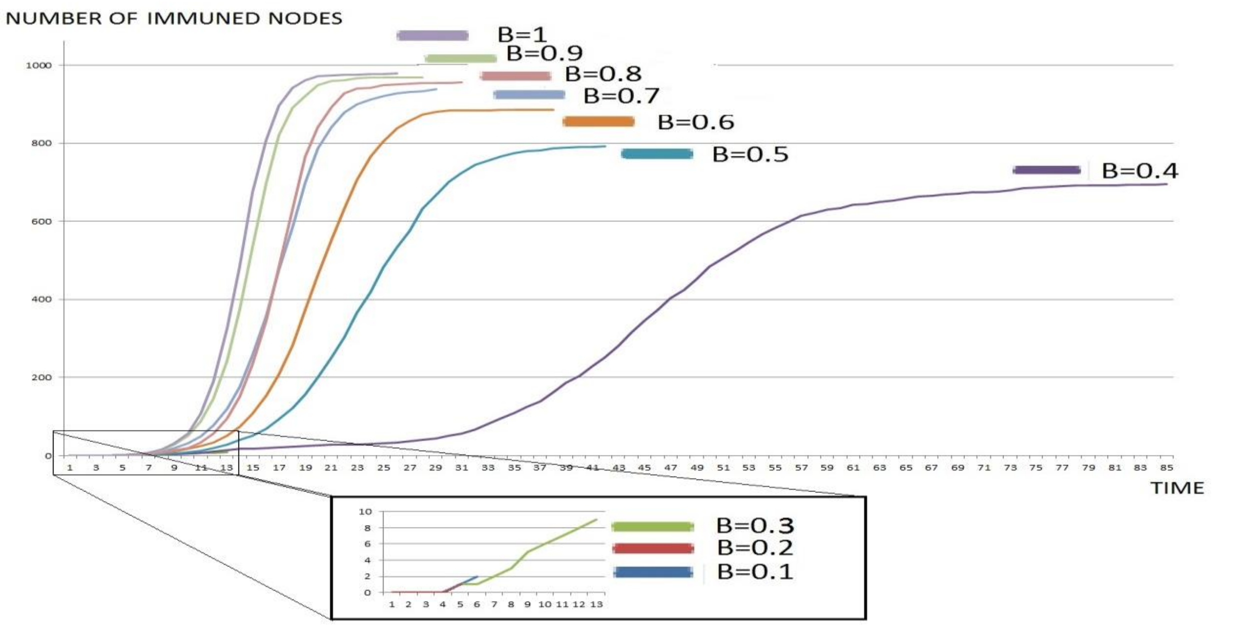


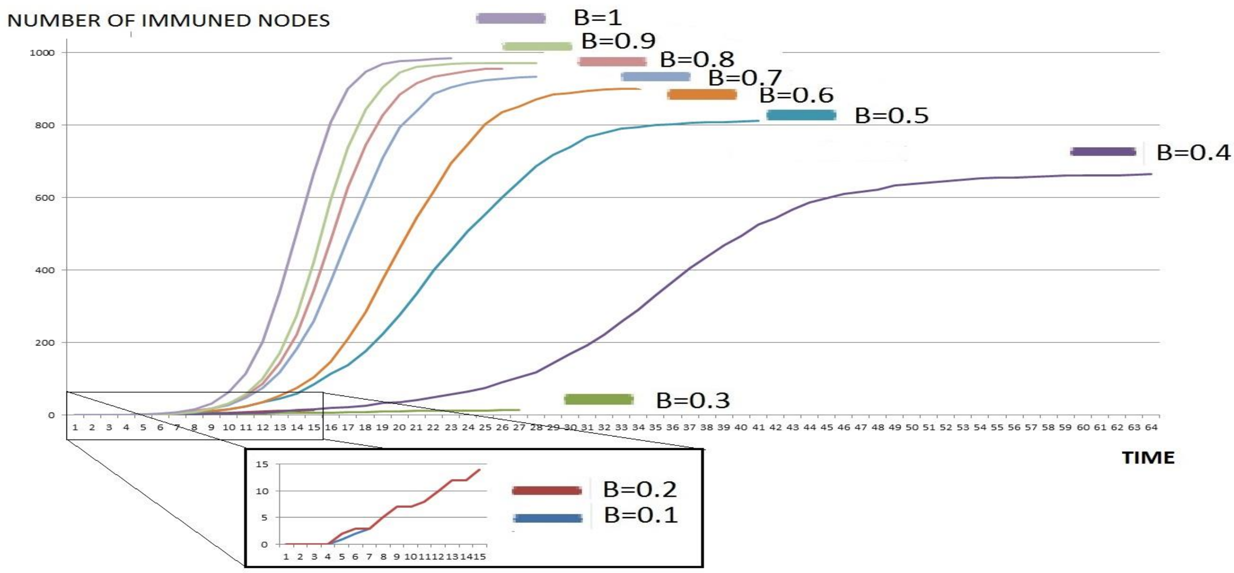


Μια παρατήρηση από τα παραπάνω είναι ότι, όσο μεγαλύτερο είναι το Β, τόσο μεγαλύτερες καμπάνες σχηματίζονται. Τούτο διότι η πιθανότητα μόλυνσης είναι μεγάλη και τελικά περισσότεροι κόμβοι μολύνονται. Αντιθέτως, για μικρό Β δεν προλαβαίνει ο ιός να εξαπλωθεί και ήδη ανοσοποιούνται οι μολυσμένοι κόμβοι. Έτσι αιτιολογούνται και οι πολύ μικρές καμπάνες, οι οποίες μεγεθύνονται στις παραπάνω εικόνες για διευκόλυνση. Σε αυτές, αρχικά μολύνονται λίγοι κόμβοι. Όμως, στη συνέχεια ιατρεύονται όλοι τους και κατά συνέπεια ο ιός σε σύντομο χρονικό διάστημα εξαλείφεται. Ένα επιπρόσθετο αξιοπρόσεκτο είναι η ισότητα του ρυθμού μόλυνσης με το ρυθμό ανοσοποίησης. Τα αποτελέσματα είναι παραπλήσια, σε παρόμοιο χρόνο, ανεξαρτήτου αριθμού συνολικών κόμβων.

Με την ίδια διαδικασία αναπαράγονται οι γραφικές παραστάσεις των συνολικά ανοσοποιημένων κόμβων, συναρτήσει χρόνου I(t), με αριθμό κόμβων N ίσο με 1.000, 100.000 και 1.000.000 αντίστοιχα.







Οι συνολικά μολυσμένοι κόμβοι αναπαρίστανται με σιγμοειδή μορφή για τους ίδιους λόγους του προηγούμενου πειράματος. Όσο μικρότερο Β υπάρχει, τόσο πιο λίγοι κόμβοι χρειάζονται ανοσοποίηση. Υπό τις τωρινές συνθήκες, φαίνεται ότι σε καμία περίπτωση δεν προλαβαίνουν να μολυνθούν όλοι οι κόμβοι. Αντιθέτως, ο αλγόριθμος πάντοτε τελειώνει με την ίασή τους.

ΠΕΙΡΑΜΑ 3 – ΜΕΤΑΔΟΣΗ ΠΛΗΡΟΦΟΡΙΑΣ ΣΕ ΔΙΚΤΥΟ

Η επόμενη προσομοίωση αναφέρεται στη μετάδοση πληροφορίας σε ένα δίκτυο, το οποίο έχει σχήμα δισδιάστατου πίνακα (σαν ένα κελί με οριζόντιους και κάθετους κόμβους). Ξεκινώντας, επιλέγεται τυχαία ένας κόμβος και λαμβάνει μια πληροφορία (π.χ. ότι υπάρχει πυρκαγιά). Σε κάθε χρονική στιγμή t επιλέγεται ένας τυχαίος γειτονικός κόμβος και λαμβάνει την πληροφορία από τον προηγούμενο κόμβο, που βρίσκονται μεταξύ τους σε απόσταση R (όπου R=1,2,…,10). Ο αλγόριθμος συνεχίζει, μέχρις ότου όλοι οι κόμβοι να λάβουν την πληροφορία. Εάν σε έναν κόμβο μεταδοθεί η πληροφορία, ενώ την έχει ήδη λάβει, δε συμβαίνει τίποτε. Η διαφορική εξίσωση του προβλήματος δίνεται από τη σχέση:

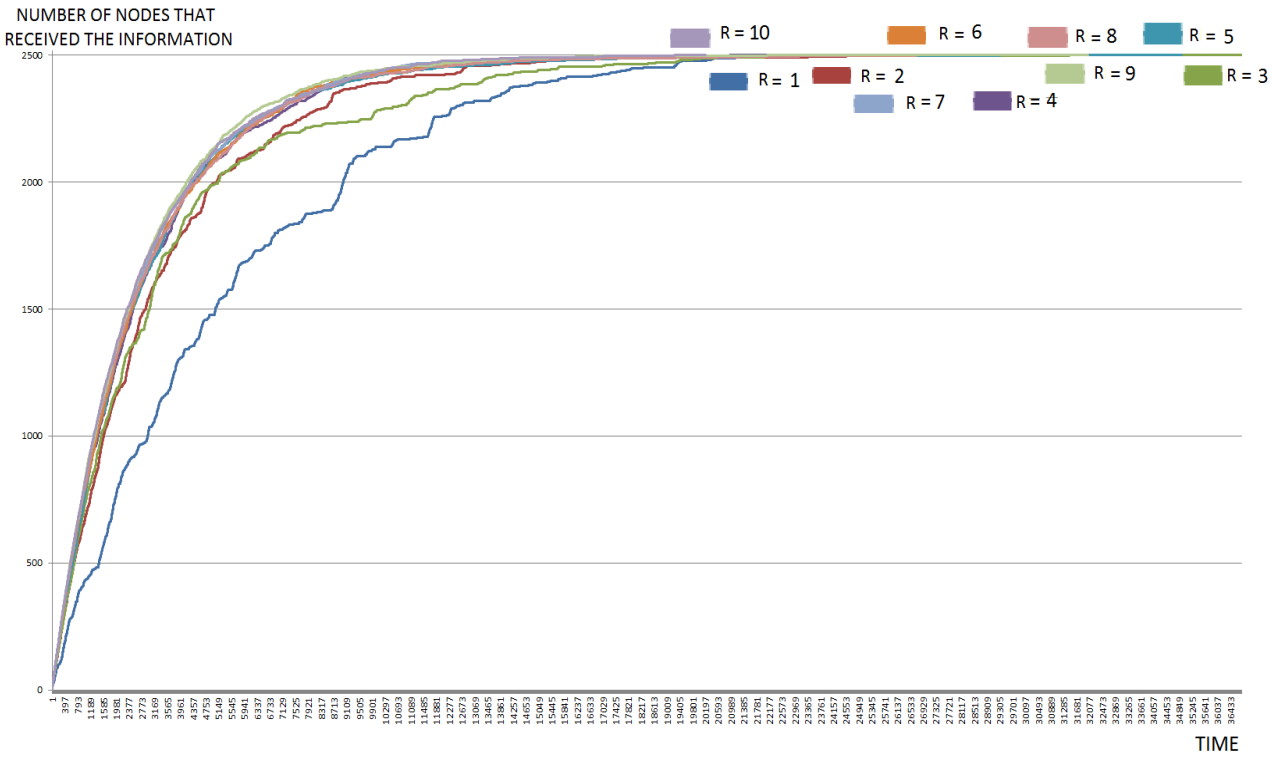
$$\frac{dI}{dt} = \beta \cdot (N - I)$$

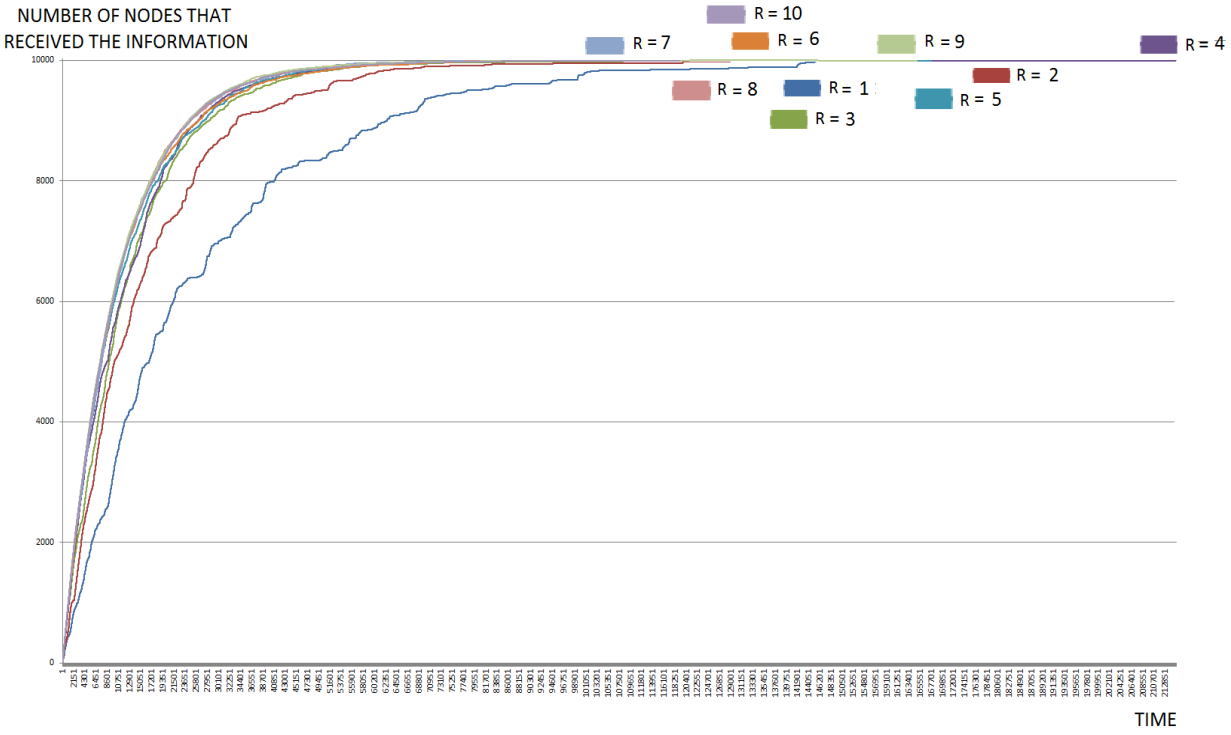
Ο παρακάτω αλγόριθμος απεικονίζει τη μετάδοση πληροφορίας σε δίκτυο.

```
typedef struct sensor{
int flag;
int radius;
float energy;
}Sensor;

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define NODES 10

int getRandomCoordinate(int coordinate, int radius, int coordinateMax);

int main(void){
srand(time(NULL));                    // Randomize the time
int time = 0;
float B = 1;                          // Randomness variable
int radius=1;                         // Radius variable
char radiusChar[6];                   // Just a variable to contain characters

for (radius=1; radius<=100; radius++){
   radius = radius+9;
   sprintf(radiusChar, "%d", radius);          // Convert int to char
   strcat(radiusChar, ".csv");                 // radiusChar + ".csv"
   FILE *FilePointer = fopen(radiusChar, "w");    // Make a file, to write results on it
   Sensor sensors[NODES][NODES]={0,-1,-1};
   sensors[rand()%NODES][rand()%NODES].flag = 1;
   int updatedSensor = 1;
   int currentSensorX = rand()%NODES;
   int randomSensorX = -1;
   int currentSensorY = rand()%NODES;
   int randomSensorY = -1;

   while(updatedSensor<NODES*NODES){
      do{
         randomSensorX=getRandomCoordinate(currentSensorX, radius, NODES);
         randomSensorY=getRandomCoordinate(currentSensorY, radius, NODES);
```

```
        }while(randomSensorX==currentSensorX);
        currentSensorY = randomSensorY;
        currentSensorX = randomSensorX;
        if(!sensors[currentSensorX][currentSensorY].flag){
           updatedSensor+=1;
           sensors[currentSensorX][currentSensorY].flag = 1;
        }
        time++;
        fprintf(FilePointer, "%d;\n", updatedSensor); //How many sensors got the message?
      }
   updatedSensor = 1;
   fclose(FilePointer);
}
   return 0;
}

int getRandomCoordinate(int coordinate, int radius, int coordinateMax){
   //one random value for one     coordination. x or y.
   int min = (coordinate-radius)<0?0:coordinate-radius;                    //if <0 then
   int max = (coordinate+radius)>coordinateMax?coordinateMax:coordinate+radius;
   return  min + rand()%((max-min)+1);
}
```

Εξάγονται λοιπόν δέκα αρχεία, με τον αριθμό των κόμβων που έλαβαν την πληροφορία ανά χρονική στιγμή για δέκα διαφορετικές αποστάσεις R. Οι παρακάτω γραφικές παραστάσεις αναπαριστούν τις προαναφερθείσες τιμές, όταν οι κόμβοι είναι συνολικά 2500 και 10.000 αντίστοιχα. Τέλος, ας σημειωθεί ότι ο αριθμός της ακτίνας απεικονίζεται με τη χρονική σειρά που τελείωσε ο αλγόριθμος από τα αριστερά προς τα δεξιά (δηλαδή, πρώτα τελείωσε ο αλγόριθμος για R=10, μετά για R=1, μετά για R=2 και ούτω καθεξής).





Διαπιστώνεται ότι αρχικά (περίπου κατά τη μετάδοση στο 5% με 10% των κόμβων) ο ρυθμός μετάδοσης είναι σχετικά ίσος με το χρόνο και στη συνέχεια, σιγά σιγά ο ρυθμός μετάδοσης μειώνεται δραματικά. Αυτό συμβαίνει, διότι στην αρχή οι περισσότεροι κόμβοι δεν έχουν λάβει την πληροφορία, άρα είναι εύκολο να επιλεχθεί ένα κόμβος τυχαία που δεν του έχει μεταδοθεί η πληροφορία. Μετά όμως, είναι δύσκολο να επιλεχθούν οι κόμβοι που δεν έχουν λάβει την πληροφορία, γιατί αυτοί είναι λίγοι και υπάρχει τυχαιότητα στη μετάδοση της πληροφορίας. Φαίνεται ακόμα ότι, όσο μικρότερη ακτίνα R υπάρχει, τόσο μικρότερος είναι ο ρυθμός μετάδοσης της πληροφορίας. Μολαταύτα, η χρονική στιγμή που θα λάβουν όλοι οι κόμβοι την πληροφορία είναι ανεξάρτητος από την απόσταση R, φαινόμενο μη λογικό. Το προσδοκώμενο αποτέλεσμα θα ήταν, όταν υπάρχει μικρότερη απόσταση R, να χρειάζεται περισσότερος χρόνος ολοκλήρωσης του αλγορίθμου, αφού μειώνεται η δυνατότητα εξάπλωσης της πληροφορίας. Όμως η τυχαιότητα μετάδοσης είναι απρόβλεπτη και ανεξάρτητη από άλλους παράγοντες, γεγονός που επαληθεύει τα παρόντα αποτελέσματα. Τέλος, διαπιστώνεται ότι, όσους λιγότερους κόμβους έχει το δίκτυο, τόσο λιγότερος χρόνος χρειάζεται για την μετάδοση της πληροφορίας σε όλους τους κόμβους, γεγονός λογικό.

ΠΕΙΡΑΜΑ 4 – ΠΡΟΣΟΜΟΙΩΣΗ ΣΕΙΣΜΟΥ ΜΕ ΤΟ ΜΟΝΤΕΛΟ OFC

Το τέταρτο και τελευταίο μέρος αναφέρεται στην προσομοίωση σεισμών με το απλό μοντέλο OFC. Μέσα σε κάποιο συγκεκριμένο χρονικό διάστημα t, λόγω των διαφόρων κινήσεων της γης και των εχόντων αυτής, το κάθε σημείο του εδάφους διοχετεύει δύναμη Fout με την πάροδο του χρόνου. Όταν η δύναμη αυτού του σημείου ξεπερνάει ένα συγκεκριμένο όριο Fcrit, τότε το σημείο αυτό κινείται και ασκεί μια δύναμη στα γειτονικά του σημεία. Στην παρούσα περίπτωση το σημείο ασκεί το ένα τέταρτο της δύναμης που διοχέτευσε. Συνεπώς, σε κάθε χρονική στιγμή ένα σύνολο σημείων θα κινούνται, δημιουργώντας έναν σεισμό. Η διαφορική εξίσωση είναι:

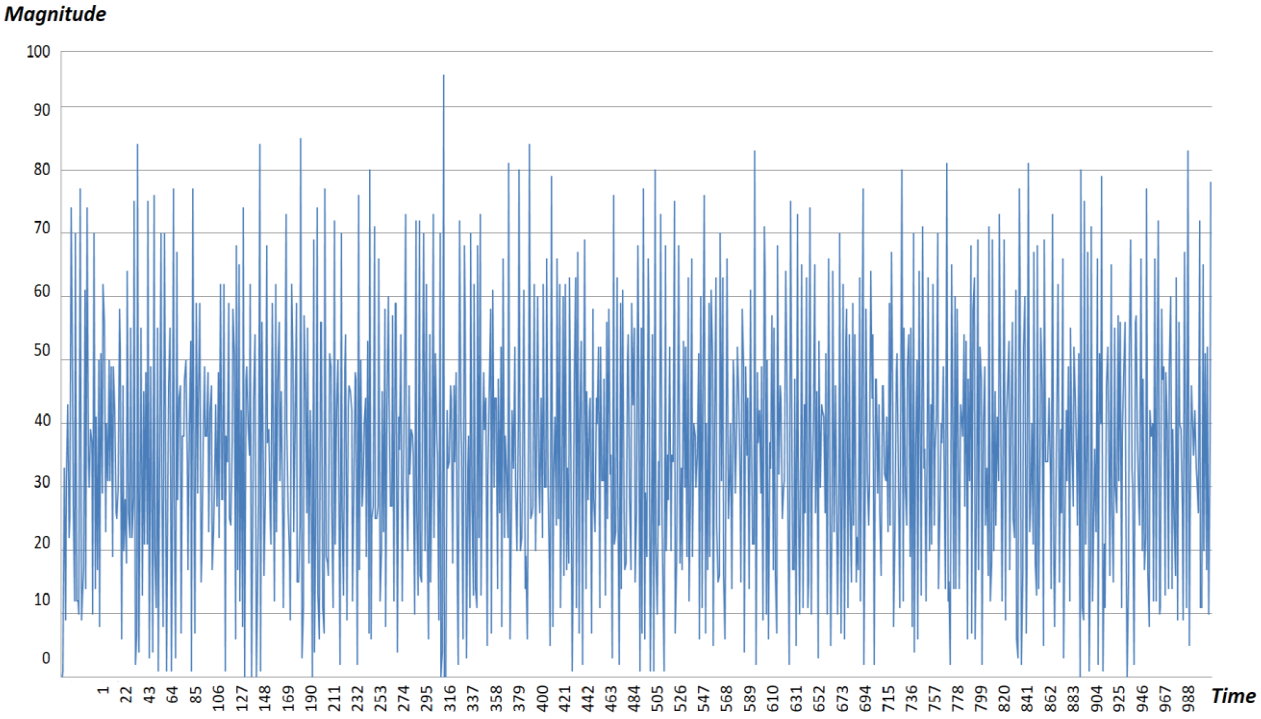
$$\frac{du}{dt} = t_{(external)} - t_{(friction)} + \int \frac{u-u_i}{|x-x_i|^a} du_i$$

όπου u η ολίσθηση, $t_{(external)}$ η δύναμη της περιστροφικής κίνησης της γης, $t_{(friction)}$ η δύναμη της τριβής, t η χρονική στιγμή και x η απόσταση. Αυτό υλοποιείται με τον παρακάτω κώδικα:

```
#include <stdio.h>
#include <stdlib.h>
#define Fcrit 0.4
#define Fout 0.01

int main(void){
float plate[10][10];
float helper_plate[10][10] = {0};
float earthquake[10][10] ={0};
int time = 0;
int isShaking = 0;
int i;
int j;
int magnitude=0;
FILE *FilePointer = fopen("Result.csv", "w");  // Make a file, to write results on it

for (i=0; i<=9; i++) {
        for (j=0; j<=9; j++){
                plate[i][j]=((float)rand()/(float)RAND_MAX)*Fcrit;
        }
}
while (time<=1000){
        isShaking=0;                    // No shaking
        for(i=0;i<=9;i++){              // Check if any plate is shaking
                for(j=0;j<=9;j++){
                        if(plate[i][j]>=Fcrit){// If the plate reaches Fcrit
                                isShaking=1; // there is at least a shaking plate
                                break;
                        }
                }
                if (isShaking){
                        break;
                }
        }
        for(i=0;i<=9;i++){
                for(j=0;j<=9;j++){
```

```
                        if(!isShaking){
                                plate[i][j]+=Fout;
                        }
                        else if(plate[i][j]>=Fcrit){ // Adjust the neighbor plates
                                if(i==0){               // 1st line
                                        helper_plate[i+1][j] += 0.25 * plate[i][j];
                                }
                                else if(i==9){          // 10th line
                                        helper_plate[i-1][j] += 0.25 * plate[i][j];
                                }
                                else{                   // Any other line
                                        helper_plate[i+1][j] += 0.25 * plate[i][j];
                                        helper_plate[i-1][j] += 0.25 * plate[i][j];
                                }
                                if(j==0){
                                        helper_plate[i][j+1] += 0.25 * plate[i][j];
                                }
                                else if(j==9){
                                        helper_plate[i][j-1] += 0.25 * plate[i][j];
                                }
                                else{
                                        helper_plate[i][j+1] += 0.25 * plate[i][j];
                                        helper_plate[i][j-1] += 0.25 * plate[i][j];
                                }
                                plate[i][j]=0;            // The plate lost its power
                                earthquake[i][j]=1;    // An earthquake began
                        }
                }
        }
        if(isShaking){
                for(i=0;i<=9;i++){
                        for(j=0;j<=9;j++){
                                plate[i][j]+=helper_plate[i][j];
                                helper_plate[i][j]=0;
                        }
                }
        }
        if(!isShaking){
                magnitude=0;
                for(i=0;i<=9;i++){
                        for(j=0;j<=9;j++){
                                if(earthquake[i][j]==1){
                                        magnitude+=1;
                                }
                                earthquake[i][j]=0;   // An earthquake ended
                        }
                }
                fprintf(FilePointer, "%d;\n", magnitude);
                time++;         }
}
fclose(FilePointer);
return 0;}
```

Η εκτέλεση του παραπάνω κώδικα παράγει ένα αρχείο .csv, με γραφική παράσταση ίδια με ένα σεισμόγραμμα:



ΒΙΒΛΙΟΓΡΑΦΙΑ-ΔΙΚΤΥΟΓΡΑΦΙΑ

KURT R. ROHLOFF - TAMER BAŞAR, April 2008, Deterministic and Stochastic Models for the Detection of Random Constant Scanning Worms

ΠΡΟΣΟΜΟΙΩΣΗ ΔΙΑΚΡΙΤΩΝ ΣΥΣΤΗΜΑΤΩΝ, ΔΙΑΦΑΝΕΙΑ - ΙΟΝΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΑΡΚΟΣ ΑΥΛΩΝΙΤΗΣ - ΜΑΡΙΑ ΝΕΦΕΛΗ ΝΙΚΗΦΟΡΟΥ - ΧΡΥΣΑ ΤΣΙΒΙΝΤΖΕΛΗ, 2019, Σημειώσεις - Μάθημα: Προσομοίωση και Μοντελοποίηση, ΙΟΝΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ