Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture with Computer Exercises:
Modelling and Simulating Social Systems with MATLAB

Project Report

Social Network Formation

Cleonela Serban & Soha Sultan

Zürich
May 2010

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich diese Gruppenarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen-Hilsmittel verwenden habe, und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichen Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Gruppenarbeit nicht, auch nicht auszugsweise, bereits für andere Prüfung ausgefertigt wurde.

Cleonela Serban Soha Sultan

Agreement for free-download

We hereby agree to make our source code of this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Cleonela Serban Soha Sultan

Contents

1. Individual contributions

We have studied the article *“A dynamic model of social network formation”* by Brian Skyrms and Robin Pemantle and implemented the models that they described: Friends I, Friends II (with and without discounting the past) and Stag Hunt.

In addition to this paper:

- We implemented the game of Iterated Prisoner’s Dilemma and we defined the strategies of this model.
- We did several simulations of all models mentioned above.
- We tried to measure the resulting networks by applying the classical networks measurements: Random network (with Poisson distribution), Scale-free network (power-law distribution) and Small-world network.

2. Introduction and Motivations

A social network can be defined as a network of individuals or organizations (nodes of the network) which are connected via one or more type of relations (links). Therefore, any two individuals having any social relation of any type will be connected by a link in the network.

Relations in the network can be friendship, physical neighbourhood, financial cooperation, group sport activities etc.

Because social networks describe all types of cooperation between people, we are all parts of social networks. Studying how people interact in these networks and how they take decisions is a very interesting field of study.

To understand these networks, researchers developed different models to describe distinct types of behaviour which vary according to the type of relations in the networks. Also the variation in peoples' characteristics via agent based modelling brings the simulation closer to reality.

In this paper we study the social model developed by Brian Skyrms and Robin Pemantle which they described in the paper "A dynamic model of social network formation". They modelled the interaction between people (agents) as games and how this interaction reinforces their future collaboration by the means of payoffs.

Section 3 is dedicated to the models that we have implemented (most of them are described in the above paper, but we also tried to model the Iterated Prisoner's Dilemma game in the same framework of modelling the interaction of social networks by games). In section 4, we describe in the first part the implementation of our models and in the second part we have some theory about network measurement together with their implementations details. We then analyse the results and compare them to known social network models in section 5 and we conclude our findings in the last section – section 6.

3. Description of the Models

3.1 Overview

In the next sections we are going to describe the models that we have implemented from the paper by Brian Skymrs and Robin Pemantle besides the iterated prisoners dilemma. We model how social networks are formed and modified, in order to study the behaviour of the network over time and its structure.

3.2 Dynamic model

We consider two types of dynamics:

- **strategic dynamics**: how individuals change their individual strategies and behaviour over time.
- **structure dynamics**: how the interactions between individuals change the structure of the network

We implement a model of structure dynamics under several conditions:

- making friends
- making friends while discounting the past
- taking strategic dynamics into consideration by interacting in a game (stag hunt and iterated prisoners' dilemma)

3.3 Introducing the models

- [**symmetric baseline model**] All models start from a symmetric initial state. Individuals in the population start by choosing another individual to interact with randomly since each individual has equal probability of being chosen. Then their choice is modified according to how their interaction was reinforced and then the process is repeated. Although the models we will describe are not complicated they introduce a social network structure after a few iterations.
- We classify the models described in the paper into two types. The first type uses uniform reinforcement and the second uses reinforcement using non-trivial strategic games. More details are described in the following sections.
- We do not consider punishment. The interaction payoff will always be positive. For example, we consider making friends as opposed to making enemies.

3.4 Uniform Reinforcement

In this section, we introduce models using uniform reinforcement. Uniform reinforcement means that the reinforcement that an individual receives from interacting with any other individual is constant, no matter with whom he interacts with. One may think that this will not lead to interesting dynamics. However this is not true as you can see from our results and analysis and also as shown analytically in the paper.

In the next three sections, we introduce three models which use uniform reinforcement. The first one uses *Asymmetric reinforcement*, the second one uses *Symmetric* *reinforcement*, while the third one uses *Symmetric reinforcement with discounting the past*.

3.4.1 Making Friends

a. Asymmetric Weights

At each round (iteration), each agent chooses another agent to interact with. The choice of whom to interact with is determined by the relative weights each agent has assigned to other agents. Because we assumed a symmetric base model, the initial weights each agent has for others are equal. Because this model uses uniform positive reinforcement, each interaction will be reinforced with the same amount. By asymmetric, we mean that if an individual A chooses to interact with an individual B, the weight associated with the interaction A to B will be reinforced while the weight of the interaction B to A will not be reinforced.

b. Symmetric Weights

As opposed to the model of making friends with asymmetric weights, this model is symmetric. If an individual A chooses to interact with an individual B, the interaction A to B will be reinforced. Also, the interaction B to A will be reinforced.

c. Symmetric Weights with discounting the past

In this model, we use symmetric weights, as explained in b. However, this model introduces discounting of the past. Discounting the past means that the payoffs that were obtained in the distant past should not have the same effect as the payoffs that were obtained recently, for example in the last iteration. This makes the model closer to reality, because in real life people's memories fade out. Recent experience may have more effects on ones' decisions than experiences he had in the remote past.

3.5 Reinforcement by games of non-trivial strategy

3.5.1 Stag – Rabbit Hunt

For this model, the agents are partitioned into two distinct types: stag hunters and rabbit hunters (but as we will see later, they can change strategy/type). The game is as follows: two individuals will go hunting together (without knowing in advance each others type). In order to hunt a stag, a cooperation of two individuals is mandatory, while for rabbits one can achieve this alone. However, hunting a stag brings greater payoff than hunting a rabit. An example of the payoffs is given in the next table:

	Stag Hunter	**Rabbit Hunter**
Stag Hunter	(1,1)	(0, 0.75)
Rabbit Hunter	(0.75, 0)	(0.75, 0.75)

There are two equilibriums in this game: Stag-Stag hunt which has the higher payoff (it is called **payoff dominant**) and Rabbit-Rabbit hunt with has least risk (it is called **risk dominant**). The scheme of payoffs of this game is asymmetric weights. Although both A – B and B – A get reinforced, the weights differ in some situations as you can observe from the above table.

In the previous models we saw how the structure of the network changes with respect to the same strategy, now we have two types of players and each player can change his strategy with a specified probability. When a player decides to change his type, he will take the one that was most successful in the previous round. Like in real life, people tend to make changes in their behaviour to adapt better to the environment – in this example we talk about following successful models in order to increase our gains.
In our experiments, the evolution of strategy is slower than the evolution of structure.

3.5.2 Iterated Prisoner’s Dilemma

Prisoner’s Dilemma problem abstractly refers to a game played by two persons where each has 2 cards **C** (cooperate) and **D** (defect). Each player selects one of the cards and puts it on the table faced down, in order not to influence the strategy choice of the other player. If both cooperate or defect they receive the same payoff, only that the cooperation pays much better. On the other hand, if they have different cards the one which cooperated wins nothing. For more about the topic, the reader is referred to *[3]*.
The general idea, for the payoff matrix (in terms of win-lose) is:

	Cooperate	**Defect**
Cooperate	win-win	lose much - win much
Defect	win much – lose much	lose - lose

For our implementation we will use the following instantiation of the payoff matrix:

	Cooperate	**Defect**
Cooperate	5-5	0-3
Defect	3-0	1-1

For the classic form of Prisoner’s Dilemma, cooperating is strictly dominated by defecting, such that the only possible equilibrium of the game is for all players to defect.

The model that we have here is for the Iterated Prisoner’s Dilemma, where the game described above is repeated a finite number of times such that players can gain knowledge about the opponents and play strategies. We defined the following two strategies – the *safe strategy* (where the player chooses to cooperate only if the probability that the opponent defects is very small) and the *risky strategy* (where the player chooses to cooperate when the probability that the opponent defects is moderated – like less than 50%). Players are divided from the start according to the two strategies, but with a given probability can change their strategy again to the previous most successful one from the previous round.

4. Implementation

We implemented and simulated the introduced models using MATLAB. In the following sections, implementation details and code will be provided.

4.1 Network Representation

A social network can be modelled as a graph where individuals are the “nodes” and relations between individuals are the “edges” of the graph.
Among different possible representations of graphs, we represent the network using adjacency matrices. The choice of representation of data as adjacency matrices is strictly related to the fact that MATLAB is specialized in working with matrices and provides many optimized operations on matrices.
An adjacency matrix is a matrix M where the value at row *i* and column *j* - *M(i, j)* describes the relation between individual *i* and individual *j*. The value at row *j* and column *i* - *M(j, i)* does not necessarily equal *M(j, i)*. Details concerning the structure of the graph differ from a model to another and will be described in the following sections.

Network *N* is described by a matrix *M* with the following specifications:

- number of rows = number of columns = number of population in the network
- values at row *i* describe the relation weights between individual *i* and all individuals.
- *M(i,i)* always equals zero *(zero matrix diagonal)*
- *M(i,j)* is the weight of the relation between individual *i* and individual *j*
- In all our implemented models, *M(i,j) > 0*
- *M* is initialized according to section 4.2 and dynamically modified in an iterated simulation according to the different models as described in section 4.3.

4.2 Initialization

As explained earlier, we use a uniform baseline model. This means that initially all weights of relations between all individuals is set to a constant, but *M(i,i)* is always set to 0 regardless of the constant value. All functions representing the different models receive initial weights as a function parameter.

MATLAB code of initialization is implemented in one line:

```
W = initialWeight * (ones(population, population) - eye(population));
```

W is initialized as a matrix of all ones using the “*ones*” MATLAB function, by subtracting a diagonal matrix of ones. The result is a matrix where all values are set to 1 except values at the diagonal. The resulting matrix is then multiplied by the constant “*initialWeight*”.
The result is: W(i,j) = initialWeight, for all i and j where i!=j.

4.3 Simulation Process

The initialization step is common for all models. However, simulation is different among the different models. Simulation is implemented as a number of subsequent iterations. In each round, individuals make decisions by choosing the person to collaborate with or by changing their strategy in a strategy based model.

4.3.1 Friends I

Friends I is the implementation name of our first model which is making friends using asymmetric weights.

- The function gets the following parameters:
    - *population* – the size of the population
    - *numIterations* – the number of iterations the simulation takes (each iteration is regarded as a time unit)
    - *payoff* –a float value representing the payoff which is added to the weight of a relation after each cooperation
    - *initialWeight* – the intitial weights used for initialization
- *W* (the weight matrix) is initialized with the value – *initialWeight*
- the function returns a probability matrix *P*, where P(i,j) represents the probability that player i and player j are friends.

```
Function [P] = FriendsI (population, numIterations, payoff,
initialWeight)
```

The simulation process is in two steps:
Choice step: The simulation is implemented as a number of iterations. In each iteration, each individual *i* chooses another individual *j* to collaborate with. The choice is made according to the weights in matrix W (initialized in section 4.2) and we use what is called **roulette wheel selection** in the following way:

```
%construct the probability vector of i visiting any other agent
totalWeight = sum(W(i,:));
P(i, :) = cumsum(W(i, :)/totalWeight);
visitProb = rand(); %it helps find the agent which will be visited
v = histc([visitProb], P(i, :));
if (sum(v) == 0)
    v(1) = 1;
else
    v = v([end 1:end-1]);
end
j = find(v == 1);
```

- we compute the probabilities of each individual to be selected by player *i* as the weight *W(i,j)* divided by the total sum of *W(i,:)*
- then we make the cumulative sum of the probabilities, such that we divide the interval [0, 1] into *population* subintervals with length proportional to the probability player *j* can be chosen by player *i*

- than we take a number at random from interval [0, 1] and we identify the subinterval in which it is contained with a player *j*

Update step: when individual i collaborate with individual j, a constant payoff is added to *W(i,j)*: *W(i,j)* = *W(i,j)* + *payoff*. The payoff is added each time an individual chooses to collaborate with another individual. The collaboration is not conditional and the other individual does not have the option to refuse collaboration.
Also, *W(i,j)* is updated when individual *i* chooses to collaborate with individual *j*, however *W(j,i)* is not updated.
Here is the MATLAB code for the update step (where v is a vector with 1 on position *j*, and 0 otherwise):

```
W(i, :) = W(i, :) + payoff * v;
```

The MATLAB code of the whole process of the simulation of Friends I:

```
function [P] = FriendsI(population, numIterations, payoff,
initialWeight)
%initialisation
W = initialWeight * (ones(population, population) - eye(population));
for t=1:numIterations
    %each agent goes to visit another agent
    for i=1:population
         %choice step
         %construct the probability vector of i visiting any other
agent
         totalWeight = sum(W(i,:));
         P(i, :) = cumsum(W(i, :)/totalWeight);
         visitProb = rand(); %it helps find the agent which will be
visited
         v = histc([visitProb], P(i, :));
         v
         if (sum(v) == 0)
             v(1) = 1;
         else
             v = v([end 1:end-1]);
         end

         %update step
   W(i, :) = W(i, :) + payoff * v;
         P(i, :) = W(i, :)/totalWeight;
    end
end
```

4.3.2 Friends II

Friends II is the implementation name of the model of making friends using symmetric weights. It also receives as a parameter the probability of discounting the past. If this probability equals zero, then the system does not consider discounting of the past. If this value is positive, the system corresponds to our third introduced model which is making friends using symmetric weights with discounting of the past.

The implementation of friends II does not differ much from the implementation described in the previous section (friends I). There are basically two differences:

1. When individual *i* chooses another individual *j* (according to the same criteria as Friends I), not only the weight W_{ij} is increased by the payoff amount, but also W_{ji} is increased by the same amount. At any time, W_{ij} equals W_{ji}. This is why it is called making friends with symmetric weights.
2. Friends II receives an extra parameter of discounting the past

Here is the MATLAB implementation of friends II,

```
function [P] = FriendsII(population, numIterations, discountPastProb, 
payoff, initialWeight)
%initialisation
W = initialWeight * (ones(population, population) - eye(population));
for t=1:numIterations
    %each agent goes to visit a agent
    for i=1:population
      %choice step
      %construct the probability vector of i visiting any other agent
         totalWeight = sum(W(i,:));
         P(i, :) = cumsum(W(i, :)/totalWeight);
         visitProb = rand(); %helps find the agent which will be 
visited
         v = histc([visitProb], P(i, :));
         if (sum(v) == 0)
             v(1) = 1;
         else
             v = v([end 1:end-1]);
         end

      %update step
      %increasing the weigths Wij and Wji by adding the payoff
      %if discountPastProb = 0, the past will be treated as the present
         W(i, :) = W(i, :) * (1.0 - discountPastProb) + payoff * v;
         W(:, i) = W(:, i) * (1.0 - discountPastProb) + payoff * (v');
         %update the probability vector of individual i
         P(i, :) = W(i, :)/totalWeight;
    end
end
```

The simulation is also done in 2 steps:
Choice Step: identical to friends I

Update Step: *discountPastProb* is the probability of discounting the past. If the *discountPastProb* is zero, this means the weights from the past will remain equally effective as the new payoffs. However, if it’s larger than 0, the weights from the past will affect the new choices less than the current weights. In the other extreme case, if the discountPastProb equals 1, only the payoffs from the last iteration will affect the choice of the partner in the current iteration.

Here is the implementation of the update step:

```
W(i, :) = W(i, :) * (1.0 - discountPastProb) + payoff * v;
W(:, i) = W(:, i) * (1.0 - discountPastProb) + payoff * (v');
%update the probability vector of individual i
P(i, :) = W(i, :)/totalWeight;
```

4.3.3 Stag-Rabbit Hunt

The simulation goes as follows:

- We encode the players strategies/types with 1= *stag hunter*, 2 = *rabbit hunter*
- The function gets parameters:
    - *population* – the size of the population
    - *numIterations* – the number of iterations the simulation takes (each iteration is regarded as a time unit)
    - *payoff* – the matrix that we described in the previous chapter, thought it is just a 2x2 matrix (instead of a 2x2x2) because the payoffs are symmetric with respect to the player types. For example, if a stag and a rabbit hunter play the game, then the payoff of the stag hunter is *payoff(1,2)*
    - *q* – probability that a player changes strategies
- *W* (the weight matrix) is initialized with a value – *initialWeight* received as a parameter

```
function [P] = StagHunt(population, numIterations, payoff,
initialWeight, q)
W = initialWeight * (ones(population, population) - eye(population));
%randomly setting strategy for each person 1 - stag, 2 – rabbit hunters
type = randi(2, 1, population);
```

- *type*(the vector that stores each player's assigned strategy – *1:stag hunter*, *2:rabbit hunter*) – initially this are given at random
- the function returns a probability matrix *P*, where P(i,j) represents the probability that player i and player j are friends.

In each round (one iteration), each player *i* makes the **choice step** as described in FriendsI to find the one to collaborate with (as discussed earlier, the players that contributed most on the weights of *i* have the higher chance of being chosen).

Next, we get the payoffs for each player, after playing Stag-Hunt game.

```
pi = payoff(type(i), type(j)); %payoff of player i
pj = payoff(type(j), type(i)); %payoff of player j
```

Then we update the weight matrix according with the payoffs of each player achieved:

```
W(i, :) = W(i, :) + pi * v;
W(:, i) = W(:, i) + pj * v';
P(i, :) = W(i, :)/totalWeight;
```

As discussed on the description of the model, with probability *q* a player can change strategy, becoming the type that was most popular in the previous round. For this we keep updated the vector *TotalPayoff* with the sum of all payoffs that were gained in that round per each type (the vector has two components).

```
%updating total payoff
TotalPayoff(type(i)) = TotalPayoff(type(i)) + pi;
TotalPayoff(type(j)) = TotalPayoff(type(j)) + pj;
```

At each round we get a random number *q0* in interval [0, 1] for each player and if it is less than *q* then the respective player changes his strategy with the one that has maximum *TotalPayoff*.

```
q0 = rand();
%find which is the most successful type
T = 1;
if (TotalPayoff(1) < TotalPayoff(2))
    T = 2;
end
if (q0 < q) %i changes type with probability q
    type(i) = T;
end
```

4.3.4 Prisoner's Dilemma

For Prisoner's Dilemma problem we again have two types of strategy: the *safe* where a player chooses to Cooperate only if he is almost sure that the opponent will do the same thing and *risky* one where the player will take the change to Cooperate even if he is not really sure that he's partner will cooperate. We modeled the certainty and uncertainty by a threshold probability value *thresholdType* that is to be compared with the probability computed from the knowledge matrix *K*.

The simulation goes as follows:

- initialize data as in StagHunt, the only difference is that we also initialize *K* with ones, which has the meaning that player *i* knows about player *j* from the previous games they played together that he played *K(i,j,1)* times Cooperate and *K(i,j,2)* times defect. Based on this fact, he will compute the probability that player *j* will defect, based on their history game.
- In each round player *i*:
    - chooses an opponent as described in **choice step**

- based on the strategy they apply, both players chose their cards as in the code (based on the above discussions):

```
%the game of player i with player j
probDi = K(i,j,2) / sum(K(i,j,:));
probDj = K(j,i,2) / sum(K(j,i,:));
%ci, cj represents the card/choice of i, j
if (probDi < thresholdType(type(i)))
    ci = 1; %i cooperates
else
    ci = 2; %i defects
end

if (probDj < thresholdType(type(j)))
    cj = 1; %j cooperates
else
    cj = 2; %j defects
end
```

- then after the game, each player updates his knowledge matrix (learns/gains experience from what happened this round)

```
%update the knowledge matrix
K(i,j,cj) = K(i,j,cj) + 1;
K(j,i,ci) = K(j,i,ci) + 1;
```

- the end is similar to StagHunt, the weight matrix *W* is updated and the *TotalPayoff* vector is also updated (because with *q* probability, at each round a player can change strategies with the most popular in the previous round)

4.2 Network measurement

Networks are described as graphs (a set of items that we call *vertices* which are connected between them by *edges*). There is an impressive number of systems that take the form of networks in the world. Social networks are an example of these networks. They model the connections between individuals (friendship, kinship, common interests etc.). They have been extensively studied in social sciences where the principal issues addressed are the **centrality** (which individuals are best connected to others or have most influence) and **connectivity** (whether and how individuals are connected to one another through the network).

Up to this point, we have described the way our agents interact and how this conducts to network formation, and how we have implemented both the evolution of strategy and structure. In this section we describe the tools we use for measuring the structure of the resulting networks from our experiments (namely the models we have implemented).

Next we describe three types of networks: Random and Scale-free networks (which measure the centrality property by means of degree distribution) and Small-world networks (that deals with the connectivity property).

4.2.1 Random networks

A random network is a network in which the connections between individuals are random and does not follow a specific structure or dynamics. Random networks have a Poisson degree distribution. To test if our models generate random networks or not, we compare the probability density function of the degree distribution of the generated network with the Poisson probability density function.
The Poisson probability density function is calculated using the MATLAB function (*Y = POISSPDF(X, LAMBDA)*), where LAMBDA is the expected value of the degree distribution. We calculated the expected value of the degree distribution as the average degree of the population.
Comparison is implemented in RandomNetwork.m.

Here is the MATLAB implementation:

```
function [xaxis, yaxis1, yaxis2 ] = RandomNetwork ( Network )
population = length(Network);
degree = degreeDistribution(Network); %degree is degree distribution of
the network
degreePDF = degree/population;
%x-axis should show the degrees present in the Network
xaxis = [0:(length(degree)-1)];
%compute the average degree for Poisson distribution
lambda = sum(xaxis.* degree')/population;
poisson =poisspdf(xaxis,lambda);

yaxis1 = degreePDF;
yaxis2 = poisson;
end
```

4.2.2 Scale-free networks

If the degree distribution of a network follows the power-law distribution, then it is called a scale-free network. In a power-law degree distribution, the fraction of nodes Y having X connections with other nodes has the following relation:

$$Y \propto X^{-\gamma}, \text{ where } \gamma \text{ is a constant} > 0.$$

To test if our models represent scale-free networks or not, we implemented *Scale_Free_Network (Network)* function in Scale_Free_Network.m

The function gets the Network (the adjacency matrix of the network graph) as a parameter. The degree distribution of the network is calculated using *degreeDistribution(Network)*. We use least squares curve fitting algorithm in MATLAB (Function name: "*lsqcurvefit*") to calculate the parameters of the fitted power-law distribution function:

$$Y = \rho X^{-\gamma}$$

In this equation, the parameters that need to be approximated are ρ and γ. A helper function FUN was also implemented to be used by the least squares algorithm.
The returned parameters ρ and γ are used to plot the power-law curve of the degree distribution. If the fitted power-law curve well approximates the actual calculated degree distribution, then the degree distribution follows a power-law distribution, in which case the network is considered to be a scale-free network.

Here is the MATLAB implementation:

```
%plots the degree distribution and the power law curve fit of the
%distribution
function [xaxis, yaxis1, yaxis2]= Scale_Free_Network( Network )
deg = degreeDistribution(Network);
xaxis = 0:(length(deg)-1);
X0 = [1, 1]; %X0 is vector of the initial parameters input to the
%FUN: is a helper function, implements Y = a X^b
X = lsqcurvefit (@FUN , X0, xaxis, (1./(deg'+1)));
X(1) = 1/X(1);
X(2) = -X(2);
powerLawCurve = log(FUN (X, xaxis));
yaxis1 = log(deg)';
yaxis2 = powerLawCurve;
end
```

The implementation of the helper function FUN:

```
%helper function for the power law fitting function
function [ Y ] = FUN( X , XData )
Y = X(1)*(XData.^X(2));
end
```

4.2.3 Small-world network

The small-world phenomenon was studied by Milgram 1967 and it concludes that two individuals, selected randomly from anywhere on the planet are "connected" through a chain of no more than six intermediate acquaintances. This phenomenon seems as to be an unlikely coincidence, but it might actually indicate the underlying structure of modern social networks. *[2]*

What do we mean when we say that the world is *small*? It means that every element in the network is somehow "close" to almost every other element, even those that are perceived to be far away (in the original experiment the number of intermediate acquaintances defines the notion of close, and people who live at great distances or far away – on different continents per example are likely to be not close).

At the intuitive level, we say that a network has small-world properties if the network is a connected undirected graph where the average distance between all pairs of vertices is very small when compared to the dimensions of the network.

The first statistic of interest is the *characteristic path length* ***(L)*** that is exactly the average distance that is mentioned above, but this requires that the entire network is known which is possible in our simulation. Another statistic is the *clustering coefficient* ***(C)*** that is a measure for the local graph structure.

In this paper, we explore the characteristic path length only.

The implementation of small-world network is straight forward:

```
function [avgLength] = SmallWorldNetwork(A)
if (isConnected(A) == 1)
    D = RoyFloyd(A);
    avgLength = sum(sum(D)) / sum(sum(D ~= 0));
else
    avgLength = -1;
end
```

We get the adjacency matrix of the network as an input (this is the matrix of probabilities that each of the models outputs after it was processed with Threshold function) and we output either -1 if the network is not connected, or the average length of all minimum paths in the network.

As described here, the first step is to check if *A* is connected. (This is done with the help of function *isConnected* which does a BFS on the networks underlying graph). If so, then we compute the minimum distance matrix of *A* using Roy-Floyd algorithm – that has $O(n^3)$ runtime complexity. Finally, we output the average length of the paths.

5. Simulation Results and Discussion

5.1 Results

5.1.1 Making Friends (Asymmetric Weights)

Parameter Name	**Parameter Value**
population	1000
numIterations	1000
initWeight	1
Threshold	0.001

We did the simulation for our first introduced model. We used the parameters in the previous table. It is clear from figure 1 – Random Network, that this model forms a random network because the degree distribution has the form of the Poisson distribution, while you can observe that the power-law is quite different when compared to the degree distribution. In figure 1 – Random Network, we draw the probability density function of the degree distribution versus the Poisson degree distribution with a value of lambda equal to the average degree distribution.

The formed network is not a small-world network because as shown in figure I there is a big proportion of the population that have a degree of 0 and therefore are disconnected.

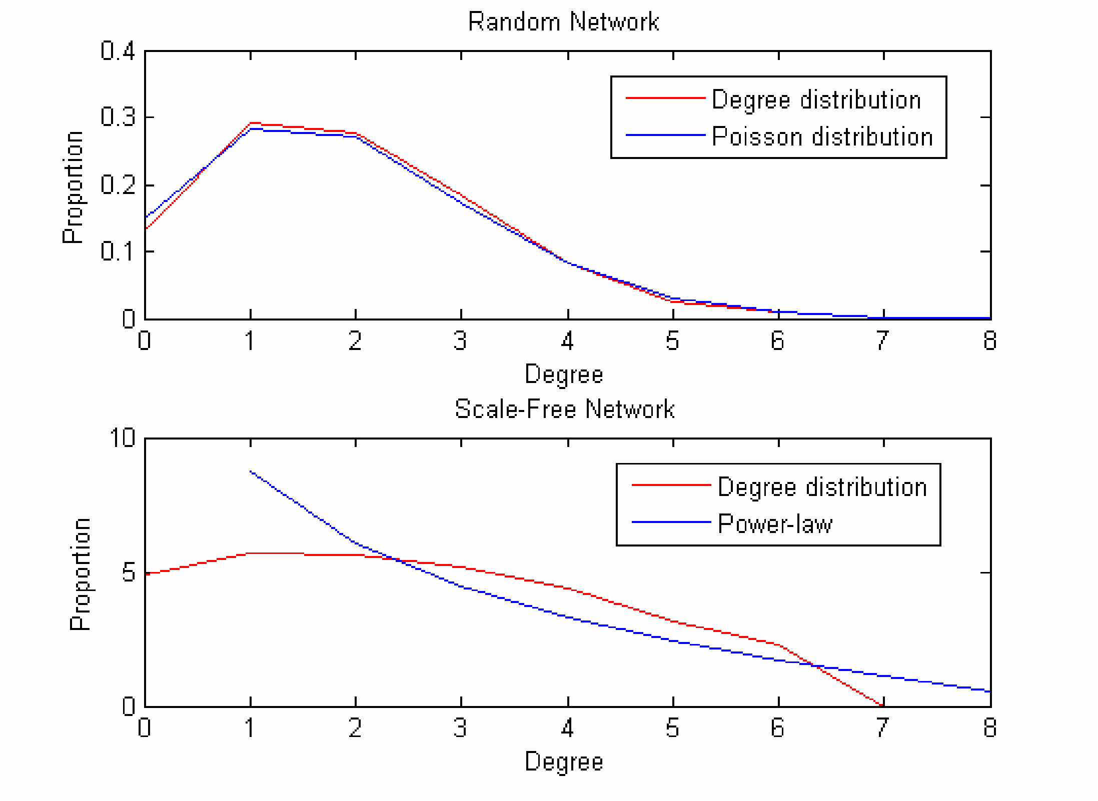

Figure 1: Friends I (simulation results)

5.1.2 Making Friends (Symmetric Weights)

Parameter Name	**Parameter Value**
population	1000
numIterations	1000
discountPastProb	0
initWeight	1
threshold	0.005

For the simulation result in figure 2, we used the parameters in the above table. As it is clear from the figure, the degree distribution of the formed network using making friends with symmetric weights is a Poisson degree distribution, therefore the formed network using this model is a random network.

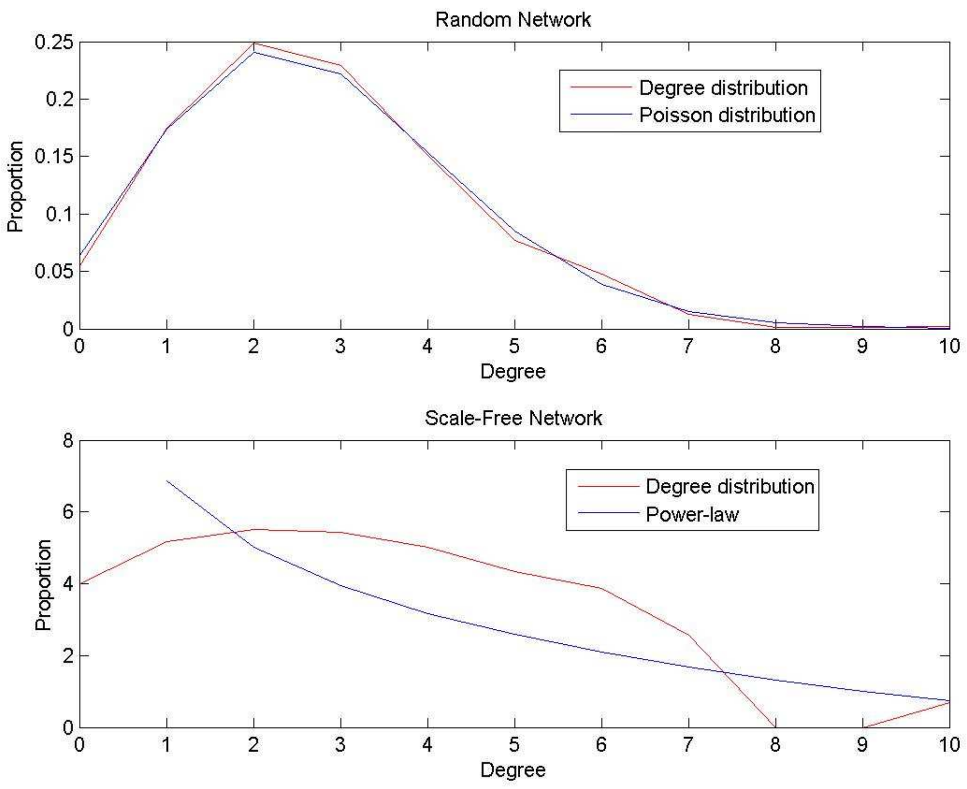

Figure 2: Friends II (without discounting the past)

5.1.3 Making friends with symmetric weights and discounting the past

5.1.3.a: discounting the past with 50% probability

Discounting the past adds more structure to the network, because it makes the model closer to real life. In reality, people forget past activities or at least remember newer activities and base their future decision based on recent activities rather than older activities.

For this simulation, we used the following parameters:

Parameter Name	**Parameter Value**
population	1000
numIterations	1000
discountPastProb	0.5
initWeight	1
Threshold	0.005

The result is not a random network because it doesn't have a Poisson degree distribution, but as you can see from Figure 3 it is similar to the Scale-Free Network.

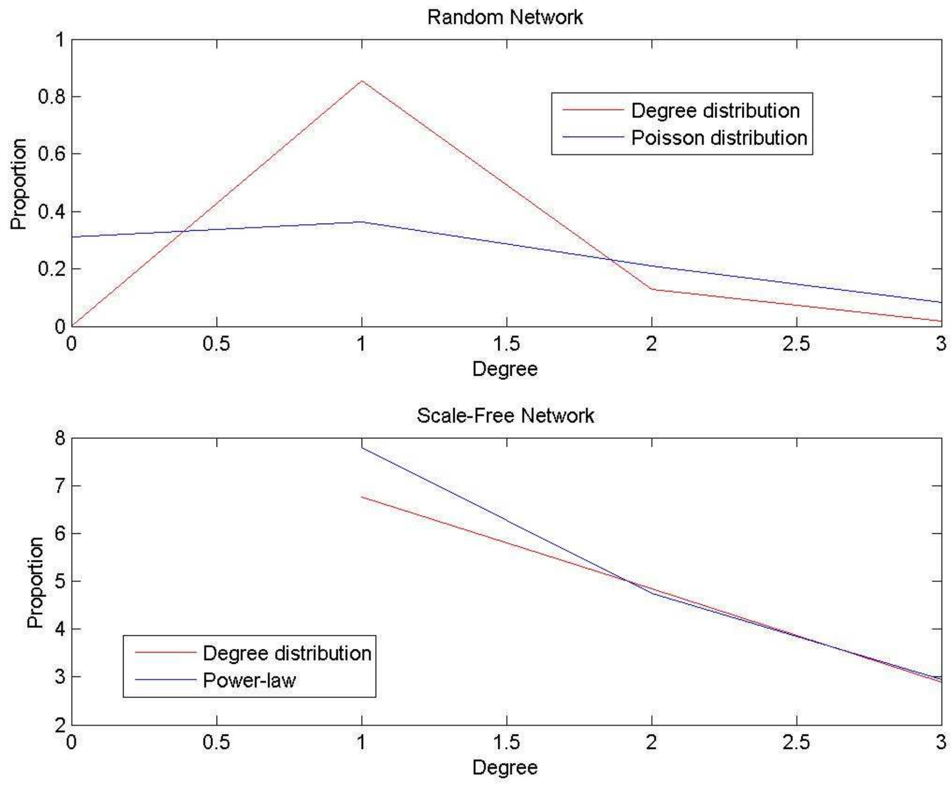

Figure 3: Friends II (with discounting the past)

5.1.3.b: totally discounting the past

In this section, we show the result of the model of making friends with symmetric weights, but with discounting the past. Therefore, the choice of the individual to collaborate will only depend on the last iteration. The individual will forget that he cooperated with anybody but the one he cooperated with last time. As a result, at the end of the iterations, each individual will make friends only with one individual which is the one he cooperated with in the last iteration. The degree distribution is not interesting in this case because all individuals have a degree of 1.

5.1.3 Stag Hunt

For all experiments the payoff matrix was the one used in the original article:

payoff	*Stag Hunt*	*Rabbit Hunt*
Stag Hunt	(1, 1)	(0, 0.75)
Rabbit Hunt	(0.75, 0)	(0.75, 0.75)

As you will see from the simulations we did, Stag Hunt resulting network is a Small-world, and its degree distribution follow the Poisson distribution.

5.1.3.1 Simulation 1

Parameter Name	**Parameter Value**
population	100
numIterations	1000
initWeight	1
Q	0.01
threshold	0.03

The simulation was conducted 10 times with the above parameters and the results show that the resulting network has small-world characteristics since the average length is 3 (which is like a constant when compared to a population of 100 persons) and also has random characteristics as you can see from the plot in figure 2.

Small-World results: the average length between any two persons is **3**.

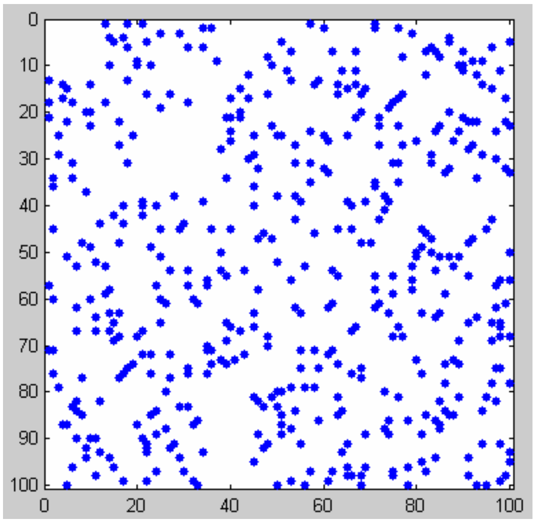

Figure 4: Sparsity plot for Stag Hunt Network (*Simulation 1*)

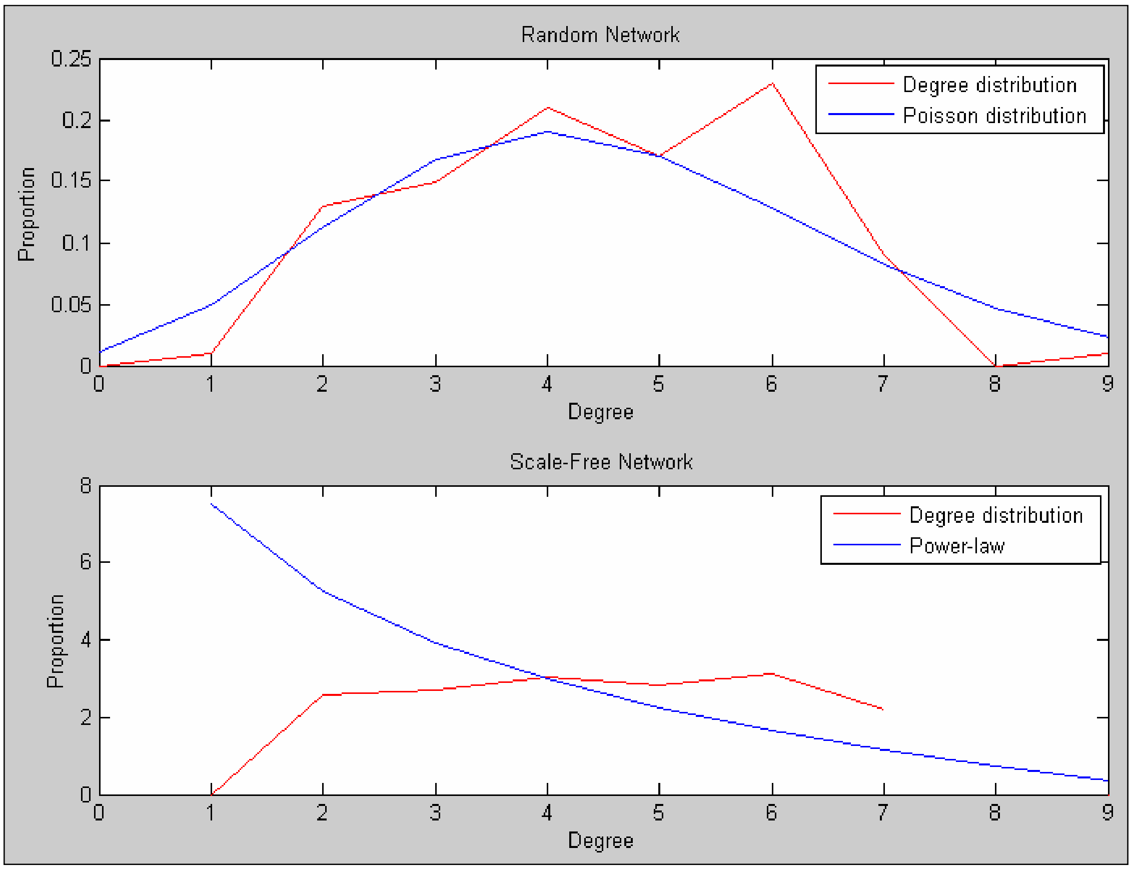

Figure 5: Stag Hunt Network measurements (*Simulation 1*)

5.1.3.2 Simulation 2

We conducted simulation 2 only 5 times with the following parameters, and we next present you our findings.

Parameter Name	**Parameter Value**
Population	1000
numIterations	1000
initWeight	1
Q	0.01
Threshold	0.003

As you can see from Figure 6, the network is not fully-connected, it has approx. 9000 edges from almost 1000 x 1000 possible. But, the **Small-World results** show that the average length between any two persons is **3**, so the resulting network is a small-world network.

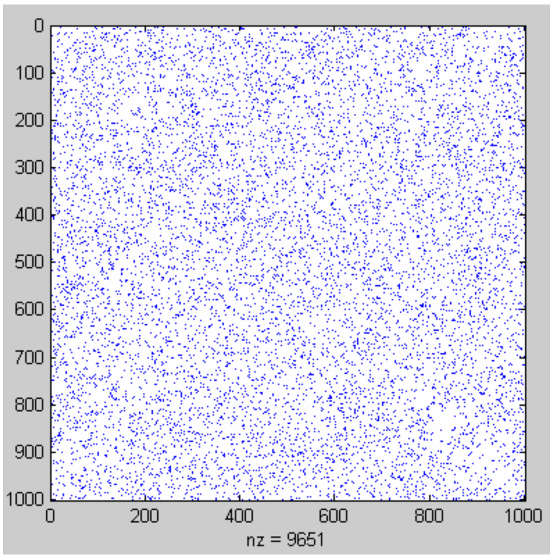

Figure 6: Sparsity representation of Stag Hunt network (simulation 2)

If you look at Figure 7, you can observe that the resulting network has random network characteristics since the degree distribution follows closely the Poisson distribution. Also, it is not the case that the network is Scale-Free.

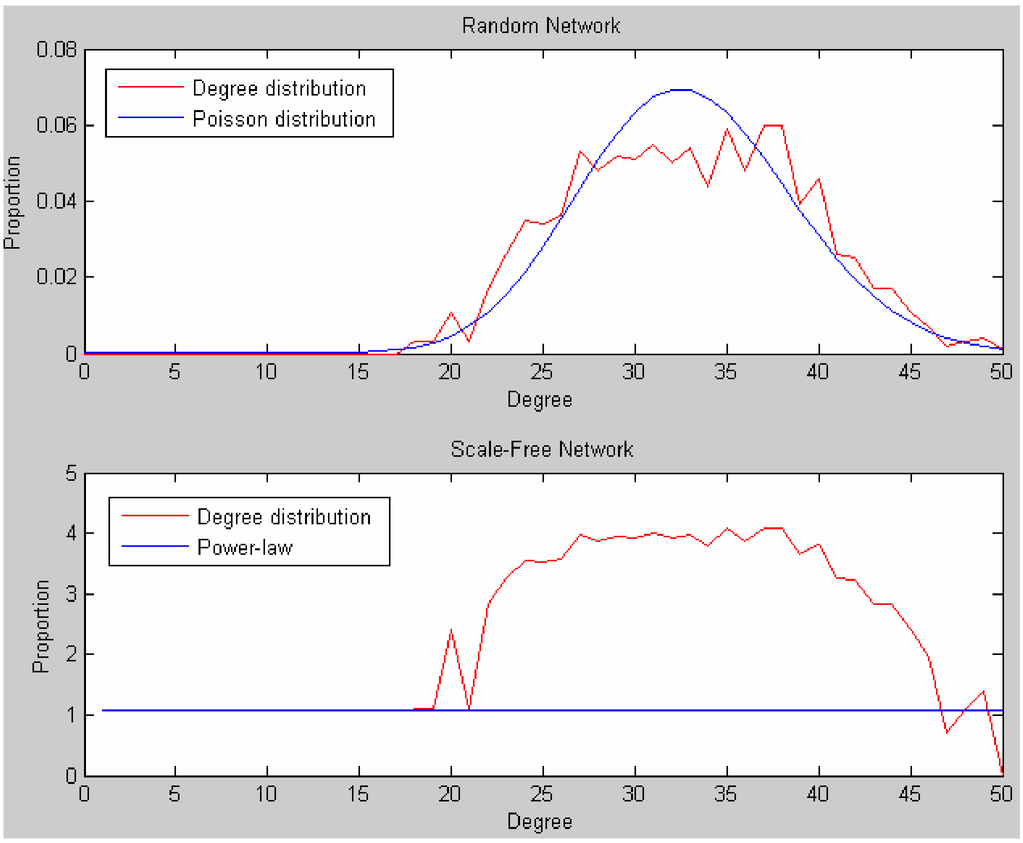

Figure 7: Stag Hunt Network (Simulation 2)

5.1.4 Iterated Prisoner's Dilemma

For all simulations, the payoff matrix was as the one described in the model:

payoff	*Cooperate*	*Defect*
Cooperate	(5, 5)	(0, 3)
Defect	(3, 0)	(1, 1)

The thresholds value for the two strategies were set at *0.35* for the risky type and *0.85* for the safe type.
Overall, the Iterated Prisoner's Dilemma Network seems to be both Small-World and Random, the same as Stag Hunt. The difference that we have observed from the probability matrix is that after the same *iterNo* it is more clear which players are friends (for example in Stag Hunt the highest probability in the matrix is 0.04, while in PD is 0.08, but this could be because of the value of the payoffs).

5.1.4.1 Simulation 1

We conducted 10 times simulation 1, with the following parameters:

Parameter Name	**Parameter Value**
Population	100
numIterations	1000
initWeight	1
q	0.01
threshold	0.03

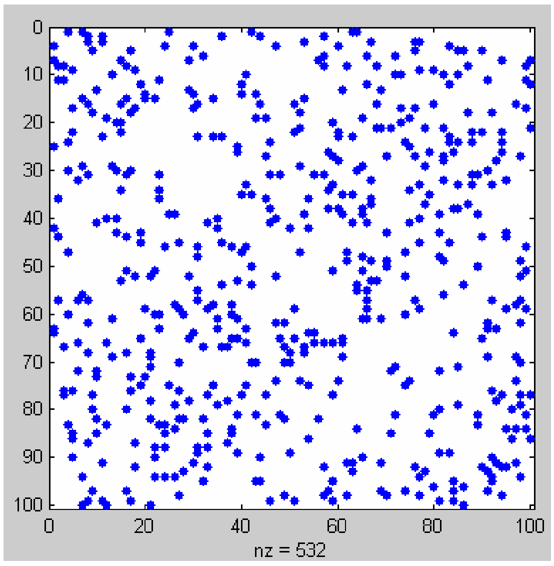

Figure 8: Sparsity of adjacency matrix - Prisoner's Dilemma *(Simulation 1)*

For the small values simulation of Prisoner's Dilemma, the network is Small-world since the results show that the average path length is only 2. In figure 8, you can see the adjacency matrix.
From figure 9, it can be seen that the degree distribution looks similar to the Poisson distribution and hence the network must be a random network.

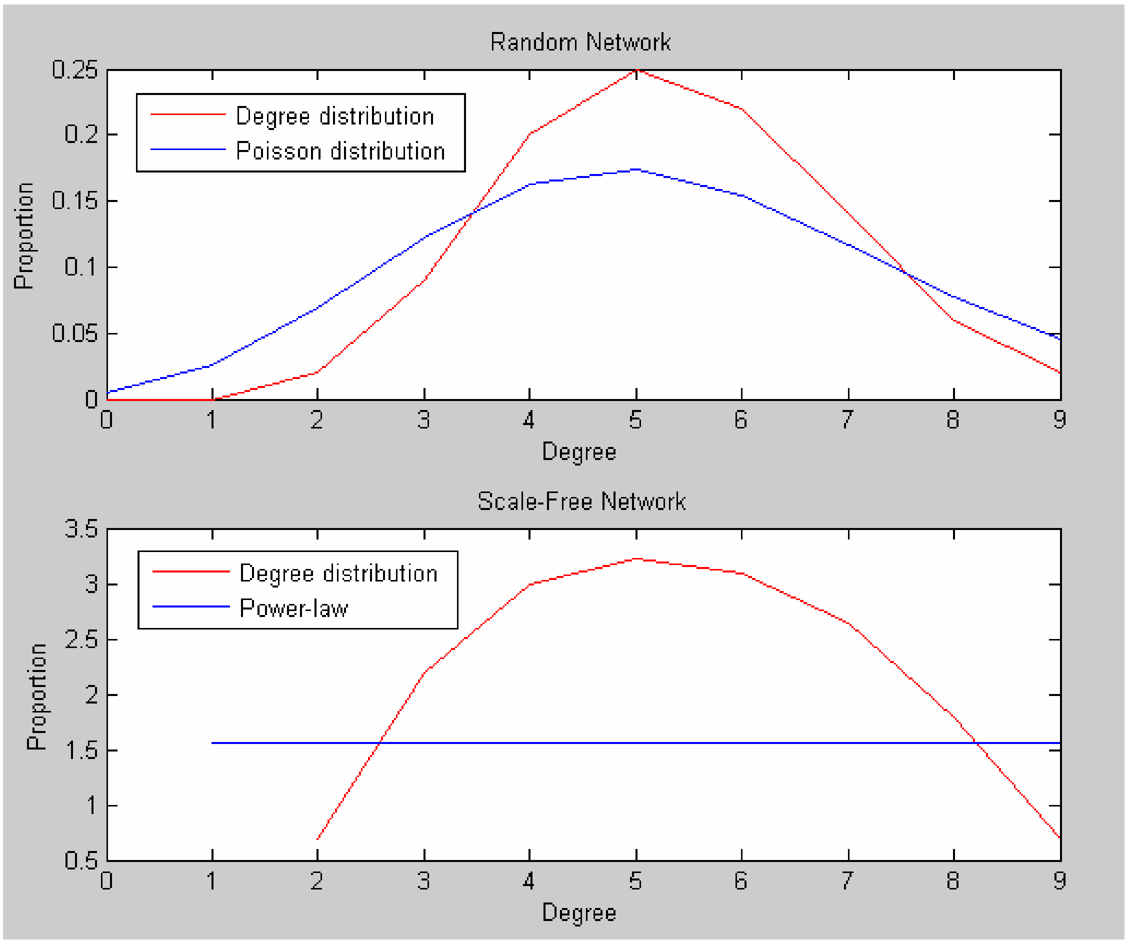

Figure 9: Prisoner's Dilemma Network analysis (*Simulation 1*)

5.1.4.2 Simulation 2

The second simulation was conducted only 5 times with the following parameters:

Parameter Name	**Parameter Value**
population	500
numIterations	1000
initWeight	1
q	0.01
threshold	0.02

By the **small-world results**, namely that the average length path is 4 we can state that the network has small-world properties, although as you can see in figure 10 is kind of sparse, but connected.

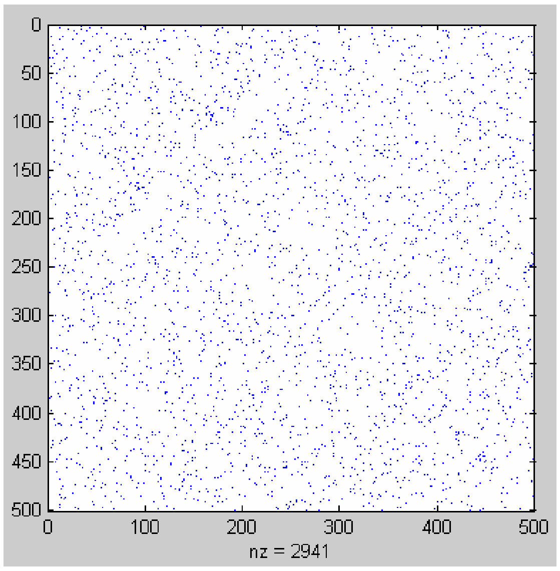

Figure 10: Sparsity matrix - Prisoner's Dilemma (*Simulation 2*)

From Figure 11, we get again that the network is random, since the degree distribution is similar with Poisson distribution.

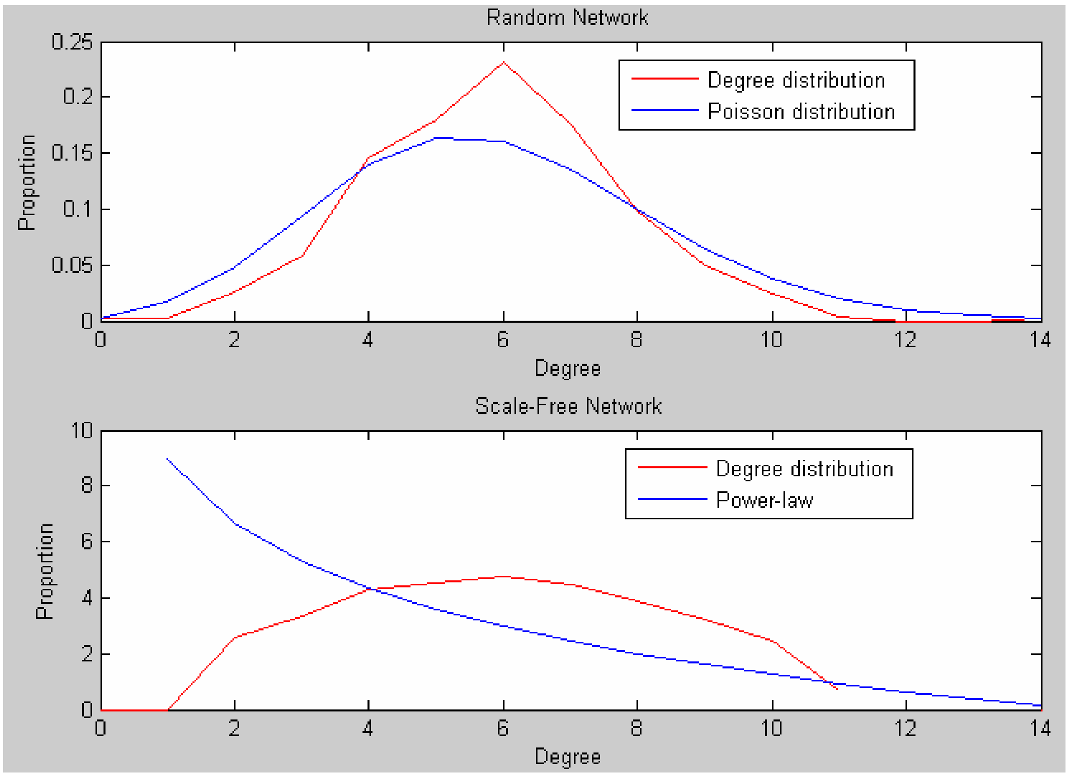

Figure 11: Prisoner's Dilemma Network analysis (*Simulation 2*)

6. Summary and Outlook

In this paper, we described and analysed some of the social network formation mathematical models which were introduced by Skyrms and Pemantle [4]. We implemented these models using MATLAB. Having the implementation ourselves, we managed to analyse them by running a number of computer simulations and observing the results. We compared the networks formed by the models to classical social networks structures which are random networks, scale free networks and small world networks. We also used the models to simulate two different games which are stag hunt and iterated prisoners' dilemma.

We showed that the introduced models consider two different types of dynamics, structure dynamics and strategy dynamics. Some of the models consider only the first type of dynamics. However, the two simulated games model the strategy dynamics as well. Adding the strategy dynamics makes the models come closer to reality because in reality people change their strategies when they find that they are losing for example.

The analysis of the first two models we described which are making friends with asymmetric weights and making friends with symmetric weights showed that the these models form a random network. However, adding the parameter of discounting the past which makes the model closer to reality because people usually make their new decisions based on recent experiences rather than old history, made a new model which forms a scale-free network.

In the analysis of the games – stag hunt and iterated prisoner's dilemma, we encountered that the resulting networks have random network properties, since their degree distribution is similar to the Poisson distribution. But, what was interesting was that we also encountered that they exhibit the small-world phenomena, since the network are connected and the average path length is constant (less than 5 for all simulations).
The single difference between the two games is that in the case of iterated prisoner's dilemma the connections that form among individuals of the population are much stronger when compared to the one in stag hunt. A proof of this statement is the fact that the probability matrix resulting from iterated prisoner's dilemma has higher probabilities when associated to the connections between people, although the matrix is not much more sparse than stag hunt's probability matrix.
In conclusion, our findings show that apparently, the different strategy that our players have with respect to the non-trivial models (iterated prisoner's dilemma, stag hunt) and the dynamics in this strategy give rise to a complex structure of the formed network, which is perceived as a random network.

We believe that we did a good and fairly complete analysis on social network formation with respect to the studied paper of Skyrms and Pemantle. Also, following their example, we tried to analyse another non-trivial model of a network where interaction took the form of another game, namely the iterated prisoner's dilemma.

7. References

1. Newman, M.E.J, *The structure and function of complex networks* [2003]
2. Watts, Duncan J., Networks, *Dynamics, and the Small-World Phenomenon*
3. http://en.wikipedia.org/wiki/Prisoner's_dilemma
4. Skyrms, Brian and Pemantle, Robin, *A dynamic model of social network formation*