ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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

Project Report

Emergence of Cooperation

Boris Daskalov & Thanuja Ambegoda

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.

Boris Daskalov Thanuja Ambegoda

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.

Boris Daskalov Thanuja Ambegoda

Table of contents

Individual contributions

1. Literature survey – Boris and Thanuja
2. Design of the simulation model - Boris & Thanuja
3. MATLAB programming & simulation
    a. Generation of graphs to represent structured populations – Boris
    b. Fitness calculation to assist the strategy update step for each iteration of the game - Thanuja
    c. Strategy update rule for the iterated game, for each iteration of the prisoner's dilemma game - Thanuja
    d. Simulation routine to observe the emergence of cooperation in different graphs provided with variable b/c - Boris
    e. Optimizing the code to enhance performance - Boris
    f. Testing & Debugging – Boris & Thanuja
    g. Generation of plots to observe the emergence of cooperation in the structured populations considered, under different conditions - Boris
4. Documentation.
    a. Model – Thanuja
    b. Code – Boris
    c. Discussion - Thanuja

Introduction and Motivations

Cooperation is an important concept in biology as well as in sociology. We can refer to cooperation in the context of a human population who interact with each other, and also, in a biological context where we consider the interaction of cell populations to groups of animals. Interactions in human populations per se, are based on some mechanisms that promote cooperation. The same is true for biological populations.

In this Matlab simulation we are presenting focuses on observing the emergence of cooperation, within a population of individuals whose interactions are depicted by a regular graph with known degree or a random scale free graph. In this setting, the individuals play a repeated prisoner's dilemma game with their neighbours.

The simulation is partially based on a previous claim by Nowak et al. (2006), which states that cooperation can evolve in a structured population if the benefit to cost ratio of the prisoner's dilemma game exceeds the average number of neighbours. i.e $b/c > k$.

This is a very interesting finding since it poses itself as an exception to the case where natural selection favours defection over cooperation in unstructured populations, which to some extent looks counter intuitive.

Description of the Model

The main components of the simulation model are as follows:

1. **Structured Populations**

   Graphs are used to describe the relationships among the individuals in a finite structured population. The existence of an edge between two nodes mean that the 2 players to whom the 2 nodes correspond, play an iterated prisoner's dilemma game with each other. In our simulations, we use:

   a. Lattices with known degree k
   b. Random scale-free graph

   a. Lattices with known degree k

   In a lattice, each node has the same degree. i.e. each node is connected to the same number of neighbours. We run simulations to observe the emergence of cooperation for lattices with different values of k. i.e. for k = 2, 3 & 6.

   b. Random Scale-free network:

   In a scale-free network, the degree distribution follows a power law. We observe the emergence of cooperation in a population whose interactions can be modelled using a random scale-free network. We generate this scale free random graph using the algorithm put forward by Barabasi & Alberts (2002), which produces a scale free graph based on preferential attachment [2].

   As described later, we run the iterated prisoner's dilemma game several times on each of the structured population to observe the emergence of cooperation, on average.

2. **Prisoner's Dilemma Payoff Matrix & the Game**

   As described earlier, the population is structured according to a graph. The edges of the graph determine who can play against whom. The expected payoff for each game is given by the Prisoner's Dilemma Payoff Matrix.

   We use the following matrix for the prisoner's dilemma game.

$$\begin{array}{c} \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cc} C & D \end{array} \\ \begin{pmatrix} b-c & -c \\ b & 0 \end{pmatrix} \end{array}$$

As depicted by the above matrix, in the prisoner's dilemma game, we have 2 strategies that the individuals could use to play. It's either cooperation 'C' or defection 'D'. The co-operators pay a cost of cooperation 'c' to all the connected neighbours (the neighbours that they play the game with). Any neighbour of any co-operator receives a benefit of cooperation 'b', from each cooperating neighbour.

More specifically, according to this payoff matrix, when C plays against C, it receives a payoff of b-c. When C plays against D, C receives a payoff of –c. When D plays against C, D receives a payoff of b. Finally, when D plays against D, it receives a payoff of zero.

Accordingly, we calculate the expected payoff for each game an individual plays. This expected payoff features in the calculation of the fitness of an individual, as shown in the next point.

In each iteration of the game (in each time step), the fitness of an individual can change depending the changes of the strategy of one of its neighbours. The update rule determines the change of the strategy of an individual (from C to D or from D to C), as described later.

3. Method to determine the fitness of an individual

The fitness of the individuals of the population is required when it comes to updating the strategy of an individual, as described in the strategy update rule. It is calculated using the following formula, under weak selection, for which w=0.01.

$$f_i = 1 - w + wF_i$$

Here, F_i is the expected payoff of the i^{th} individual calculated from the game. i.e. the value of payoff obtained from the payoff matrix. Here, w is the intensity of selection. In weak selection, where w is closer to zero, the payoff of the game contributes only to a small fraction of the total fitness of the individual. It is trivial to observe that the contribution of the game to the fitness of the individual is governed by w.

4. Update rule to propagate the strategy among the individuals
Death-Birth (DB) update rule:

During each time step of the simulation, one individual changes his strategy according to an update rule. We consider the Death-Birth (DB) update rule in our game. At each time step, a random player is chosen to die. Then, the neighbors compete for the empty site proportional to their fitness.

When the strategy of an individual changes according to this rule, the fitness of each of its neighbors also changes. This is because, the fitness depends on the payoff generated by the game and the payoffs the neighbors get depend on the strategy of this individual (as described earlier).

Implementation

Our implementation can be described by the following steps.

1. **Initialization:**
   Here we generate the graphs that we will later use as structures for the population. We generate 5 graphs:
   - 2D and 3D periodic lattices.
   - A 6D hypercube.
   - A random scale-free network generated by the principle of preferential attachment.

   Also in this part we make a list of the benefit / (cost * avgDegree) ratios and a corresponding list with the selection intensity values.

```
graphs = [
    struct('name', '2D regular', 'A', createNDLattice(8, 2));
    struct('name', '3D regular', 'A', createNDLattice(4, 3));
    struct('name', '6D hypercube', 'A', createNDLattice(2,6));
    struct('name', 'scale-free', 'A', randomScaleFreeNetwork(64));
    ];

bOverck             = [0.5  1    2    1];
selectionIntensity  = [0.01 0.01 0.01 0];
numTrajectories = 1000;
%number of iterations
nIter = 100000;
```

2. **Parameters and trajectory loops:**
   We are running the simulation for different values of the parameters for which we want to compare the results. In the outer two loops we iterate over all the graph structures and selection parameters. Our simulation is a stochastic process in which the results don't depend only on the parameters. For each set of parameter values we simulate 1000 trajectories. Inside the parameters loops there is a trajectories loop. Inside the trajectories loop we prepare the payoffs matrix and run the simulation with the given parameters. After the simulation finishes we store the result in the "res" array that we use afterwards to compute statistics.

```
res = zeros(numel(graphs), numel(bOverck), numTrajectories);

for g=1:numel(graphs)
    for i = 1:numel(bOverck);
        for k = 1:numTrajectories

            % get the graph adjacency matrix
            A = graphs(g).A;

            % calculate the average degree
            avgDegree = mean( sum(A) );
```

```
            % initialize the game payoffs matrix
            cost = 2;
            benefit = cost * avgDegree * bOverck(i);
            P = [ benefit-cost -cost;
                  benefit       0   ];

            % do the simulation
            res(g,i,k) = simulate(A,  nIter, P, selectionIntensity(i));

            % print progress
            if mod(k,10)==1
                disp(['g=', num2str(g), ' i=', num2str(i),...
                      ' k=',num2str(k)])
            end
        end
    end
end
```

3. Visualizing and storing the results.

In this part we generate bar graphs that present the fixation probabilities for the different parameter values. The probabilities are estimated as the fraction of simulated trajectories in which the co-operators reached fixation in the whole population. Results from the simulation and the

```
for g=1:numel(graphs)
    % calculate the fixation probabilities for the different selection
    % values
    fixationProb = sum(res(g,:,:) == 1, 3) / numTrajectories;

    % create a bar plot of the fixation probabilities
    figure;
    bar(fixationProb);
    set(gca, 'FontSize', 12);
    title(graphs(g).name);
    ylim([0 1]);
    xlabel('benefit / (cost * averageDegree)');
    ylabel('Cooperator fixation probability');
    set(gca, 'XTick', [1 2 3 4]);
    set(gca, 'XTickLabel', {'0.5', '1.0', '2.0', 'no selection'});

    % save the plot
    print([ 'prob_' graphs(g).name '.eps' ]);
end

% save the simulation results
save('graphs.mat', 'graphs');
save('res.mat', 'res');
```

4. Helper functions:

a. crateNDLattice()

This function creates an ND lattice and returns the adjacency matrix.

```
function [ A ] = createNDLattice( n, d )
% Creates an d-dimensional periodin lattice with size n
% in each dimension. The result has a n^d nodes.
% return A - adjacency matrix

% preallocate adjacency matrix
A = zeros(n^d);

% for each node
for i=1:n^d
    % create a connection with the following node
    % in each dimension
    for j=1:d
        b = mod(i-1 + n^(j-1), n^d) + 1;
        A(i, b) = 1;
        A(b, i) = 1;
    end
end

end
```

b. randomScaleFreeNetwork()

This function creates a random scale free network by simulating a preferential attachment process as described by [3]. It starts with a 2 nodes connected by an edge and sequentially adds new nodes. The new node is connected with the old ones with probabilities proportional to the degrees of the old nodes.

```
function [ A ] = randomScaleFreeNetwork( n )
% Creates a random scale free network with n nodes using
% a preferential attachment process.
% returns A - adjacency matrix

% create the adjacency matrix
A = sparse(n,n);

% add the first two nodes and connect them with an edge
A(1,2) = 1;
A(2,1) = 1;

% initialize the degrees array
d = zeros(1,n);
d(1)=1;
d(2)=1;


% foreach node
for i=3:n

    % calculate the connection probabilities
    p = d / sum(d);
```

```
    % add at least one connection to ensure that
    % the resulting network will be connected
    c = randsample(i-1,1,true,p(1:i-1));
    A(i,c) = 1;
    A(c,i) = 1;
    d(i) = d(i) + 1;
    d(c) = d(c) + 1;

    % generate a random set of neighbours
    r = rand(1,i-1);
    neighbours = (r <= p(1:i-1));

    % connect to the neighbours and update degrees
    A(i, neighbours) = 1;
    A(neighbours, i) = 1;
    d(neighbours) = d(neighbours) + 1;
end
```

c. simulate()

This function simulates the evolution of a population given the parameters of the model. Its return value indicates whether the co-operators or the defectors reached fixation.

```
function [ r ] = simulate( A, maxIterations, Payoffs,
selectionIntensity )
%Perform the simulation given the model parameters
% A             - Adjacency matrix.
% maxIterations - The maximum number of iterations that we
will perform.
% Payoffs       - payoffs matrix
% selectionIntensity - weak selection parameter
% returns r
%           == 1 if cooperators reach fixation
%           == 2 if defectors reach fixation
%           == 0 if maxIterations is reached

n = size(A, 1);

% convert adjacency matrix to adjacency list representation
Alist = cell(1,n);
for i=1:n
    Alist{i} = find(A(i,:));
end

COOPERATOR = 1;
DEFECTOR = 2;

% initialize node strategies to 50% cooperators and
% 50% defectors in a random way
strategy = COOPERATOR * ones(1,n);
strategy( randsample(n, n/2) ) = DEFECTOR;

% calculate initial fitness values
f = fitness(Alist, strategy, Payoffs, selectionIntensity);
```

```
% initialize return value
r = 0;

for i=1:maxIterations

    % perform a death-birth update step
    [strategy f] = deathBirthUpdate(Alist, strategy, Payoffs,
f, selectionIntensity);

    % count the number of cooperators
    ct(i) = sum(strategy == COOPERATOR);

    if ct(i) == n
        % cooperators reached fixation => end simulation
        r = 1;
        break;
    end
    if ct(i) == 0
        % defectors reached fixation => end simulation
        r = 2;
        break
    end
end

end
```

d. fitness()

Calculate the fitness values given a model state.

```
function [ f ] = fitness( A, s, P, w)
%Calculates the fitness vector.
%   A - adjacency lists (cell array of lists)
%   s - node strategies
%   P - payoff matrix
%   w - intensity of selection
%   returns f - fitness vector

n = numel(s);

f = zeros(1,n);

for i=1:n
    % get the neighbours
    neighbours = A{i};

    % calculate a payoffs vector
    payoffs = P(s(i), s(neighbours));

    %calculate the fitness
    f(i) = 1 - w + w * sum(payoffs);
end
end
```

e. deathBirthUpdate()

This function performs a death-birth update step given a model state.

```
function [ ns nf ] = deathBirthUpdate( A, s, P, f, w)
%Perform a state update using the death-birth update.
%   A - adjacency lists (cell array of lists)
%   s - strategies vector
%   P - payoff matrix
%   f - fitness vector
%   w - intensity of selection
%   returns ns - new strategy vector
%   returns nf - new fitness vector

n  = numel(s);

% select a random individual to die
d = randi(n);

% find the neighbours and get their fitness
neighbours = A{d};
neighFitness = f(neighbours);

% crop negative fitness values
neighFitness(neighFitness < 1e-6) = 1e-6;

% select an idividual to replace it with probability
% proportional to the fitness
ni = randsample(numel(neighbours), 1, true, neighFitness);
b = neighbours(ni);

% construct new strategy vector
ns = s;
ns(d) = s(b);

%Construct new fitness vector
nf = f;

% update for the dead and new-born individual
payoffs = P(ns(d), ns(neighbours));
nf(d) = 1 - w + w * sum(payoffs); % weak selection

% update its neighbours fitness
nf(neighbours) = f(neighbours) + w * ...
                 (P(ns(neighbours), ns(d)) - P(s(neighbours),
s(d)))';

end
```

Simulation Results and Discussion

We carried out the simulation as elaborated in the previous section and obtained the following results.

Results:

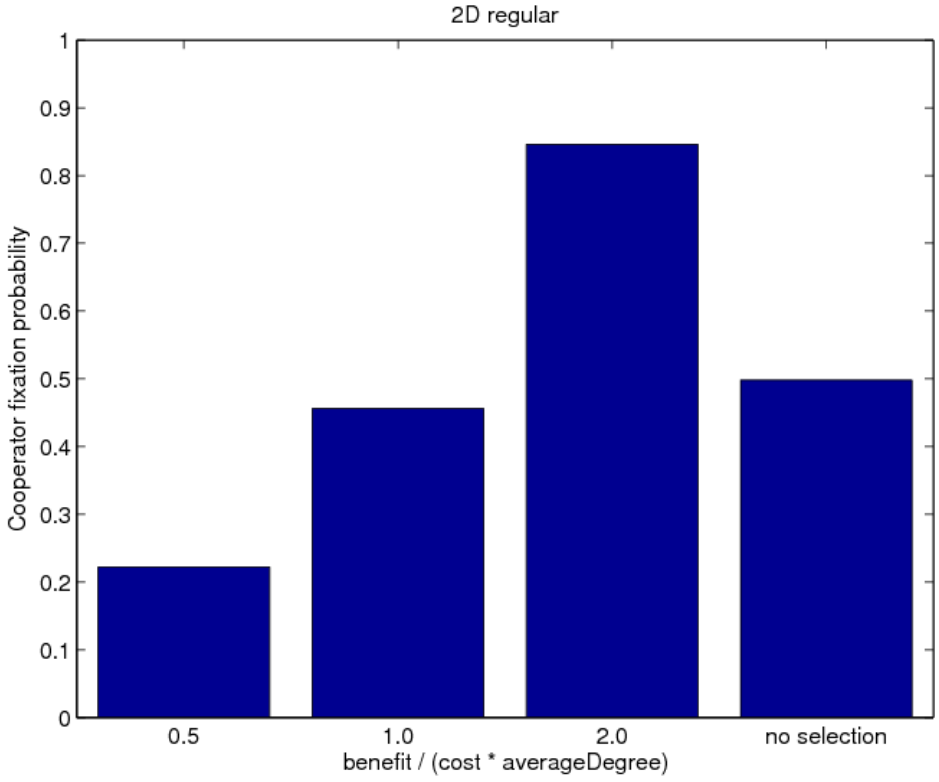
For each graph & for each update rule we want to observe the percentage of fixations of co-operators vs the ratio b/c, to see what happens when k > b/c & k < b/c.

Here to observe this variation, we have plotted b/(ck) against the fixation probability of co-operators, for each of the graphs we have considered.

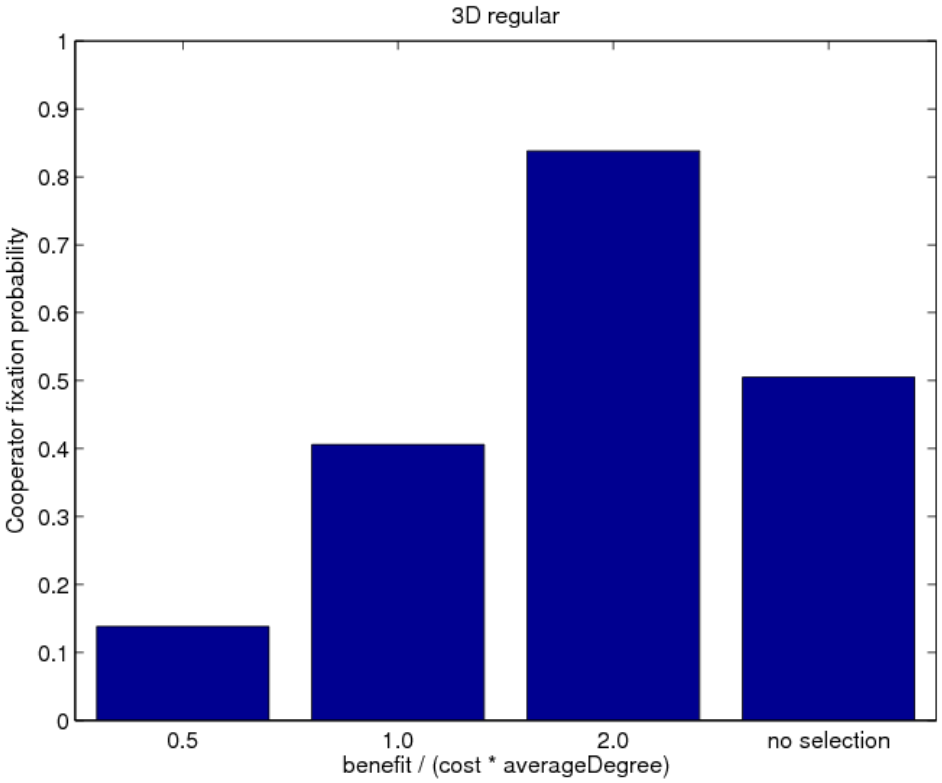
More specifically what we expect to see is the following:
If b/c > k -> b/(ck) > 1, then we should see natural selection is favouring the fixation of co-operators. In addition to that, we should observe that when b/c < k -> b/(ck) < 1, then the fixation of co-operators is not favoured by natural selection.

To see if natural selection favours the fixation or not, we compare the fixation probabilities to the case where there is no selection effect. In every graph, we can see that the fixation probability of the co-operators is 50% when there are no selection effects. So, if natural selection favours the fixation of cooperation, the fixation probability should be greater than 50%.

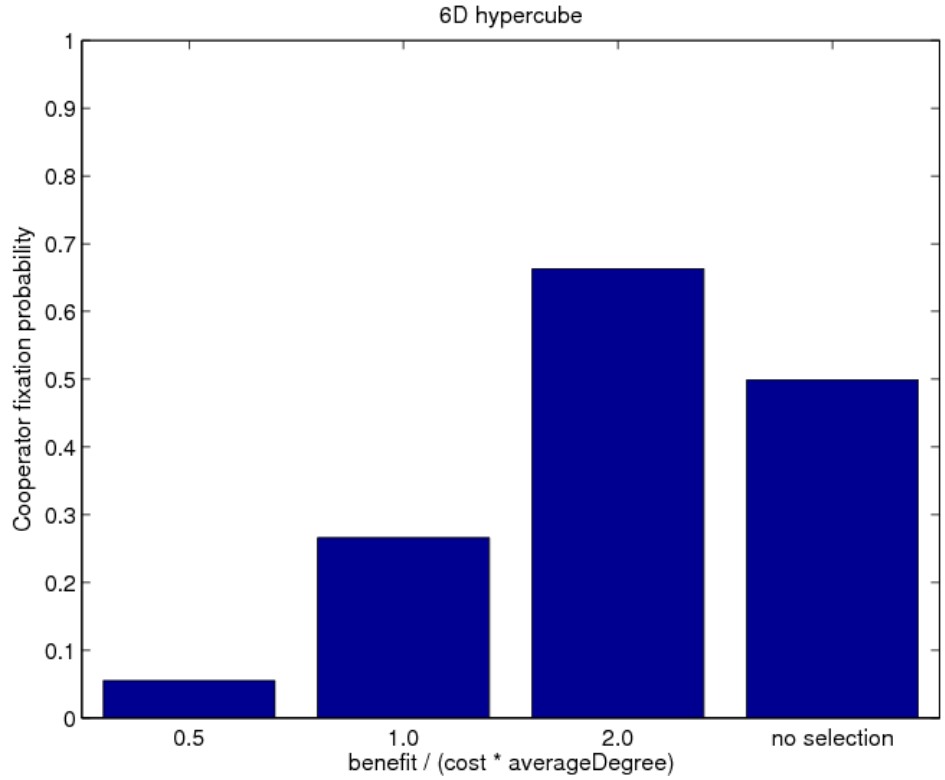
1. Regular lattice: K = 2



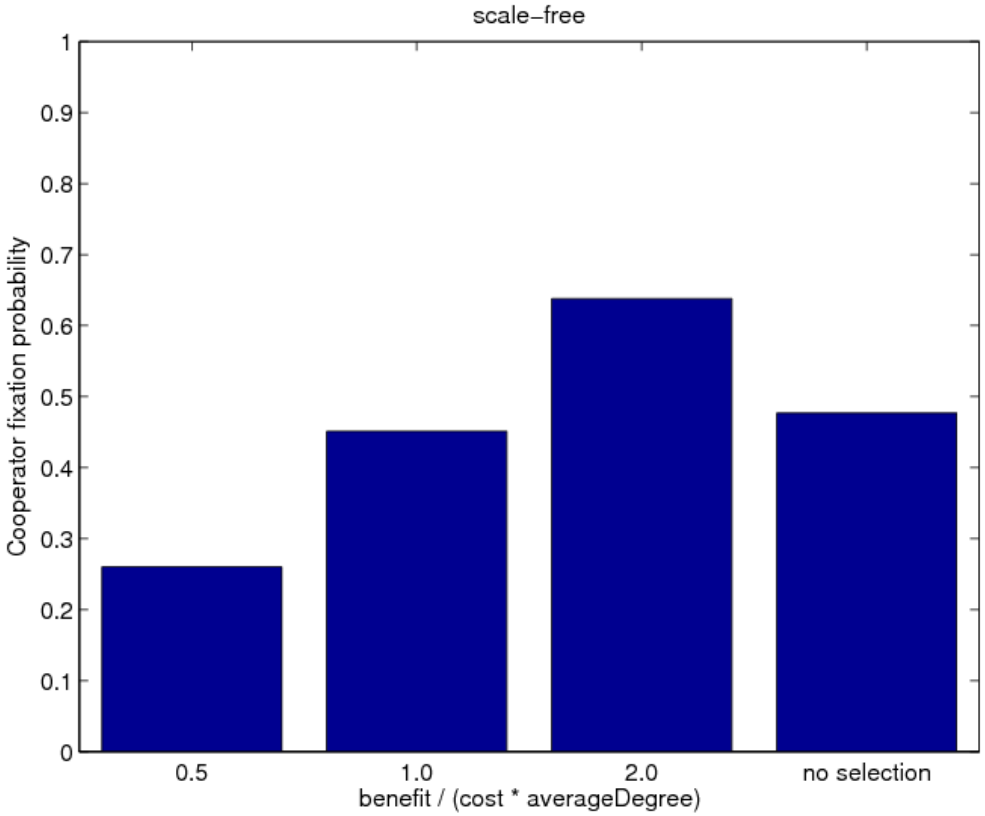
2. Regular Lattice: k = 3



3. Regular Lattice: k = 6



4. Random scale free graph



Discussion:

Regular degree 2 lattice:
Here we observe that when b/(ck) is 0.5, the fixation probability of the co-operators is less than 25%. When b/(ck) is 1, then the fixation probability is almost 50%. Also, when b/(ck) = 2, the fixation probability is above 80%. Clearly, this shows that for this graph, when b/ck > 1, natural selection favours the fixation of co-operators.

Regular degree 3 lattice:
Here we observe that when b/(ck) is 0.5, the fixation probability of the co-operators is around 15%. When b/(ck) is 1, then the fixation probability is almost 50%. Also, when b/(ck) = 2, the fixation probability is above 80%. This shows that for this graph too, when b/ck > 1, natural selection favours the fixation of co-operators.

Regular degree 6 lattice:
Here we observe that when b/(ck) is 0.5, the fixation probability of the co-operators is less than 25%. When b/(ck) is 1, then the fixation probability is almost 30%. Also, when b/(ck) = 2, the fixation probability is above 60%. This shows that for this graph, when b/ck > 1, natural selection favours the fixation of co-operators. However, we can observe that while favoured by natural selection, the fixation probability of co-operators is not as high as the previous 2 graphs when b/ck = 2.

Random scale free network:

Here we observe that when b/(ck) is 0.5, the fixation probability of the co-operators is less than 30%. When b/(ck) is 1, then the fixation probability is almost 50%. Also, when b/(ck) = 2, the fixation probability is above 60%. This shows that for the scale free graph, when b/ck > 1, natural selection favours the fixation of co-operators.

Summary and Outlook

From the simulation results it is quite clear that natural selection favours the fixation of co-operators over defectors when the benefit to cost ratio exceeds the average number of neighbours. i.e. when b/c exceeds the average degree of the graph.

We have simulated this using lattices with different degrees and with scale free graphs. Here, we could observe that when we start with an initial randomly placed population of 50% co-operators and 50% defectors in a structured network, the iterative prisoner's dilemma game with a strategy update rule has a probability over 50% to let the co-operators fixate in the population, if the ratio b/c is greater than the average number of neighbours (k) per node of the graph. The fixation probability of 50% where there is no selection involved is the control case to which we compare our results obtained when there is selection, to observe the impact of natural selection, on the fixation probability of the co-operators.

Moreover, we observe that co-operators are not favoured by natural selection when b/c is less than k.

Therefore, we can conclude that there is room for cooperation to evolve which is determined by the social relatedness (number of neighbours), in the absence of strategic complexity or reputation effects, which is inline with the conclusions of reference [1].

However, as a concluding remark, we would like to highlight the fact that the fixation probability of co-operators decrease with the increase of the average degree of the graph, although it is still higher than 50% when b/c > k. The reason why the increase of the degree has a negative effect on the fixation of the co-operators will be a good question for further research and we have not looked into it since this is out of the scope of our goals for this project.

References

1. Ohtsuki, Hauert, Lieberman & Nowak (2006). "A simple rule for the evolution of cooperation on graphs and social networks**".** Nature Vol 441|25 May 2006|doi:10.1038
2. Lieberman, E., Hauert, C., and Nowak, M.A. (2005) Evolutionary dynamics on graphs. Nature. 433, 312–316
3. R. Albert, A. L. Barabási (2002). "Statistical mechanics of complex networks". *Reviews of Modern Physics* **74**: 47–97. doi:10.1103/RevModPhys.74.47.