Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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

Project Report

Success-driven Migration Applied to Spatial Games

Damian Karrer & Dominik Spicher

Zürich
May 2010

Statement of Originality

We hereby declare having conducted this report independently, having used no other than the declared references and having highlighted all passages taken from them. Furthermore we declare having created all parts of this report for this project uniquely.

Damian Karrer Dominik Spicher

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.

Damian Karrer Dominik Spicher

Table of contents

1 Individual Contributions

Dominik implemented a first version of the simulation script. Damian refined it, and fixed some (major) bugs. Both the various simulations and their interpretation were done by both of us. Most of the introduction and the sections on game-theoretic aspects were written by Dominik, while Damian concentrated on the sections concerning the model and the simulation results.

2 Introduction

> *Cooperation or co-operation is the process of working or acting together, which can be accomplished by both intentional and non-intentional agents. In its simplest form it involves things working in harmony, side by side, while in its more complicated forms, it can involve something as complex as the inner workings of a human being or even the social patterns of a nation.*[1]

Cooperation among individuals is, as suggested by the above description, ubiquitous in our world. Often though, it is unclear how cooperation among selfish individuals can be achieved. This holds in particular for the Prisoner's Dilemma, where individuals are tempted to defect, even though bilateral cooperation would achieve the highest combined profit. In [2], it is shown that for this game, there exist mechanisms which can promote cooperation or even allow it to burst in a defecting environment under noisy conditions.

In this paper, we analyse the effect of those mechanisms, i.e. imitation and success-driven migration in two different games, the Prisoner's Dilemma and the Snowdrift Game. We first want to reproduce some of the results of the paper for the Prisoner's Dilemma. Taking a similar approach for the Snowdrift Game, we want to investigate in what ways those mechanisms can similarly support and promote migration here. It can be expected that cooperation will be easier to achieve for this game, because the potential risk of defecting is higher than in the Prisoner's Dilemma. It remains to be shown how cooperation in the Snowdrift Game reacts to imitation and success-driven migration.

In order to do so, we design a generic MATLAB program to simulate either game. We will run experiments under different noise conditions and compare the influence of success-driven migration on cooperation. For the Snowdrift Game it is already suggested in [2] that cooperators and cheaters will coexist in the imitation-only case, whereas for the Prisoner's Dilemma overall defection is expected.

Our research questions are as follows:

- To what extent does success-driven migration promote cooperation in the Snowdrift Game in a noisy environment?
- How does this effect compare to the results gained for the Prisoner's Dilemma?

3 Game-theoretic Aspects

3.1 Games

3.1.1 Prisoner's Dilemma

The Prisoner's Dilemma is a game between two individuals which are both presented with the options to either cooperate or defect. The payoffs for the two individuals are such, that defecting is the dominant strategy. However, and this applies especially to the iterated version (cf. section 3.2), the payoffs would be higher for bilateral cooperation than for bilateral defection. Figure 1 shows a payoff matrix for the Prisoner's Dilemma.

A / B	cooperate	defect
cooperate	1 / 1	0 / 1.3
defect	1.3 / 0	0.1 / 0.1

Fig 1: Payoff matrix for the Prisoner's Dilemma

The importance of the game and the vast attention it has received in research stems from the fact that it models a situation which is very common in politics, economics and everyday life. For example, consider two parties sharing a limited natural resource. In the long term, it would be better for both to consume as little as possible of it. However, they both are tempted to use more of it to gain a short-term advantage.

If we denote the reward for bilateral cooperation by R, the punishment for bilateral defection by P, the temptation to unilaterally defect by T and the “sucker's payoff” by S, the Prisoner's Dilemma is characterized by $T > R > P > S$ and $2R > T + S$.

3.1.2 Snowdrift Game

The Snowdrift Game (also called “Chicken”) is similar to the Prisoner's Dilemma but the worst possible outcome is for both actors do defect. A common example illustrates this game: Two drivers drive towards each other on a collision course. If one player swerves (i.e. cooperates) and the other doesn't, he will be called a “chicken”. However if both refuse to swerve (i.e. defect), they will both die. A typical payoff matrix is shown in figure 2. The Snowdrift Game is characterized by $T > R > S > P$. It is similar to the Prisoner's Dilemma in that a situation of mutual cooperation is unstable because of $T > R$. The difference, however, is that the payout for bilateral defection is lower than the payout of unilateral cooperation for the cooperator (i.e. $S > P$ rather than $P > S$). Therefore, there exists no dominant strategy for the Snowdrift Game.

A / B	cooperate	defect
cooperate	3 / 3	2 / 4
defect	4 / 2	0 / 0

Fig 2: Typical payoff matrix for the Snowdrift Game

3.2 Iterated and Spatial Games

To make the games described above more interesting, one can consider two extensions to them: In the iterated version, players not play the game once, but many times. This might allow them, depending on the model that is used, to play according to a specific strategy and even decide how to play depending on the former actions of the other player.

For the spatial version of a game, one places many agents on a grid, where each agent plays the game with its neighbors. This allows for some interesting mechanisms to take place, for example migrating into a “better” neighborhood.

We will consider a combination of the two, i.e. the iterated spatial versions of the two games.

3.3 Neighborhoods

When considering spatial games, it is useful to define neighborhoods of a cell. We work with two different neighborhoods, the Moore neighborhood and the von Neumann neighborhood. They are visualized in figure 3.

(a) (b)

Fig. 3: The Moore (a) and von Neumann (b) neighborhood of order one

4 Description of the Model and Implementation

We consider the spatial version of the Prisoner's Dilemma and the Snowdrift Game on a rectangular grid with periodic boundary conditions (thus the grid is actually treated as a torus, with the left- and rightmost as well as the upper- and lowermost cells treated as neighbors). At the beginning of the experiment, agents are randomly distributed on the grid up to a specified density (a parameter). In each time step, agents get updated in a randomly order. The agents play the game with their four direct neighbors (von Neumann neighborhood of order one), resulting in their current income. Each one then first performs migration (if it is switched on), then imitation (if on). For both migration and imitation, the kind of neighborhood and its order are parameters of the experiment. We have implemented two different migration strategies:

- Concrete migration: An agent looks at the mean income of all the neighbors of the cell it inspects and chooses to migrate to the cell with the highest one.
- Hypothetical migration: An agent fictitiously plays the game with the neighbors of the cell it inspects, that is, it calculates its bargain if it were located there. The agent will migrate to the cell with the highest hypothetical payoff.

Of course, only empty cells are considered as possible places to migrate to.

The games can be simulated either with or without noise. We consider three different kinds of noise: Noise 1 means that each agent will spontaneously change its strategy with probability r. If so, it becomes a cooperator with probability q, and with probability 1-q it turns into a defector. If an agent mutates its strategy, it will not imitate another agent in this time step (thus, normal imitation, if set on, is performed with probability 1-r). Both parameters r and q can be set at the beginning of the run. Noise 1 provides a certain level of independence of the final fraction of cooperators from the initial state [2]. With noise 2, there will be random relocations. An agent is selected with probability r (which is the same parameter as for noise 1) to randomly relocate, i.e. move to an arbitrary free square. If an agent relocates randomly, it won't additionally perform normal migration. This noise poses a challenge for possible clusters of cooperation to survive, because they can be invaded by randomly relocating defectors [2]. For noise 3, strategy mutations and random relocations are combined. An agent which is selected with probability r first randomly relocates and then mutates its strategy.

During execution, various values are collected for statistics, for example the number of migrated agents or the mean income of a cooperator and a cheater. Examples of those results are shown in section five.

5 Simulation Results and Discussion

The choice of the parameters was done in accordance to the experiment parameters in [2]. This allowed us to a certain degree to check our models for bugs in the implementation. For the Prisoner's Dilemma the values are T = 1.3, R = 1, P = 0.1 and S = 0. For the Snowdrift Game, we used the values T = 4, R = 3, P = 0, S = 2. The other simulation parameters are the same for both games.

We have a 49x49 grid, of which we fill half the squares with agents at initialisation. The agents are distributed randomly and cooperators and cheaters both make up 50% at the beginning.

The applied migration strategy is hypothetical, the migration range is a Moore neighborhood of order five. The agents imitate others in a von Neumann neighborhood of order one.

For the noise, we set both the parameters r and q to 0.05, which means randomly resetting the strategy (noise 1) will produce much more defectors than cooperators.

We first want to investigate what formations, if any, the agents build after a relatively big number of iterations. In order to make a statement about the effects of imitation and success-driven migration, we consider a specific run of the simulation. In addition, we run the experiment for the four different noise conditions to see if those effects are robust with respect to strategy mutation and random relocation.

Figures 4 and 5 show an example run after 100 iterations for the Prisoner's Dilemma and the Snowdrift Game respectively. Cooperators are blue, whereas defectors are red; white indicates empty cells.

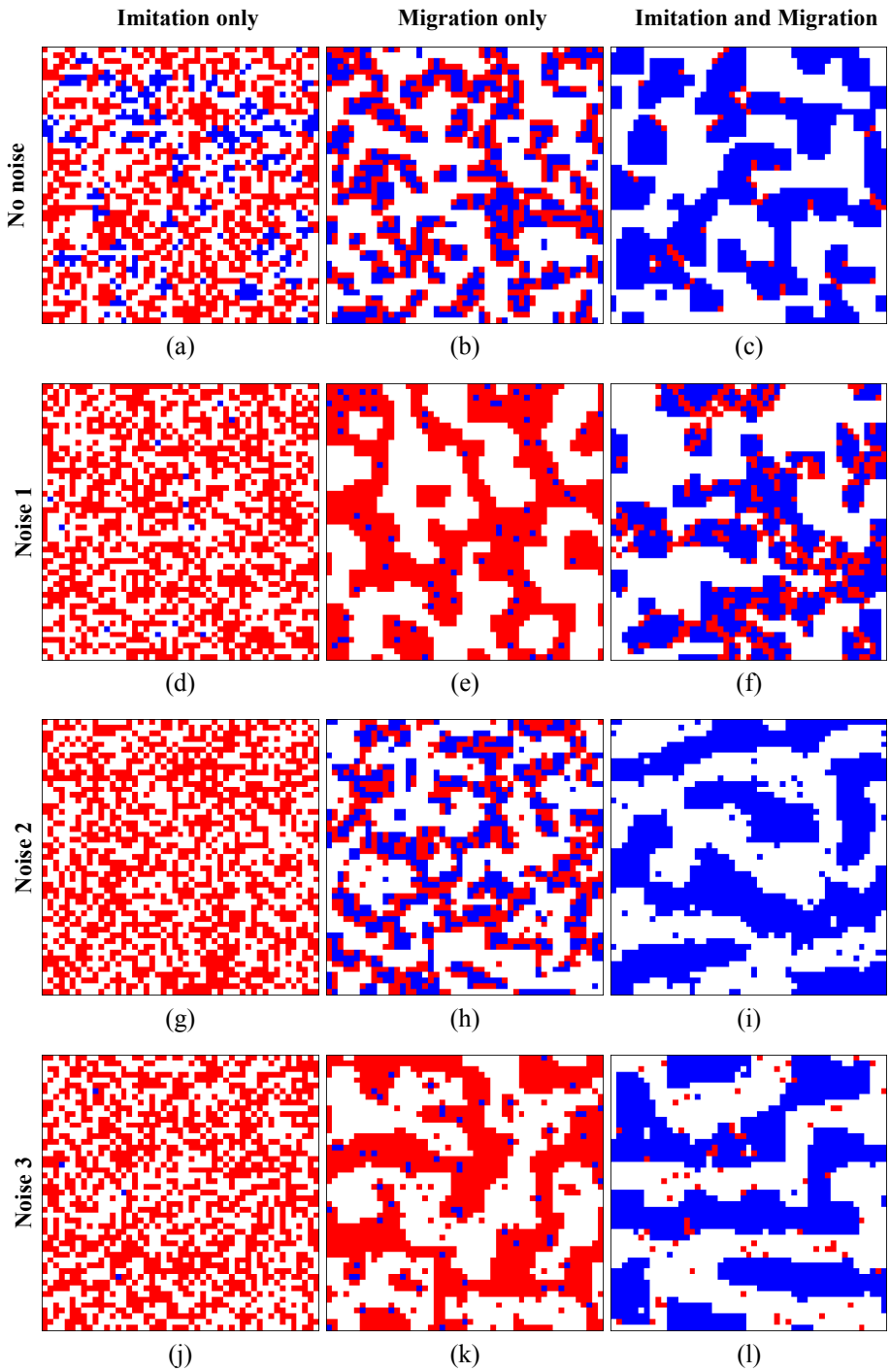


Fig. 4: Simulation results for Prisoner's Dilemma

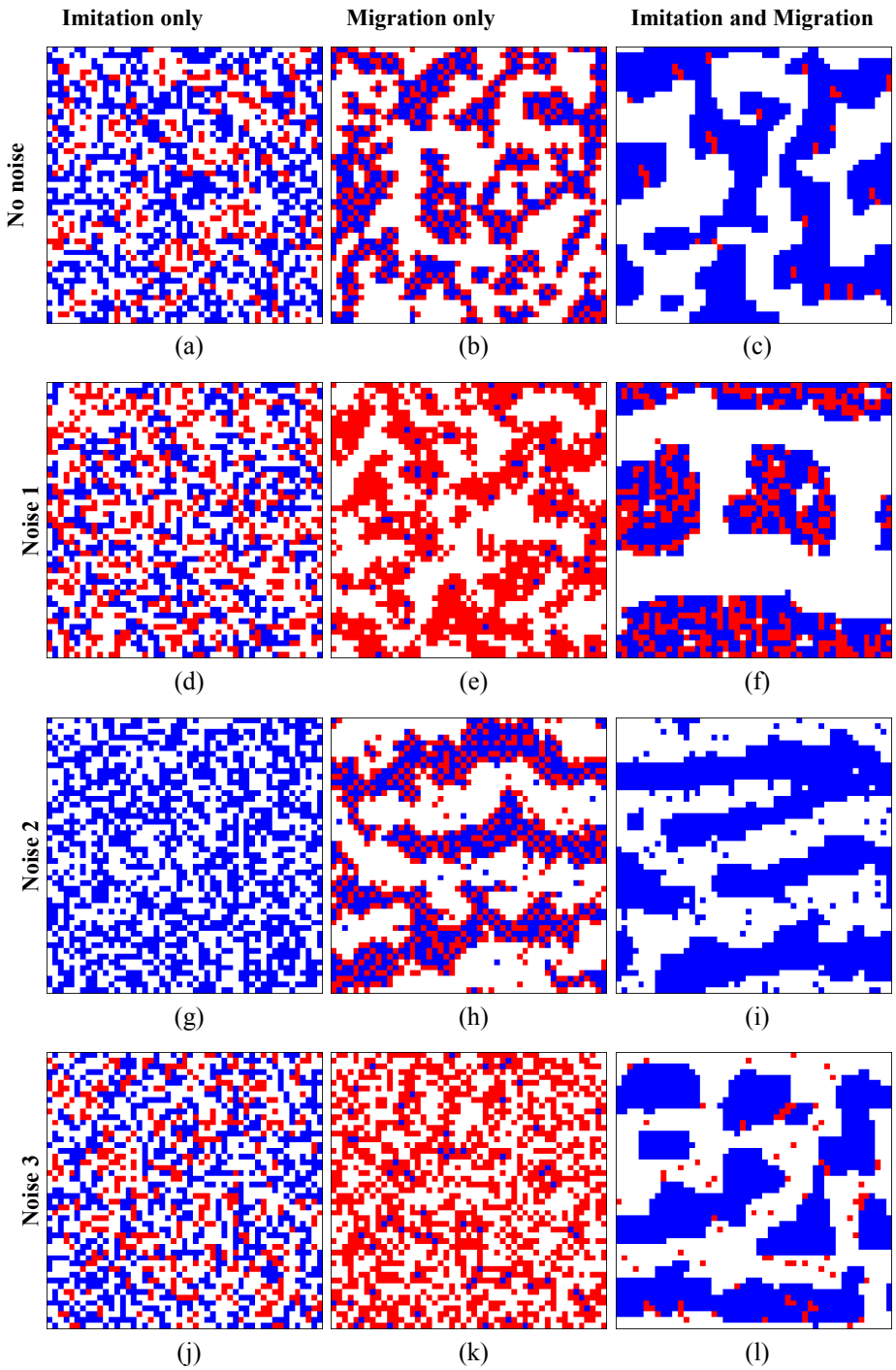


Fig. 5: Simulation results for Snowdrift Game

5.1 Prisoner's Dilemma

The results we got for the Prisoner's Dilemma are in accordance with the results from the given paper [2].

5.2 Snowdrift Game

In the migration only cases one can clearly see the effects of strategy mutations (e, k), as most individuals change their strategy to defecting. Without strategy mutation (b, h) the number of defectors and cooperators stays the same. Through migration the agents organise themselves in groups to achieve higher incomes.

As the expected payoff for cooperation is higher than that for defection in the Snowdrift Game , it is to be expected that the addition of migration to imitation raises the number of cooperating agents. This effect is much smaller with noise 1 (f), however, as compared to the other cases with both imitation and migration (c, i, l).

With noise 1 there is strategy mutation (f). Agents that mutate their strategy turn with probability 0.95 to defectors. As there are much more cooperators, the new defectors are almost exclusively surrounded by cooperators and therefore their income is very high. Because of that, they get imitated immediately. This effect is reflected in about 200 imitations of defectors per time step (cf. figure 6). In spite of this, their total number stays more or less constantly at 400 (figure 7). So we can conclude that the environment is generally hostile for defectors, but because of the aforementioned strategy mutation followed by imitation effect, the defectors are not being wiped out.

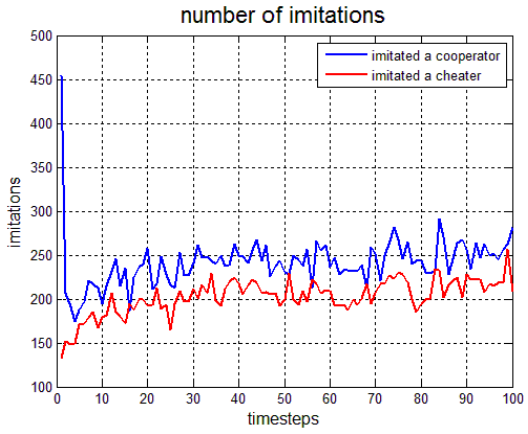


Fig. 6

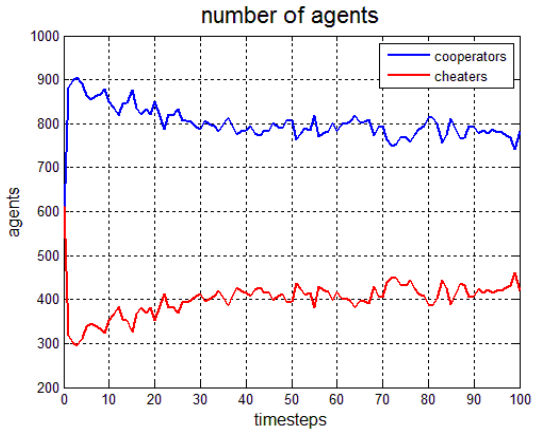


Fig. 7

Simulation values for the Snowdrift Game with noise 1, imitaion and migration

Because there is no relocation at all without migration and without noise (a), defectors survive as there are no cooperators in their imitation range or else because the payoff of the cooperators in

range is statically lower. With random relocation however (g), the remaining cheaters sooner or later get relocated and imitate cooperators, and are thus wiped out completely. With added migration (i), the cooperators migrate into large groups for higher payoff, whereas without migration (g) the pattern is speckled.

5.3 Comparison

In the Snowdrift Game it is more attractive to cooperate than in the Prisoner's Dilemma, as the imitation only cases show.

The difference between the two games is further made apparent when comparing the migration only cases without noise (b). In both games the defectors seek to be adjacent to cooperators to earn the temptation T. But while in the prisoners dilemma S is smaller than P, which makes cooperators avoid contact with cheaters, in the Snowdrift Game P is higher than S. This makes it more acceptable for cooperators to have cheaters as neighbors, while at the same time the cheaters earn less from their kind. This results in a more speckled pattern, as defectors in the Snowdrift Game migrate into groups of cooperators, while in the Prisoner's Dilemma they surround the cooperators and feed from them. This can also be observed with noise 2 added (h).

In the migration only case with strategy mutation (e, k), the defectors in the Prisoner's Dilemma migrate into large groups, as they still earn the punishment P. In the Snowdrift Game, however, P is the smallest payoff and zero in our simulations. Therefore it doesn't matter for a defector whether it is located next to other defectors or empty cells, which results in a more speckled pattern.

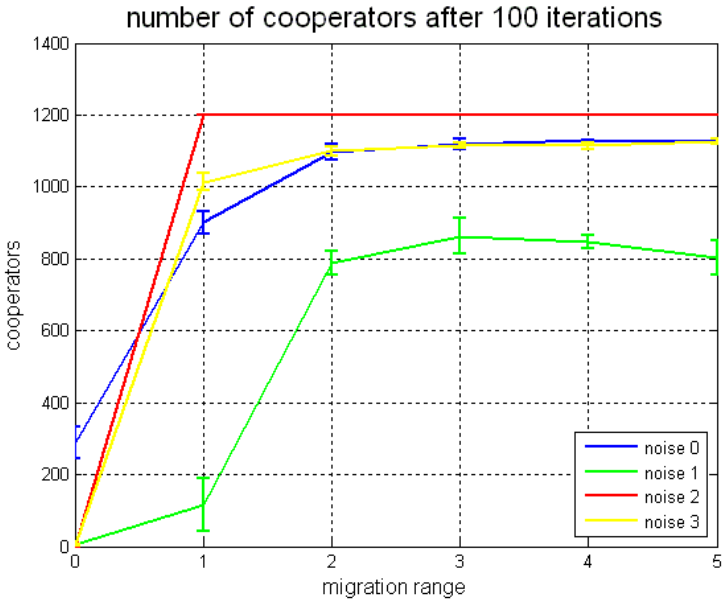
5.4 Influence of the Migration Range

In a second experiment, taking a similar approach as in [3], we want to measure the influence of the migration range on cooperation. We measure the number of cooperators after 100 iterations for different neighborhoods, namely Moore neighborhoods of order zero (i.e. imitation only) to five. We do this for each noise condition and for both games. Results are shown in figure 8.

As we can see, cooperation in the Prisoner's Dilemma benefits much more from migration than in the Snowdrift Game, where cooperation is already on a high level for imitation only (cf. figure 5).

Furthermore, it becomes apparent, that for the used grid which is relatively small (49x49 cells), a migration neighborhood with range greater than two doesn't amplify the level of cooperation much more. This is important because a greater migration neighborhood significantly reduces the numerical performance of our model.

(a) **Prisoner's Dilemma**



(b) **Snowdrift Game**

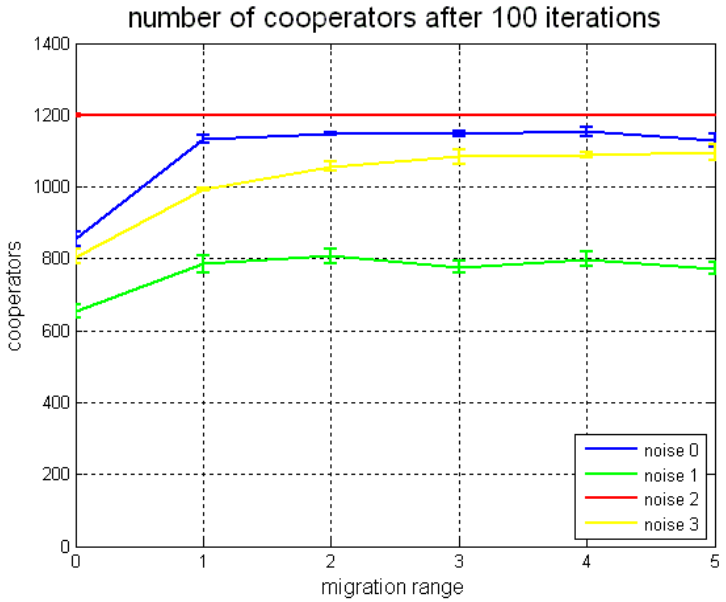


Fig. 8: These plots show the average number of cooperators (of five runs) after 100 iterations for the two games. Vertical bars indicate the standard deviation in each direction.

6 Summary and Outlook

In this section we provide answers to our research questions and elaborate on possible extensions to our model.

To what extent does migration promote cooperation in the Snowdrift Game in a noisy environment?

In the Snowdrift Game, cooperators migrate to each other to form large groups. Moreover, defectors migrate too and eventually have cooperators in their migration range, thus enhancing cooperation, as cooperators are generally more successful in the Snowdrift Game and therefore get imitated. Noise 1 (strategy mutations) lessens the positive effect of migration on cooperation, while the other forms of noise do not disturb cooperation.

How does this effect compare to the results gained for the Prisoner's Dilemma?

With both imitation and migration, the results are about the same for both the Snowdrift Game and the Prisoner's Dilemma. With migration only however there are some differences. With strategy mutation (noise 1 and noise 3) the defectors distribute themselves over the cooperators in the Snowdrift Game, whereas in the Prisoner's Dilemma they rather surround small groups of cooperators. In the migration only case without random relocation (no noise and noise 2) it can be observe that in the Snowdrift Game it is not desirable for cheaters to migrate to each other, in contrast to the Prisoner's Dilemma.

One could further study the effect of migration on the Snowdrift Game and the Prisoner's Dilemma with slightly other rules or parameters. E.g. in our simulations we exclusively applied a von Neumann imitation range and a hypothetical migration strategy. One could simulate the games with a Moore imitation neighborhood and/or the concrete migration strategy. Success-driven migration could also be applied to other spatial games.

7 References

[1] Wikipedia, „Cooperation“
http://en.wikipedia.org/wiki/Cooperation

[2] Helbing, Dirk; Yu, Wenjian, „The outbreak of cooperation among success-driven individuals under noisy conditions“, PNAS, 2009

[3] Helbing, Dirk; Ju, Wenjian, „Migration as a mechanism to promote cooperation“, World Scientifc, Advances in Complex Systems, 2008

8 Appendix A: MATLAB-Code

8.1 Main Script

```
%main script for simulation

%shared variables with functions. Convention: global variables start with
%an upper-case, local ones with a lower-case letter
global G M N R S T P Income NeighIncome

%------------
% Parameters
%------------
M = 49;                 %grid height
N = 49;                 %grid width
G = zeros(M,N);         %grid
p = 0.5;                %density of agents
tmax = 100;             %number of iterations

game = 'sd';            %'sd' (snowdrift) or 'pd' (prisoner's dilemma)

migration = 'on';
rmig = 5;               %migration range
migneigh = 'moore';     %'moore' or 'neumann'
mig_strategy = ...
    'hypothetical';     %'concrete' or 'hypothetical'

imitation = 'on';
rim = 1;                %imitation range
imneigh = 'neumann';    %'moore' or 'neumann'

noise = 3;              %0: no noise;
                        %1: noise 1 (strategy mutation)
                        %2: noise 2 (random relocation);
                        %3: noise 3 (noise 1 & 2 combined)
r = 0.05;               %probability of strategy mutation / random relocation
q = 0.05;               %probability of turning into a cooperator
                            %within a strategy mutation

plots = ('on');
%------------


%------------
% Payoffs
%------------
if strcmp(game,'pd')
    %these values are used for the prisoner's dilemma
    T = 1.3;
    R = 1;
    P = 0.1;
    S = 0;
elseif strcmp(game,'sd')
    %these values are used for the snowdrift game
    T = 4;
    R = 3;
    P = 0;
    S = 2;
else
    error('fail: wrong game parameter');
```

```
end
%------------

%------------
% Initializations
%------------
agentscount = N*M*p;     %number of agents
Income = zeros(M,N);     %incomes (accumulated payoffs)
NeighIncome = zeros(M,N); %neighborhood incomes
cheatcount = 0;          %number of cheaters
coopcount = 0;           %number of cooperators
migcheat = [];           %number of migrated cheaters
migcoop = [];            %number of migrated cooperators
imcheat = [];            %number of imitations of a cheater
imcoop = [];             %number of imitations of a cooperator
inccoop = [];            %cooperators incomes
inccheat = [];           %cheaters incomes
%------------



%------------
% distribute agents randomly
%------------
curragents = 0;         %current number of people on the grid
while curragents < agentscount
    indexi = floor(M*rand())+1;
    indexj = floor(N*rand())+1;
    if G(indexi,indexj) ~= 0
        continue;
    end
    % generate a random number between 1 and 2
    G(indexi,indexj) = floor(2*rand())+1;
    if G(indexi,indexj) == 1
        coopcount(1) = coopcount(1) + 1;
        curragents = curragents + 1;
    elseif G(indexi,indexj) == 2
        cheatcount(1) = cheatcount(1) + 1;
        curragents = curragents + 1;
    end
end
%------------

%claculation of initial incomes
for i=1:M
    for j=1:N
        calculate_income(i,j); %stored in Income
    end
end

% calculation of initial neighbourhood incomes
for i=1:M
    for j=1:N
        calculate_neighbourhood_income(i,j); %stored in NeighIncome
    end
end

%figure to plot the agents in
if strcmp(plots,'on')
    figure
```

```
    hold on
end

%------------
% simulation over time
%------------
for timestep = 1:tmax

    %reset the variables which have to be reset for every timestep
    migcount = 0;               %number of migrated people
    migcoopcount = 0;           %number of migrated cooperators
    migcheatcount = 0;          %number of migrated cheaters
    imcoopcount = 0;            %number of imitations of a cooperator
    imcheatcount = 0;           %number of imitations of a cheater

    %for the number of cooperators and cheaters,
    %starting values for next iteration are those from the previous ones
    coopcount = [coopcount; coopcount(end)];
    cheatcount = [cheatcount; cheatcount(end)];

    if strcmp(plots,'on')
        colormap([1 1 1; 0 0 1; 1 0 0]);     % Define colors: white, blue, red
        clf;                                 % Clear figure
        imagesc(G, [0 2]);                   % Display grid
        pause(0.01);
        box on;
    end

    %store locations of all agents in agentcoord
    agentsleft = G ~= 0;
    agentcoord = [];
    for s = 1:M
        for t = 1:N
            if agentsleft(s,t)
                agentcoord = [agentcoord; s, t];
            end
        end
    end

    %while there are agents which have not been updated yet, choose one
    %randomly and update him
    while size(agentcoord,1) > 0
        randcoord = floor(size(agentcoord,1)*rand())+1;
        i = agentcoord(randcoord,1);
        j = agentcoord(randcoord,2);
        agentcoord = [agentcoord(1:randcoord-1,:);
                      agentcoord(randcoord+1:end,:)];

        %the boolean variable noiseeffect indicates if a normal or noisy
        %update is taking place
        noiseeffect = (rand() < r);

        %those boolean variables store information if the income needs to
        %be racalculated and if the agent has moved
        recalcincome = false;
        relocated = false;

        %either perform random relocation ...
        if ((noise == 2) || (noise == 3)) && noiseeffect
            i2 = floor(M*rand())+1;
            j2 = floor(N*rand())+1;
```

```
    while (G(i2,j2) ~= 0)
        i2 = floor(M*rand())+1;
        j2 = floor(N*rand())+1;
    end
    G(i2,j2) = G(i,j);
    G(i,j) = 0;
    relocated = true;
    inew = i2;
    jnew = j2;
    recalcincome = true;

%... or normal migration
elseif strcmp(migration,'on')
    %parameters for migration candidate
    imax = i;
    jmax = j;
    if strcmp(mig_strategy,'concrete')
        maxincome = NeighIncome(i,j);
    elseif strcmp(mig_strategy,'hypothetical')
        maxincome = Income(i,j);
    else
        error('fail: wrong migration strategy parameter');
    end
    %search for largest income in area specified
    %by the migration radius and neighborhood
    if strcmp(migneigh,'moore')
        neighbours = mooreneigh(rmig);
    elseif strcmp(migneigh,'neumann')
        neighbours = neumannneigh(rmig);
    else
        error('fail: wrong migration neighbourhood parameter');
    end
    for ik = 1:length(neighbours)
        i2 = i + neighbours(ik,1);
        j2 = j + neighbours(ik,2);
        %ensure dynamic grid boundaries
        if (i2 < 1)
            i2 = i2 + M;
        end
        if (i2 > M)
            i2 = i2 - M;
        end
        if (j2 < 1)
            j2 = j2 + N;
        end
        if (j2 > N)
            j2 = j2 - N;
        end
        if (G(i2,j2) == 0)
            if strcmp(mig_strategy,'concrete')
                if NeighIncome(i2,j2) > maxincome
                    imax = i2;
                    jmax = j2;
                    maxincome = NeighIncome(i2,j2);
                end
            elseif strcmp(mig_strategy,'hypothetical')
                hinc = calculate_hypothetical_income(i2,j2,G(i,j));
                if hinc > maxincome
                    imax = i2;
                    jmax = j2;
                    maxincome = hinc;
```

```
                    end
                else
                    error('fail: wrong migration strategy parameter');
                end
            end
        end
        %check if a migration candidate was found
        if (imax ~= i || jmax ~= j)
            G(imax,jmax) = G(i,j);
            G(i,j) = 0;
            migcount = migcount + 1;
            recalcincome = true;
            relocated = true;
            inew = imax;
            jnew = jmax;
            if G(imax,jmax) == 1
                migcoopcount = migcoopcount + 1;
            elseif G(imax,jmax) == 2
                migcheatcount = migcheatcount + 1;
            end
        end
    end

    if recalcincome
        % recalculate incomes which have changed
        % => cells in moore-neighbourhood of range 1 of origin
        %    and destination
        calculate_income(i,j);
        calculate_income(i2,j2);
        neighbours = mooreneigh(1);
        for ik = 1:8
            calculate_income(i+neighbours(ik,1),j+neighbours(ik,2));
            calculate_income(i2+neighbours(ik,1),j2+neighbours(ik,2));
        end
        % recalculate neighbourhood incomes which have changed
        % => cells in moore-neighbourhood of range 2 of origin
        %    and destination
        calculate_neighbourhood_income(i,j);
        calculate_neighbourhood_income(i2,j2);
        neighbours = mooreneigh(2);
        for ik = 1:24
            calculate_neighbourhood_income(i+neighbours(ik,1),j+neighbours(i
k,2));
            calculate_neighbourhood_income(i2+neighbours(ik,1),j2+neighbours
(ik,2));
        end

    end
    if relocated
        i = inew;
        j = jnew;
    end
    recalcincome = false;
    %either perform strategy mutation ...
    if ((noise == 1) || (noise == 3)) && noiseeffect
        mutatetocoop = (rand() < q);
        if mutatetocoop && (G(i,j) == 2)
            G(i,j)=1;
            cheatcount(end)=cheatcount(end)-1;
            coopcount(end)=coopcount(end)+1;
            recalcincome = true;
```

```
            elseif ~mutatetocoop && (G(i,j) == 1)
                G(i,j)=2;
                cheatcount(end)=cheatcount(end)+1;
                coopcount(end)=coopcount(end)-1;
                recalcincome = true;
            end

        %... or normal imitation
        elseif strcmp(imitation,'on')
            if G(i,j) == 0
                continue
            end
            %parameters for imitation candidate
            maxincome = Income(i,j);
            imax = i;
            jmax = j;

            if strcmp(imneigh,'moore')
                neighbours = mooreneigh(rim);
            elseif strcmp(imneigh,'neumann')
                neighbours = neumannneigh(rim);
            else
                error('fail: wrong imitation neighbourhood parameter');
            end

            for ik = 1:length(neighbours)
                i2 = i + neighbours(ik,1);
                j2 = j + neighbours(ik,2);
                %ensure dynamic grid boundaries
                if (i2 < 1)
                    i2 = i2 + M;
                end
                if (i2 > M)
                    i2 = i2 - M;
                end
                if (j2 < 1)
                    j2 = j2 + N;
                end
                if (j2 > N)
                    j2 = j2 - N;
                end
                if Income(i2,j2) > maxincome
                    imax = i2;
                    jmax = j2;
                    maxincome = Income(i2,j2);
                end
            end
            %check if an imitation candidate was found
            if ((imax ~= i) || (jmax ~= j)) && (G(imax,jmax) ~= 0) && (G(i,j) ~=
G(imax,jmax))
                if (G(imax,jmax) == 1) && (G(i,j) == 2)
                    G(i,j)=1;
                    cheatcount(end)=cheatcount(end)-1;
                    coopcount(end)=coopcount(end)+1;
                    imcoopcount = imcoopcount + 1;
                    recalcincome = true;
                elseif (G(imax,jmax) == 2) && (G(i,j) == 1)
                    G(i,j)=2;
                    cheatcount(end)=cheatcount(end)+1;
                    coopcount(end)=coopcount(end)-1;
                    imcheatcount = imcheatcount + 1;
```

```
                    recalcincome = true;
                end
            end
        end

        % recalculate incomes which have changed
        % => cells in moor-neighbourhood of range 1 of origin
        if recalcincome
            calculate_income(i,j);
            neighbours = mooreneigh(1);
            for ik = 1:8
                calculate_income(i+neighbours(ik,1),j+neighbours(ik,2));
            end
            % recalculate neighbourhood incomes which have changed
            % => cells in moore-neighbourhood of range 2 of origin
            calculate_neighbourhood_income(i,j);
            neighbours = mooreneigh(2);
            for ik = 1:24
                calculate_neighbourhood_income(i+neighbours(ik,1),j+neighbours(i
k,2));
            end
        end
    end
    if coopcount(end) == 0
        inccoop = [inccoop;0];
    else
        inccoop = [inccoop;sum(sum(Income(G==1)))/coopcount(end)];      %mean
income of cooperators
    end
    if cheatcount(end) == 0
        inccheat = [inccheat;0];
    else
        inccheat = [inccheat;sum(sum(Income(G==2)))/cheatcount(end)];   %mean
income of cheaters
    end
    migcheat = [migcheat;migcheatcount];
    migcoop = [migcoop;migcoopcount];
    imcoop = [imcoop;imcoopcount];
    imcheat = [imcheat;imcheatcount];
end
%------------

%------------
% plot the collected values
%------------
if strcmp(plots,'on')
    %plots the number of cooperators and cheaters
    figure
    plot(0:tmax,coopcount,'b',0:tmax,cheatcount,'r','LineWidth',2);
    title('number of agents','FontSize',16);
    xlabel('timesteps','FontSize',12);
    ylabel('# agents','FontSize',12);
    legend('cooperators','cheaters');
    box on
    grid on

    %plots the number of migrated people distinguishing
    %between cooperators and defectors
    figure
    plot(1:tmax,migcoop,'b',1:tmax,migcheat,'r','LineWidth',2);
    title('number of migrated agents','FontSize',16);
```

```
    xlabel('timesteps','FontSize',12);
    ylabel('# of migrated people','FontSize',12);
    legend('cooperators','cheaters');
    box on
    grid on

    %plots the number of imitations distinguishing
    %imitation of cooperators and defectors
    figure
    plot(1:tmax,imcoop,'b',1:tmax,imcheat,'r','LineWidth',2);
    title('number of imitations','FontSize',16);
    xlabel('timesteps','FontSize',12);
    ylabel('imitations','FontSize',12);
    legend('imitated a cooperator','imitated a cheater');
    box on
    grid on

    %plots the mean income of cooperators and cheaters
    figure
    plot(1:tmax,inccoop,'b',1:tmax,inccheat,'r','LineWidth',2);
    title('mean income of agents','FontSize',16);
    xlabel('timesteps','FontSize',12);
    ylabel('mean income','FontSize',12);
    legend('cooperators','cheaters');
    box on
    grid on
end
%------------
```

8.2 Auxiliary Functions

```
function a = neumannneigh(n)
%returns the relative coordinates of all the members of the
%vonNeumann-neighbourhood of order n, (0,0) excluded
%each row in a corresponds to a cell
    a = [];
    for i = -n:n
        for j = -abs(n-abs(i)):abs(n-abs(i))
            if (i ~= 0) || (j ~= 0)
                a = [a;i,j];
            end
        end
    end
end
```

```
function a = mooreneigh(n)
%gives the relative coordinates of all the members of the Moore
%neighbourhood of order n, (0,0) excluded
%each row in a corresponds to a cell
    a = [];
    for k = -n:n
        for l = -n:n
            if (k == 0) && (l == 0)
                continue
            end
```

```
            a = [a;k,l];
        end
    end
end



function calculate_income(i,j)
    %calculates income of the player in cell (i,j) (stored in Income(i,j)
    %=> plays with its 4 neighbours in the Neumann-Neighbourhood of range 1
    global G M N R S T P Income

    %ensure dynamic grid boundaries
    if (i < 1)
        i = i + M;
    end
    if (i > M)
        i = i - M;
    end
    if (j < 1)
        j = j + N;
    end
    if (j > N)
        j = j - N;
    end
    Income(i,j) = 0;
    if G(i,j) == 0
        return
    end
    % Iterate over the von Neumann neighbourhood and bargain
    neighbours = neumannneigh(1);
    for k=1:length(neighbours)
        i2 = i+neighbours(k,1);
        j2 = j+neighbours(k,2);
        %ensure dynamic grid boundaries
        if (i2 < 1)
            i2 = i2 + M;
        end
        if (i2 > M)
            i2 = i2 - M;
        end
        if (j2 < 1)
            j2 = j2 + N;
        end
        if (j2 > N)
            j2 = j2 - N;
        end
        % bargain (four different cases)
        if (G(i,j) == 1) && (G(i2,j2) == 1)
            Income(i,j)=Income(i,j)+R;
        end
        if (G(i,j) == 1) && (G(i2,j2) == 2)
            Income(i,j)=Income(i,j)+S;
        end
        if (G(i,j) == 2) && (G(i2,j2) == 1)
            Income(i,j)=Income(i,j)+T;
        end
        if (G(i,j) == 2) && (G(i2,j2) == 2)
            Income(i,j)=Income(i,j)+P;
        end
    end
```

```
end
```

```
function calculate_neighbourhood_income(i,j)
    %calculates the neighbourhood-income of the cell (i,j)
    %(stored in NeighIncome(i,j)), i.e. how much the players in the
    %Moore-Neighbourhood of range 1 of that cell earn

    global M N Income NeighIncome
    %ensure dynamic grid boundaries
    if (i < 1)
        i = i + M;
    end
    if (i > M)
        i = i - M;
    end
    if (j < 1)
        j = j + N;
    end
    if (j > N)
        j = j - N;
    end
    NeighIncome(i,j) = 0;
    % iterate over the Moore neighbourhood
    neighbours = neumannneigh(1);
    for k=1:length(neighbours)
        i2 = i+neighbours(k,1);
        j2 = j+neighbours(k,2);
        %ensure dynamic grid boundaries
        if (i2 < 1)
            i2 = i2 + M;
        end
        if (i2 > M)
            i2 = i2 - M;
        end
        if (j2 < 1)
            j2 = j2 + N;
        end
        if (j2 > N)
            j2 = j2 - N;
        end
        NeighIncome(i,j) = Income(i2,j2)+NeighIncome(i,j); % summation of all
the neighbourhood incomes
    end
end
```

```
function hypincome = calculate_hypothetical_income(i,j,me)
%calculates hypothetical income (not stored) for an agent
%of type me on (i,j) (used for hypothetical migration strategy)
    global G M N R S T P
    hypincome = 0;
    % Iterate over the Moore neighbourhood and bargain
    neighbours = neumannneigh(1);
    for k=1:length(neighbours)
        i2 = i+neighbours(k,1);
        j2 = j+neighbours(k,2);
        %ensure dynamic grid boundaries
        if (i2 < 1)
            i2 = i2 + M;
        end
        if (i2 > M)
            i2 = i2 - M;
        end
        if (j2 < 1)
            j2 = j2 + N;
        end
        if (j2 > N)
            j2 = j2 - N;
        end
        % bargain (four different cases)
        if (me == 1) && (G(i2,j2) == 1)
            hypincome = hypincome+R;
        end
        if (me == 1) && (G(i2,j2) == 2)
            hypincome = hypincome+S;
        end
        if (me == 2) && (G(i2,j2) == 1)
            hypincome = hypincome+T;
        end
        if (me == 2) && (G(i2,j2) == 2)
            hypincome = hypincome+P;
        end
    end
end
```