# Spatial cooperation games

**Level 2 module in "Modelling course in population and evolutionary biology"**

**(701-1418-00 SS)**

Module author: Dusan Misevic

Course director: Sebastian Bonhoeffer
Theoretical Biology
Institute of Integrative Biology
ETH Zürich

## 1 Introduction

Cooperation is ubiquitous in biological systems. Cooperative interactions are thought to be the basis of some of the most important evolutionary transitions, such as evolution of chromosomes or the transition from uni- to multicellular organisms. Similarly, some of the most interesting and complex behaviours in animal and human societies require cooperation. A classic example is food sharing in vampire bats (*Desmodus rotundus*), where bats are willing to share food amongst each other without any apparent immediate benefit to the food donor. Game theory has frequently been used to understand how this and other cooperative systems can overcome the fitness costs and persist in the face of cheating and exploitation.

Probably the best-known game theory paradigm is the **prisoner's dilemma (PD)**[a]. The classic metaphor of this game sounds more like a brain teaser and goes as follows: Two criminals committed a crime together and both got caught. The police know there is not enough evidence to convict them of all the crimes they are accused of. So they separate the prisoners and offer them the same two options: they can testify for the prosecution against the other prisoner, or remain silent. If both prisoners remain silent (i.e. they "cooperate" with each other), they will

[a] You can find more information on the history and implications of the game in the script of S. Bonhoeffer's lecture "Ecology and Evolution II: Populations" on the main course web page.

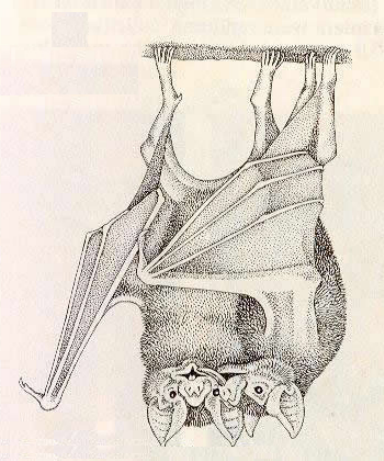


Figure 1: A classic example for cooperation is food sharing in vampire bats.

receive just six months in jail for some minor charges. If they both testify (i.e. they "defect", or "cheat" each other), they will both receive a 2-year jail sentence. However, if one testifies while the other one remains silent, the one who testifies will go free immediately, while the other one will take the blame for all crimes and receive a 5-year jail sentence. Each prisoner has to make his choice independently and without any knowledge of the choice made by the other prisoner. The dilemma poses the question: how should the prisoners act? Should one cheat or cooperate?

The simple, if depressing, answer is that one should always cheat. The reasoning goes as follows: If your fellow inmate cheats, you are better off cheating (2 v. 5 years in jail); if the other prisoner cooperates, you should still cheat (home free v. 6 months in jail). Thus, independently of what the opponent does, you should cheat. The same situation may hold for simple biological interactions: it is generally not a good idea to help an interaction partner that tries to exploit you; and even if the partner tries to help you, you may still be better off by exploiting the other, instead of returning the favour. However, as you will see as you develop the models in this module and as has been repeatedly shown in various biological systems, this very simple conclusion depends strongly on various population and environmental factors and is not always true.

In a general way, prisoner's dilemma can be stated as follows: Each of the players chooses between two options: to cooperate with the other player or to defect. If both cooperate they get a payoff $R$ (reward), if both defect they get a payoff $P$ (punishment). If a player defects but the other player cooperates, the first one gets payoff $T$ (temptation), while the cooperator gets payoff $S$ (sucker's payoff). Depending on the relationships of the four payoff parameters ($R$, $P$, $T$, $S$), different dynamics may emerge. In particular, amongst all games that may be determined by these 4 parameters, the prisoner's dilemma is defined by the relationship: $T > R > P \geq S$.

A variant of the prisoner's dilemma, the **snowdrift game (SD)**, uses a metaphor of two persons stuck in a car in a snowdrift. They can choose to invest energy to dig the car out, the cost of work depending on whether they are both shovelling or one driver has to do all the work. Formally, the snowdrift game is defined with the same type of payoff matrix as the prisoner's dilemma, but with $T > R > S > P$. The crucial difference is that when both players defect in the snowdrift game, they stay stuck in the snow and effectively $P = 0$. So while in prisoner's

dilemma it is better to defect when your opponent defects, in snowdrift game that is not the case – both players defecting means they will remain stuck in the snow, so it is better to cooperate when your opponent defects. As you will see, this seemingly slight change of the parameters may have strong implications on the outcome of situations where the game is played multiple times.

Multiple examples of both PD and SD exist in nature: some fish species play prisoner's dilemma when inspecting a predator and deciding how close to approach it, while yeast cells on a sugar plate play the snowdrift game by deciding whether to produce and (inevitably) share a necessary metabolic enzyme that must be secreted to degrade sugar. In all cooperative games, the question of cheaters necessarily arises: why not let someone else do the work and just reap the benefits? Why not let the other fish inspect the predator and obtain the knowledge without risking the attack yourself? Why not let another yeast cell produce the necessary enzymes for both of us? Why not cheat?

## 2 Developing the model

The strategies that you will explore in this model will be limited to so-called, "pure strategies", where individuals have no choice about their strategy: while playing PD or SD, they either always cooperate or always defect. We will thus refer to them as cooperators and defectors. Interactions between individuals will influence their reproduction – the "score" obtained by playing the prisoner's dilemma will determine the probability of producing offspring with a strategy identical to one's own. By modelling the frequency of cooperators and defectors over time, we will explicitly model the evolution of cooperative behaviour under different conditions. These types of models are referred to as "iterated" because the interaction between organisms involved in a prisoner's dilemma happens not just once, but repeatedly over an extended period of time.

The primary environmental condition of interest here will be the spatial structure of a population. The populations will be modelled in two different ways: with or without space. A frequently used biological example of a spatially unstructured population is a bacterial population living in a liquid, frequently mixed medium (see photo of flasks with *E. coli* on the next page). There, all the pair-wise interactions between individual bacteria are equally likely, and they are all experiencing identical environmental conditions. On the other hand, bacteria growing on the surface of a food source placed in a Petri dish (see photo of a plate with *E. coli* on the next page) typically cannot move, and thus interact only with their immediate neighbours, experiencing different environmental conditions depending on their location. Both of these paradigms apply to a multitude of situations in nature, and sometimes even to the same species. These models will directly investigate the influence of the population structure (spatial v. not-spatial) on the evolution of cooperation.

For simplicity, we can parameterize the payoffs in PD and SD as follows:

$$\begin{aligned} \text{PD: } (R,\ T,\ S,\ P) &= (b-c,\ b,\ -c,\ 0) \\ \text{SD: } (R,\ T,\ S,\ P) &= (b-c/2,\ b,\ b-c,\ 0)\ , \end{aligned}$$

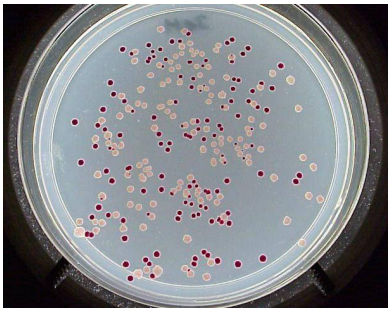



Figure 2: *E. coli* can be grown both in well-mixed and in spatially structured cultures.

where $b$ represents the benefit and $c$ the cost of cooperation, and in all cases $b > c > 0$.

There are several common methods in individual-based modelling for updating the state of the population. Here we recommend that you implement asynchronous updating, where one individual is competed against others and evaluated at a time, and its fate determined (as opposed to synchronous updating, where all individuals are competed and the whole population is updated at the same time). The state of an entire unstructured (non-spatial) population at a given time point can be stored as a two-element vector, consisting of the number of cooperators and defectors in the population. Alternatively, the population can be implemented as a vector with binary, 0 or 1, entries, where 0's signify defectors and 1's stand for cooperators (this implementation makes sampling slightly simpler).

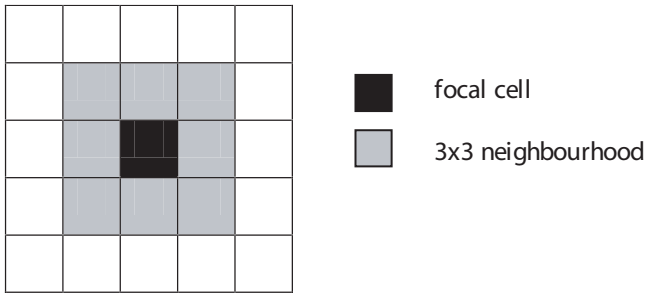
Changes in the population are implemented as follows. At each update, a randomly chosen focal organism $x$ is competed with another, also randomly chosen, organism $y$, and the single-interaction payoff for $x$, denoted $P(x, y)$, is calculated. The same is done for another pair of randomly selected organisms ($u$ and $v$), and the payoff $P(u, v)$ is calculated. The replacement probability $w$ is calculated as $w = (P(u, v) - P(x, y))/\alpha$, where $\alpha$ is a standardization factor equal to maximal payoff difference, i.e. $\alpha = \max(R, T, S, P) - \min(R, T, S, P)$, ensuring $w \leq 1$. Then $x$ is replaced by an offspring of $u$ (inheriting the strategy of $u$) with probability $w$; if $w < 0$, replacement never happens. In essence, the competition between $x$ and $u$ for the site occupied by $x$ is decided not by pitting them directly against each other, but by observing their reward in competitions with "third parties". This scheme may seem complicated at first. Think about possible alternatives and why or why not those alternatives could capture the evolution of cooperation.

In the non-spatial case, all four individuals involved in the competitions above are chosen randomly from the whole population. Such an iterated, non-spatial PD with asynchronous updating has been implemented in the starting script that you can download from the webpage of the module. Interpret the algorithm. How is a "generation" implemented in this asynchronous updating scheme? How is the population initialized? Experiment with changing the parameters and varying the algorithm. Use 5 different $(b, c)$ pairs of your choice, record and plot the proportion of cooperators over time. What do you expect the outcome to be and why? Specifically, run the model for high and low cost-benefit ratios, $\rho = c/(b-c)$. Run the model until it reaches equilibrium, i.e. the frequencies of defectors and cooperators stop changing over time. Interpret

the results: can cooperation in PD spread or be maintained without spatial structure?

## 2.1 Developing a spatial prisoner's dilemma model

Spatial structure can be implemented most easily by a matrix of binary entries, corresponding to a population spread over a two-dimensional surface. The indeces of an element describe the coordinates of an individual on the lattice, and the value determines its type. E.g. if you store your population in the matrix `M`, then `M[i,j]` contains the strategy (type) of the individual at position (i,j). The interacting individuals $y$ and $u$ are chosen from the neighbourhood of $x$, and $v$ is chosen from the neighbourhood of $u$. Neighbourhood of a site in a square grid is typically defined as the 8 sites that are in direct contact with it (for a more detailed explanation see below).




When designing your model, use a "periodic boundary condition" – if an organism were to "fall off the edge" of the grid, it would appear on the other side. Thus, the grid is not really a flat square, but a toroidal surface (think of a doughnut, 3D circle with a hole), and the opposite edges of the lattice are functionally adjacent. In our current simulations, there is no real movement, but interactions can still "spread" over the edge. E.g. the left neighbour of a cell in the leftmost column will be a cell in the rightmost column. To code for the periodic boundary condition you could test for such occurrences and find the right indeces every time an interaction spreads over an edge, but that would be rather cumbersome and time consuming. Instead, in `R` you can change the order of matrix indices and essentially do it automatically. For example, the command `h[c(10,1:9),]` shifts the rows of the matrix `h` down, placing the last row in the first position. Thus when you look for the "left" neighbour of the cell at position (5, 3) in the grid, you can just look for the (5, 3) element in the matrix with the changed indices, such as the one above. The same element shifting "trick" can also be done for columns.

Start your simulations with populations of $N = 2500$ individuals arranged on a $50 \times 50$ square grid, unless otherwise noted. You can also try larger population sizes ($100 \times 100$ or $150 \times 150$), but that increases running time. Run the simulations for 100 generations. Find out how long it takes to reach equilibrium, and adjust simulation length accordingly. When you are running multiple simulations (e.g. by looping over a parameter to be varied), you can even implement a test to ensure that equilibrium has been reached by the end of each simulation. Run simulations for the same sets of parameters that you used for the non-spatial model. Note

the outcomes (percentage of cooperators at the end of the simulation). Are they different than in the non-spatial case? Interpret the results: what is the effect (if any) of spatial structure on the evolution of cooperation?

Finally, explicit modelling of "space" allows you to inspect spatial patterns in the population, in addition to looking at frequencies over time. The state of the population can easily be visualized by the `image()` function of `R`. E.g. `image(M,col=c("blue", "red"))` will plot the entries of `M` using two colours, blue and red, to represent cooperators and defectors on the grid. You may plot the population at each time step or at specified time intervals.

# 3 Exercises

Note: To complete this Level 2 module you should solve all basic exercises and one or more advanced exercise.

## 3.1 Basic exercises

Eb1. Adjust the payoff matrix to implement SD instead of PD in both the spatial and the non-spatial models (you may implement both games within the same scripts by including a "switch" for game choice). Start with a mixed population of cooperators and defectors, and record and plot the proportion of cooperators over time in the population for the same $(b, c)$ pairs as you have done for the prisoner's dilemma. What do you expect the outcome to be and why? Are the results qualitatively different than for PD? Offer an explanation as to why or why not.

Eb2. After a comparison between models, now focus on the effect of payoff parameters within each model. Test each model for at least 20 values of $\rho$ in the $(0, 0.5)$ range (`R`-hint: implement a loop over the vector of parameter values to be tested). Record the proportion of cooperators over time. Does the outcome (percentage of cooperators at the end of the simulation) depend on $\rho$ and if so, how?

Eb3. What is the effect of the initial ratio of defectors and cooperators in the models? Choose a particular value of $\rho$ and run the models for 5 different initial defector/cooperator ratios.

## 3.2 Advanced/additional exercises

Ea1. Implement and perform simulations for different neighbourhood sizes: instead of $3 \times 3$ try $5 \times 5$, $11 \times 11$, $21 \times 21$. What effect do you expect the increase of neighbourhood size will have on the evolution of cooperation?

Ea2. Implement a spatial iterated PD/SD model with synchronous updates. This involves competing all individuals in a "frozen" snapshot of the population and then updating the whole population at once based on the results of those competitions. This can be implemented by looping through all cells of the lattice. However, running time can be

reduced considerably by using R's efficient matrix operations wherever possible. You may try to obtain interacting pairs in vectors or matrices and then you can compute the pay-off for all interactions with a single (and efficient) command. First, notice that the pay-off for an individual playing strategy $x$ in an interaction with an individual playing strategy $y$ (remember: strategies are coded as 0 for defectors and 1 for cooperators) can be calculated for all cases as $P(1-x)(1-y) + T(1-x)y + Sx(1-y) + Rxy$. Second, by "scalar multiplication" of vectors or matrices you can extend this to calculating all the scores in the population simultaneously. In R, given two matrices `M` and `N`, `M*N` produces a new matrix, `Q`, where `Q[i,j] = M[i,j] * N[i,j]`. Note that this is different from the matrix multiplication typically used in algebra. Implement synchronous updating and run the model for the same sets of parameters as the asynchronous variant. Are the outcomes (percentage of cooperators at the end of the simulation) different, and if so, how?

Ea3. Implement a different update rule in the spatial simulations: let individuals compete with all individuals in their neighbourhood (instead of just one), and give the focal site to the individual with the highest payoff in each neighbourhood. You can do this in a model with either synchronous or asynchronous updating. Run the model for the same sets of parameters as before. Are the outcomes (percentage of cooperators at the end of the simulation) different than in the non-spatial case or the basic implementation of the spatial model, and if so, how? R-hint(s): Calculating the maximum score in a neighbourhood can be done in many ways, but most obviously by going though the entire population and calculating the maximum score in the neighbourhood of each individual. For a potential improvement of the program running time, one could also construct a 3-dimensional array with `all_scores <- array(0, dim=c(n, n, num_neighbours))` to hold all the scores in each neighbourhood, and then obtain the location of the neighbour with the highest score using `apply(all_scores, 1:2, which.max)`. For more information, look up R's `help()` on `array()`, `apply()` and `which.max()`. This would be especially useful if the synchronous updating scheme is used above.

Ea4. So far, you have modelled "evolution" by starting with a mixed population and observing which strategy wins in the long run (or whether they can coexist). Now implement mutation, i.e. allow some probability for the offspring to mutate from the strategy of its parental type to the other type. This allows you to start simulations with a homogeneous population composed of a single type, and observe whether rare mutants of the other type can invade the population.

Ea5. Modify the graphing function to indicate the history of each site in spatial simulations, e.g. red/blue for cooperators/defectors that have occupied each site for 2 or more generations, yellow when a cooperator has been replaced by a defector in the current generation, and green when a defector has been replaced by a cooperator in the current generation.

Ea6. Implement a heterogeneous environment with cells or larger patches that cannot be occupied by either type. Does this affect the outcome of the simulations?