The Mathematica® Journal

Simulation of Evolutionary Dynamics in Finite Populations

Bernhard Voelkl

In finite populations, evolutionary dynamics can no longer be described by deterministic differential equations, but require a stochastic formulation [1]. We show how *Mathematica* can be used to both simulate and visualize evolutionary processes in limited populations. The Moran process is introduced as the basic stochastic model of an evolutionary process in finite populations. This model is extended to mixed populations with relative fitness differences. We combine population ecology with game theoretic ideas, simulating evolutionary games in well-mixed and structured populations.

The Moran Process

The Moran process is a simple stochastic model to study selection in finite populations [2]. We consider a population of constant size with two types of individuals, type 1 and type 0. At each time step a single individual is allowed to reproduce a clone of the same type. Furthermore, to keep the population size constant, one individual must die. The Moran process is a birth-death update process. Individuals for reproduction and elimination are chosen randomly. If both random choices fall on the same individual, the individual will be replaced by its own identical offspring and the population remains unchanged. The variable i denotes the number of type 1 individuals in the population of size n. The number of type 0 individuals is therefore $n - i$. The Moran process is defined on the state space $i = 0, \ldots, n$. The probability of choosing a type 1 individual is given by i / n and the probability of choosing a type 0 individual is $(n - i) / n$.

 © 2011 Wolfram Media, Inc.

If a type 0 individual is chosen for reproduction and a type 1 individual for elimination, i decreases by one. If a type 1 individual is chosen for reproduction and a type 0 individual for elimination, i increases by one. In all other cases the populations of types 1 and 0 remain unchanged. The according probabilities for these events are given by:

$$\begin{aligned} p_{i,i-1} &= i(n-i)\,/\,n^2\,, \\ p_{i,i+1} &= i(n-i)\,/\,n^2, \\ p_{i,i} &= 1 - p_{i,i-1} - p_{i,i+1}. \end{aligned} \tag{1}$$

As $p_{0,1} = 0$ and $p_{n,n-1} = 0$, $p_{0,0} = 1$ and $p_{n,n} = 1$. The states $i = 0$ and $i = n$ are therefore absorbing states: when the process has reached such a state it cannot change anymore. Although both types of individuals reproduce at the same rate, one type will always replace the other. Given no time constraints, the coexistence of both types is impossible. The probability that a population with i type 1 individuals will end up in state $i = n$ is given by $x_i = i / n$.

□ Simulation of a Moran Process

To visualize evolutionary dynamics, we represent populations as graphs where each vertex represents an individual. This requires the package *Combinatorica*.

The function `initial[n, i]` constructs a list of length n that represents the initial population with i individuals of type 1 and $n - i$ individuals of type 0.

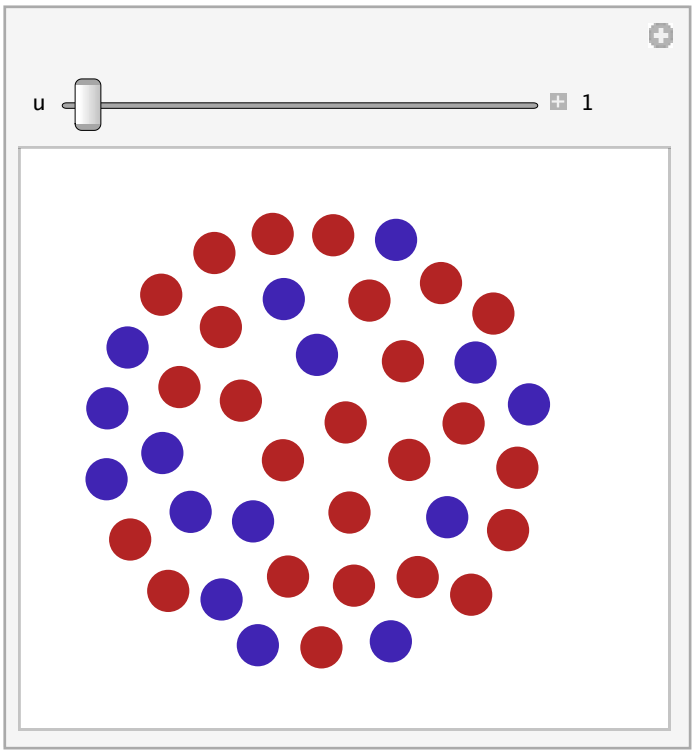
The function `update[pop]` randomly selects two elements `{repro, elim}` of the list `pop` and replaces the value of `elim` (the individual chosen for elimination) by the value of `repro` (the individual chosen for reproduction).

The function `moran[n, i]` starts with an initial condition of i type 1 individuals and $n - i$ type 0 individuals. In each round, one individual is chosen for reproduction and one for elimination. This update process is repeated until the population reaches one of the two absorbing states.

The function `showpop[pop]` plots a graph without edges where `pop` is taken as a list of vertex weights. The `VertexRenderingFunction` is used to color the vertices according to their type as represented by their vertex weights.

Now we simulate the Moran process for a population size of $n = 40$ individuals and $i = 15$ type 1 individuals in the initial condition. The `Manipulate` function lets us see the evolution of the population. The `Manipulate` will work immediately without the need to evaluate it.

```
Manipulate[
 showpop[evolution[[u]]],
 {{u, 1}, 1, Dynamic[Length[evolution]], 1,
  Appearance → "Labeled"},
 AutorunSequencing → {1},
 SynchronousInitialization → False,
 Initialization :→ (
   Quiet@Get["Combinatorica`"];

   initial[n_, i_] := Join[Table[1, {i}], Table[0, {n - i}]];

   update[pop_List] := Module[{repro, elim},
     {repro, elim} = RandomInteger[{1, Length[pop]}, 2];
     ReplacePart[pop, elim → pop[[repro]]]];

   moran[n_, i_] := NestWhileList[update, initial[n, i],
     n > Total[#] > 0 &];

   showpop[pop_] := Module[{g},
     g = Combinatorica`SetVertexWeights[
       Combinatorica`CompleteGraph[Length[pop]], pop];
     GraphPlot[g, EdgeRenderingFunction → None,
      VertexRenderingFunction →
       ({If[Combinatorica`GetVertexWeights[g][[#2]] == 0,
           Hue[0, 0.8, 0.7], Hue[0.7, 0.8, 0.7]],
          PointSize[0.08], Point[#]} &), ImageSize → 280]];

   evolution = moran[40, 15];

   If[u > Length[evolution], u = Length[evolution]]
  )]
```



The Moran Process with Relative Fitness

Now we consider the case when the two types, 1 and 0, have different fitness. Fitness determines the rate at which they reproduce. If we set the fitness of type 0 to 1 and the fitness of type 1 to r, then the probability that type 1 is chosen for reproduction is given by $p_1 = r\,i\,/\,(r\,i + n - i)$ and the probability that type 0 is chosen is given by $p_0 = (n - i)\,/\,(r\,i + n - i)$. The probabilities for being chosen for elimination remain unchanged. The fixation probability for i type 1 individuals is given by:

$$x_i = \frac{1 - 1\,/\,r^i}{1 - 1\,/\,r^n}. \tag{2}$$

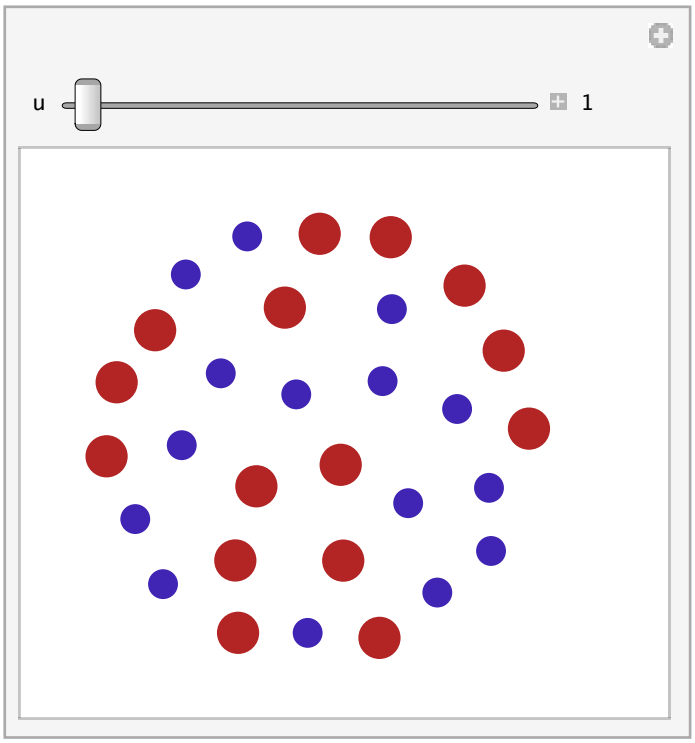
Because the probability of being chosen for reproduction depends now on the continuous variable r, the simulation is modified. In `updateRel[pop, r]` we evaluate the number of type 1 individuals that is equivalent to the total of the list `pop`. Thereafter we evaluate `reprod`, the probability with which a type 1 individual is chosen for reproduction. Finally we randomly select one element of list `pop` and replace it by 1 with probability `reprod` and by 0 with probability `1 - reprod`.

The function `moranRel[n, i, r]` simulates a Moran process in a mixed population of size n with i individuals with relative fitness r and $n - i$ individuals with relative fitness 1.

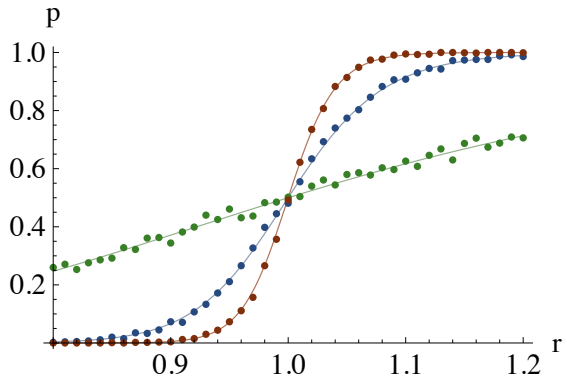
In `showpopRel[pop, r]`, the `VertexRenderingFunction` is used to color the vertices according to their type and to alter the point size of the vertices proportional to the square root of their relative fitness r.

Here we simulate the Moran process for a population size of $n = 30$ and $i = 15$ type 1 individuals with relative fitness 0.5.

```
Manipulate[
 showpopRel[evolution2[[u]], 0.5],
 {{u, 1}, 1, Dynamic[Length[evolution2]], 1,
  Appearance → "Labeled"},
 AutorunSequencing → {1},
 SynchronousInitialization → False,
 Initialization :→ (
   Quiet@Get["Combinatorica`"];

   initial[n_, i_] := Join[Table[1, {i}], Table[0, {n - i}]];

   updateRel[pop_List, r_] :=
    Module[{j, reprod, elimPos},
     j = Total[pop];
     reprod = r j / (r j + Length[pop] - j);
     elimPos = RandomInteger[{1, Length[pop]}];
     ReplacePart[pop,
      elimPos → RandomChoice[{1 - reprod, reprod} → {0, 1}]]];

   moranRel[n_, i_, r_] :=
    NestWhileList[updateRel[#, r] &, initial[n, i],
     n > Total[#] > 0 &];

   showpopRel[pop_, r_] :=
    Module[{g},
     g = Combinatorica`SetVertexWeights[
       Combinatorica`CompleteGraph[Length[pop]], pop];
     GraphPlot[g, EdgeRenderingFunction → None,
      VertexRenderingFunction →
       (Flatten[
          {If[Combinatorica`GetVertexWeights[g][[#2]] == 0,
            {Hue[0, 0.8, 0.7], PointSize[0.08]},
            {Hue[0.7, 0.8, 0.7], PointSize[0.08 Sqrt[r]]}],
           Point[#]}] &), ImageSize → 280]];

   evolution2 = moranRel[30, 15, 0.5];

   If[u > Length[evolution2], u = Length[evolution2]]
  )]
```



Simulating the evolution of populations of different sizes and with different values for the relative fitness, r, we can see how both parameters can influence selection strength. Figure 1 shows that for a population size of $n = 100$ individuals, even small fitness differences have a strong effect on the fixation probability of the respective types, while for a small population of $n = 10$ individuals, fitness differences must be much larger to lead to the same fixation probabilities.



▲ **Figure 1**. Fixation probabilities, p, for a type of relative fitness, r, with an initial abundance of 50% in populations of $n = 10$ (green), 50 (blue), and 100 (red) individuals. Points indicate estimates based on running `moranRel` 1000 times for r-values of 0.8 to 1.2 in increments of 0.02. Solid lines show expected fixation probabilities based on equation (2).

Evolutionary Games

We consider a population with two types of individuals, where individuals interact with each other at regular rates. Whenever two individuals interact with each other, they receive payoffs from this interaction according to the payoff matrix:

$$\begin{array}{c} \\ 1 \\ 0 \end{array} \begin{array}{c} \begin{array}{cc} 1 & 0 \end{array} \\ \begin{pmatrix} a & b \\ c & d \end{pmatrix} \end{array} \tag{3}$$

As the probability that two individuals of specified type interact depends on the relative frequencies of the types given by i for type 1 and $n - i$ for type 0, the expected payoffs for type 1 and type 0 are:

$$\begin{aligned} F_{(1)\,i} &= \frac{a\,(i-1) + b\,(n-1)}{n-1}, \\ F_{(0)\,i} &= \frac{c\,i + d\,(n-i-1)}{n-1}. \end{aligned} \tag{4}$$

Payoffs contribute to the fitness of the individuals by

$$\begin{aligned} f_{(1)\,i} &= 1 - w + w\,F_{(1)\,i}, \\ f_{(0)\,i} &= 1 - w + w\,F_{(0)\,i}, \end{aligned} \tag{5}$$

where $r_i = f_{(1)\,i} / f_{(0)\,i}$. The parameter w is a measure for the strength of selection. If $w = 1$, fitness is completely determined by the payoff; if $w = 0$, fitness is independent of the payoff. When studying evolutionary processes, biologists usually assume that lifetime fitness of an individual is determined by many variables, thus any single gene will only have a weak effect on selection. A selective strength w between 0.01 and 0.05 is often suggested to study evolution under weak selection.

To start the simulation we choose a population size n and define `population` as an empty list. The number of type 1 individuals, i, is chosen randomly from the interval $[1, n - 1]$. We create a list `strategies` of length n and set i randomly chosen elements to 1 while the remaining elements are 0. (The randomization of the positions is not necessary right now, as it will not influence the outcome, but it will become important later on.) This list represents our population and is equivalent to the list `pop` in the previous section.

For our model of selection we introduce a death-birth update process. This means that in every round a randomly chosen individual will be eliminated and the vacated space will be taken over by a new individual. The probability that this new individual will be of type 1 is proportional to the overall fitness of the type 1 individuals in the neighborhood of `elim`. Fitness is evaluated after elimination, thus the effective population size n^e equals $n - 1$.

The function `fitness[strategies, elim, n]` takes the list `strategies`, deletes the entry at position `elim`, and evaluates the relative fitness of type 1 according to equations (4) and (5). In this case the payoffs $a - d$ are chosen so that the game represents a prisoner's dilemma and we set the selection strength w to 0.05.

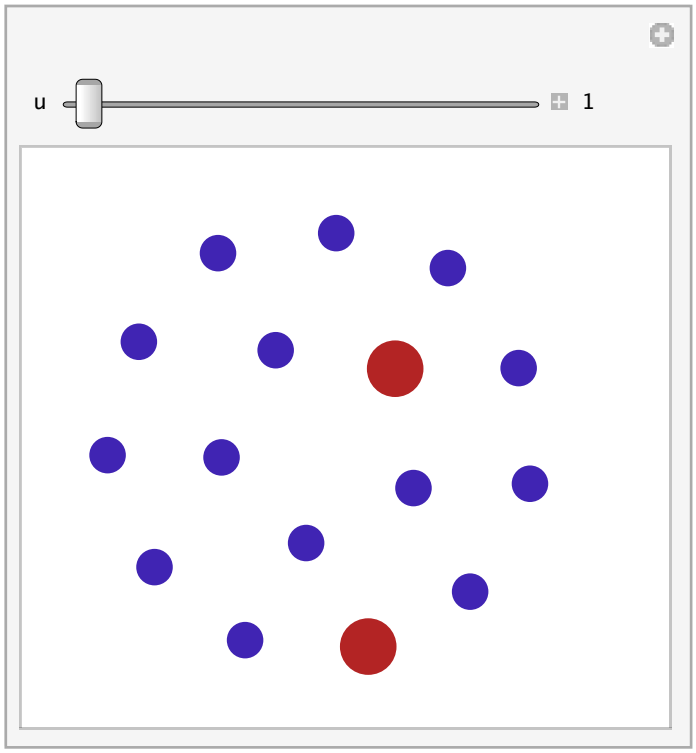
As in the previous section, we repeat the update process with `updateGame` until one of the two absorbing states is reached. In each round we randomly choose one individual for elimination. Thereafter we request a random number. If this random number is below the limit evaluated by the function `fitness`, the eliminated individual is replaced by a type 1 individual, otherwise by a type 0 individual.

In `showGame[r, strategies]`, `VertexRenderingFunction` is used to color the vertices according to their type as indicated by the vertex weight list `strategies` and to alter the point size of the vertices in proportion to their relative fitness, where i is the number of type 1 individuals in `strategies` and r / i is the relative fitness for type 1.

The `Manipulate` function lets us see the evolution of the population.

```
Manipulate[
 showGame @@ population[[u]],
 {{u, 1}, 1, Dynamic[Length[population]], 1,
  Appearance → "Labeled"},
 AutorunSequencing → {1},
 SynchronousInitialization → False,
 Initialization :> (
   Quiet@Get["Combinatorica`"];

   updateGame[pop_, n_] :=
    Module[{elim}, elim = RandomInteger[{1, n}];
     If[Random[] ≤ fitness[pop[[2]], elim, n],
      ReplacePart[pop, {2, elim} → 1],
      ReplacePart[pop, {2, elim} → 0]]
    ];

   fitness[strategies_, elim_, n_] :=
    Module[{i, f1, f0, a, b, c, d, w},
     i = Total[Delete[strategies, elim]];
     f1 = 1 - w + w (a i (i - 1) + b i (n - 1 - i)) / (n - 1);
     f0 =
      1 - w + w (c i (n - 1 - i) + d (n - 1 - i) (n - 2 - i)) / (n - 1);
     f1 / (f1 + f0) //. {a → 5, b → 0, c → 3, d → 1, w → 0.05}
    ];

   population = Module[{n = 16, strategies, r},
     strategies = ReplacePart[Table[0, {n}],
       Partition[RandomSample[Range[n],
           RandomInteger[{1, n - 1}]], 1] → 1];
     r = fitness[strategies, RandomInteger[{1, n}], n];
     NestWhileList[updateGame[#, n] &, {r, strategies},
      Total[#[[2]]] > 0 && Total[#[[2]]] < n &]
    ];
```

```
showGame[r_, strategies_] := Module[{gr, n},
   n = Length[strategies];
   gr = Combinatorica`CompleteGraph[n];
   GraphPlot[gr, EdgeRenderingFunction → None,
    VertexRenderingFunction →
     (Join[If[strategies[[#2]] == 0,
         {Hue[0, 0.8, 0.7],
          PointSize[
           0.3 Sqrt[(1 - r) / (n - Total[strategies])]]},
         {Hue[0.7, 0.8, 0.7],
          PointSize[0.3 Sqrt[r / Total[strategies]]]}],
       {Point[#]}] &), ImageSize → 280]
  ];

If[u > Length[population], u = Length[population]]
)]
```



Evolutionary Games in Structured Populations

So far we have considered the case of well-mixed populations. That means that each individual interacts with all other individuals equally often. While this is a convenient assumption for modeling, anthropologists studying human organizations and biologists studying animal social behavior have repeatedly emphasized the fact that human and animal societies are rarely well-mixed, but structured; that is, individuals usually interact only with a small subset—a neighborhood—of the population [3, 4]. We can incorporate information about the structure of a population by assigning weights to the edges of the graph, where the edge weight is proportional to the probability that these two individuals interact with each other [5].

We study the evolution of cooperation in a small, heterogeneous population. Individuals can be either of type `COOP` (cooperator) or `DEF` (defector). When two cooperating individuals interact, each individual gains a benefit b from the mutual cooperative act, but also has to pay some costs c. If a cooperator interacts with a defector, the cooperator has to pay the costs c, but the defector gets the benefit b without paying any costs. If two defectors meet they get nothing, but they also have no costs. This leads to the payoff matrix

$$\begin{array}{c} \\ Coop \\ Def \end{array} \begin{array}{c} \begin{array}{cc} Coop & Def \end{array} \\ \begin{pmatrix} b-c & -c \\ b & 0 \end{pmatrix} \end{array} \tag{6}$$

Simulation of Games in Structured Populations

As an example we take the `sociomatrix` of a group of nine chimpanzees (*Pan troglodytes*) [6]. Entries in `am` represent frequencies of directed grooming actions within dyads of apes.

As in the previous section, we assume a death-birth update process. In each round a randomly chosen individual is eliminated, but now only the neighborhood of this individual—that is, those individuals that interacted with the eliminated individual—compete for the vacated space [7]. The likelihood that a vacated space is filled with a type `COOP` individual is proportional to the fitness of its `COOP` neighbors and their interaction strength with the vacated space. The fitness of the neighbors is derived from the payoffs these individuals gain from interactions with their neighbors.

The function `probcoop` evaluates the probability with which the eliminated individual at position `elim` will be replaced by an individual of type COOP. First we calculate `benefits`, the benefits that individuals in the neighborhood of `elim` receive from the interactions with their neighbors. Thereafter we calculate `costs`, the costs that individuals in the neighborhood of `elim` have to pay. The list `nhstrat` gives the strategies of the neighbors of `elim`, where 1 stands for a type COOP individual and 0 for a type DEF individual; `benefitsCOOP` sums up the benefits of only those individuals in the neighborhood of `elim` that are of type COOP, while `benefitsDEF` is the sum of benefits for the `DEF` individuals in the neighborhood of `elim`. Equivalently we calculate the costs for

`COOP` individuals in the neighborhood of `elim` as `costsCOOP`. Type DEF individuals have no costs. The variables `nCOOP` and `nDEF` give the numbers of cooperators and defectors in the neighborhood of `elim`. The relative fitness of `COOP` is evaluated according to equations (4) and (5). By setting c to 1, we need only one parameter value: the b/c ratio, which characterizes the payoff matrix. As in the previous example, we assume weak selection and set the selection strength w to 0.05.

To visualize evolutionary dynamics in the population, we introduce the function `allfit`, which gives the relative fitness for all individuals in the population. This function is not necessary to simulate the evolutionary process, because in each round all that is needed is the local information of the neighborhood around the eliminated individual. Especially for large and weakly connected populations, `allfit` performs a lot of superfluous computations. We include `allfit` only for visualization purposes, but not when repeating the simulation many times to estimate fixation probabilities.

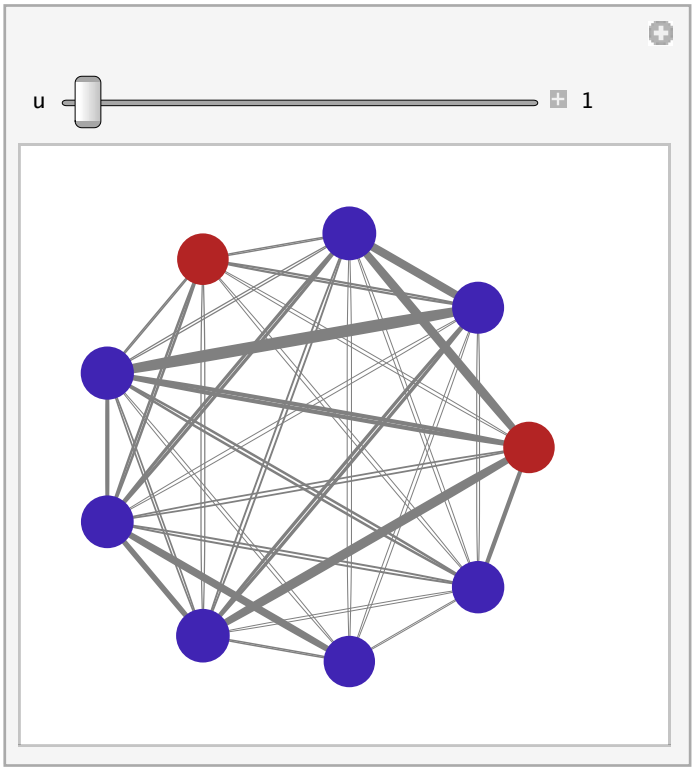
Here we simulate the evolution of a population with a population structure given by the adjacency matrix `sociomatrix`. Entries in the adjacency matrix are frequencies of dyadic interactions per round; `neigborhood` is a list of length n, where the i^{th} element is a list giving the neighborhood of individual i. The initial number of type COOP individuals, `initialcooperators`, is chosen randomly from the interval $[1, n-1]$ and values of 1 for type COOP and 0 for type DEF are randomly allocated to the population list `strategies`. Strategies are updated until the population reaches an absorbing state, as described in the previous section. At each round we attach a list with two elements to `structuredPopulation`, containing a list with the square root of the relative fitness for all individuals and a list with vertex weights, denoting the individuals' type.

In `showGame2` we use `"CircularEmbedding"` as the plot method, because this is the most common visualization in the social sciences for groups of small size. The `EdgeRenderingFunction` is used to visualize the strength of connections between the individuals: the thickness of the line is proportional to the edge weight in the adjacency matrix `am`.

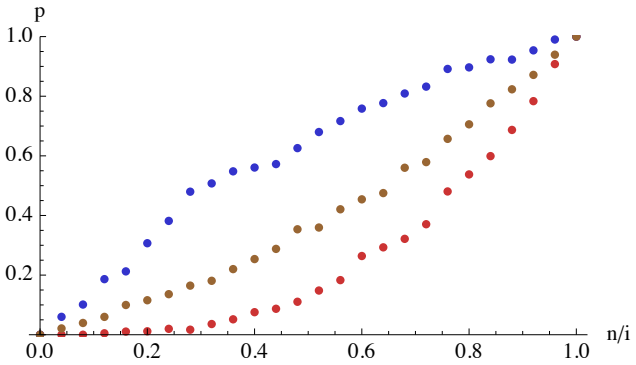
Here is a visualization of the evolution of a mixed, heterogeneous population of cooperators and defectors. Defectors are indicated by red vertices; cooperators are colored blue. Interaction frequencies are indicated by the thickness of the connecting lines.

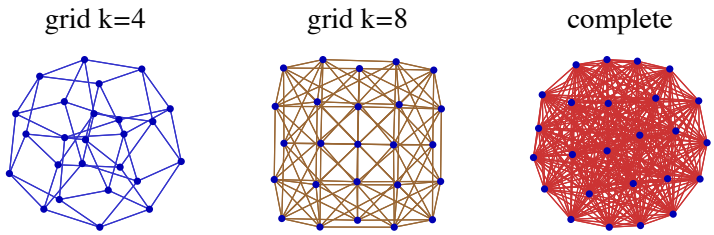
```
Manipulate[
 showGame2 @@ structuredPopulation[[u]],
 {{u, 1}, 1, Dynamic[Length[structuredPopulation]],
  1, Appearance → "Labeled"},
 AutorunSequencing → {1},
 SynchronousInitialization → False,
 Initialization :→ (
   Quiet@Get["Combinatorica`"];

   probcoop[elim_, strategies_, neighborhood_,
     sociomatrix_, n_] :=
    Module[{benefits, costs, nhstrat, benefitsCOOP,
      benefitsDEF, costsCOOP, nCOOP, nDEF, fCOOP, fDEF,
      c, b, w},
     benefits = Apply[Plus,
       Table[strategies, {Length[neighborhood[[elim]]]}]
        Transpose[sociomatrix][[neighborhood[[elim]]] , 2];
     costs = Apply[Plus, sociomatrix[[neighborhood[[elim]]],
       2];
     nhstrat = strategies[[neighborhood[[elim]]];
     benefitsCOOP = Total[benefits nhstrat];
     benefitsDEF = Total[benefits] - benefitsCOOP;
     costsCOOP = Total[costs nhstrat];
     nCOOP = Total[nhstrat];
     nDEF = Length[nhstrat] - nCOOP;
     fCOOP =  nCOOP (1 - w) + w (benefitsCOOP b - costsCOOP c);
     fDEF =  nDEF (1 - w) + w benefitsDEF b;
     fCOOP / (fCOOP + fDEF) /. {c → 1, b → 10, w -> 0.05}
    ];

   allfit[sociomatrix_, strategies_, n_] := Module[{w, b},
     (1 - w) +
       w
        (b Total[sociomatrix Transpose[Table[strategies,
                 {n}]]] - Total[Transpose[sociomatrix]]) /
         Total[Flatten[sociomatrix]] //. {w → 0.05, b → 40}
    ];

   updateGame2[pop_List] :=
    Module[{elim, strategies = pop[[2]], neighborhood = pop[[3]],
      sociomatrix = pop[[4]], n = pop[[5]]},
     elim = RandomInteger[{1, n}];
     If[Random[] ≤ probcoop[elim, strategies,
        neighborhood, sociomatrix, n],
      ReplacePart[pop, {2, elim} → 1],
      ReplacePart[pop, {2, elim} → 0]]
    ];
```

```
createPopulation[] :=
  Module[{sociomatrix, neighborhood, initialcooperators,
    strategies, n},

   sociomatrix = {{0, 0, 238, 22, 160, 68, 219, 0, 109},
     {0, 0, 265, 85, 317, 26, 129, 10, 43},
     {238, 265, 0, 54, 49, 125, 73, 40, 33},
     {22, 85, 54, 0, 59, 97, 48, 0, 8},
     {160, 317, 49, 59, 0, 105, 65, 12, 97},
     {68, 26, 125, 97, 105, 0, 157, 183, 76},
     {219, 129, 73, 48, 65, 157, 0, 56, 5},
     {0, 10, 40, 0, 12, 183, 56, 0, 15},
     {109, 43, 33, 8, 97, 76, 5, 15, 0}};
   n = Length[sociomatrix];
   neighborhood =
    Table[Flatten[Position[sociomatrix[[i]], x_?(# ≠ 0 &)]],
     {i, n}];
   initialcooperators = RandomInteger[{1, n - 1}];
   strategies = ReplacePart[Table[0, {n}],
     Partition[RandomSample[Range[n],
        initialcooperators], 1] → 1];

   NestWhileList[updateGame2,
    {Sqrt[allfit[sociomatrix, strategies, n]],
     strategies, neighborhood, sociomatrix, n},
    Total[#[[2]]] > 0 && Total[#[[2]]] < n &]
  ];

showGame2[fitnesslist_, strategylist_, neighborhood_,
   sociomatrix_, n_] := Module[{gr},
   gr = Combinatorica`FromAdjacencyMatrix[
     sociomatrix / Plus @@ Flatten[sociomatrix] //. 0 → ∞,
     Combinatorica`EdgeWeight, Type → Directed];
   gr = Combinatorica`SetVertexWeights[gr, fitnesslist];
   GraphPlot[gr, Method -> "CircularEmbedding",
    MultiedgeStyle → 0.01, DirectedEdges → True,
    EdgeRenderingFunction →
     ({GrayLevel[0.5],
        Thickness[0.3 Combinatorica`GetEdgeWeights[gr][[
           Position[gr, #2][[1, 2]]]], Line[#]} &),
    VertexRenderingFunction →
     ({If[strategylist[[#3]] == 1, Hue[0, 0.8, 0.7],
         Hue[0.7, 0.8, 0.7]],
        PointSize[0.1 Combinatorica`GetVertexWeights[gr][[
           #3]]], Point[#]} &), ImageSize → 240]
  ];

structuredPopulation = createPopulation[];
If[u > Length[structuredPopulation],
 u = Length[structuredPopulation]]
)]
```



In sparse graphs the fixation probability for cooperation is higher than in fully connected graphs. Ohtsuki and collaborators [7] have demonstrated that there exists a simple rule for the emergence of cooperation on regular or scale-free graphs: a single cooperator has a higher fixation probability than expected for a neutral type (with $r = 1$) if the benefit/cost ratio is larger than the average vertex degree, $b / c > k$.





▲ **Figure 2**. (a) Fixation probability p of cooperation in relation to the proportion of cooperators in the initial condition n/i for 25 individuals in a grid graph with $k = 4$ (blue), a grid graph with $k = 8$ (brown), and a complete graph (red).

Conclusion

Cooperation has been described at all levels of biological organization, from molecules and cells to groups and populations, and in taxa as diverse as the myxomycota, arthropods, and vertebrates. But, already when outlining his concept of evolution by means of natural selection, Charles Darwin identified cooperative behavior as a special difficulty, potentially fatal to his whole theory [8]. Cooperation means that one individual experiences costs (by spending time and energy or accepting additional risks) for a mutual benefit. When two individuals cooperate, their mutual benefits from the cooperative interaction might be higher than their shared costs. However, cooperators are always vulnerable to being exploited by selfish partners who take the benefits but refuse to share the costs. Much theoretical and empirical work on cooperation has focused on identifying conditions under which cooperation can be evolutionarily stable against exploitation. So far five different mechanisms have been proposed as candidate facilitators for cooperation: group selection, kin selection, direct reciprocity, indirect reciprocity, and network reciprocity [9]. All these theoretical propositions seem able to solve one of the biggest enigmas of modern evolutionary biology, but which of these mechanisms works is still unresolved, given the parameter values and noise of real world populations. Attempts to answer this question will require both formal modeling and numeric simulations incorporating empirical data.

References

[1] M. A. Nowak, *Evolutionary Dynamics: Exploring the Equations of Life*, Cambridge MA: The Belknap Press of Harvard University Press, 2006.

[2] P. A. P. Moran, *The Statistical Processes of Evolutionary Theory*, Oxford: Clarendon Press, 1962.

[3] F. C. Santos, J. M. Pacheco, and T. Leanerts, "Evolutionary Dynamics of Social Dilemmas in Structured Heterogeneous Populations," *Proceedings of the National Academy of Sciences* (USA), **103**(9), 2006 pp. 3490–3494. doi:10.1073/pnas.0508201103.

[4] R. Axelrod, *The Evolution of Cooperation*, New York: Basic Books, 1984.

[5] B. Voelkl and R. Noë, "The Influence of Social Structure on the Propagation of Social Information in Artificial Primate Groups: A Graph-Based Simulation Approach," *Journal of Theoretical Biology*, **252**(1), 2008 pp. 77–86. doi:10.1016/j.jtbi.2008.02.002.

[6] T. Nishida and K. Hosaka, "Coalition Strategies among Adult Male Chimpanzees of the Mahale Mountains, Tanzania," in *Great Ape Societies* (W. C. McGrew, L. F. Marchant, and T. Nishida, eds.), Cambridge: Cambridge University Press, 1996 pp. 114-134.

[7] H. Ohtsuki, C. Hauert, E. Lieberman, and M. A. Nowak, "A Simple Rule for the Evolution of Cooperation on Graphs and Social Networks," *Nature*, **441**, 2006 pp. 502–505. doi:10.1038/nature04605.

[8] C. Darwin, *On the Origin of Species by Means of Natural Selection*, London: J. Murray, 1859.

[9] M. A. Nowak, "Five Rules for the Evolution of Cooperation," *Science*, **314**, 2006 pp. 1560–1563. doi:10.1126/science.1133755.

B. Voelkl, "Simulation of Evolutionary Dynamics in Finite Populations," *The Mathematica Journal*, 2011. dx.doi.org/doi:10.3888/tmj.13–8.

Acknowledgments

I want to thank one anonymous reviewer for improving the program. This research has received funding from the European Community's Sixth Framework Programme (FP6/2002-2006) under contract n. 28696.

About the Author

Bernhard Voelkl started his career at the Centre National pour la Recherche Scientifique (CNRS) in Strasbourg at the Département Ecologie, Physiologie et Ethologie and is now a research fellow at the Center for Integrative Life Sciences in Berlin. As part of the research initiative GEBACO, "*Towards the Genetic Basis of Cooperation*," he investigated the preconditions necessary for the evolution of cooperative behavior in animals. He uses *Mathematica* for formal modeling, simulation studies, and statistical data analysis.

Bernhard Voelkl
Center for Integrative Life Sciences and
Institute for Theoretical Biology, Humboldt University
Invalidenstrasse 43, 10115 Berlin, Germany
bernhard.voelkl@c-strasbourg.fr