NetLogo simulation of cooperation in space

Jeremy Van Cleve

In this modeling session, we will use NetLogo to study how cooperation can evolve when individuals live on a grid and copy successful strategies from their neighbors. We won't model individuals as turtles here; rather we'll just use patches and assume that individuals are fixed in space. Each individual (patch) will be either a cooperation or noncooperator (defector). When updating their strategy (cooperate or not) in the next generation or time step, each individual will look to its neighbors and copy a strategy based on how successful it is in its neighborhood

This model is a bit more complex and will take a while to setup. Let's see how far we get and hopefully we get to some interesting patterns relating the size of the neighborhood and the cost and benefit of cooperation to whether cooperation can evolve. This connects directly to Hamilton's rule.

The beginning

1. Open the file titled "Altruism - basic.nlogo". This contains a skeleton of sorts with couple of things (though not many!) already in place. In the interface, you'll see that it has the setup and go buttons and a plot for tracking the number of cooperators and the number of selfish types (noncooperators). In the code, you'll see that there is a patches-own section, which we'll need to add to, the initialize and go procedures, and couple extra procedures that say "Ignore". We'll use those functions later once we get the whole simulation working.

2. Before writing any code, we should map out what we'd like the simulation to do. Specifically, we should have a very detailed plan for how individuals in this population behave and how they reproduce. Here is a sketch of what I have in mind (we will discuss and add to this if necessary):

   (a) The population will have two types of things, patches with a cooperator and patches with a selfish individual.

   (b) Individuals who cooperate generate a benefit that is split among their neighbors and themselves equally.

   (c) Cooperators pay a cost for cooperating.

   (d) Selfish individuals obtain the benefit of cooperation from the neighbors who are cooperators.

   (e) During reproduction, each individual dies and a new individual fills their patch. The new individual will be a clone of one of the dying individual's neighbors or the dying individual itself. You could also think of this as each individual having an opportunity

to change its strategy each time step and choosing a strategy based on what its neighbors are doing.

Coding

1. Now that we know the lifecycle that we'd like to model, we can start figuring out the variables and procedures we'll need. Its good to think about these a bit first before writing any code so that we don't have to go back and make huge changes later if we discover we didn't design things correctly.
   (a) What kinds of global variables do we need?
   (b) What kinds of variables do the patches (individuals) need?
   (c) What procedures do we need? That is, what "tasks" need to be done that we can put in defined blocks of code.
2. Now that we have a map of what we'll need, let's start writing some code! We can work on this in tandem and I'll contribute guidance to help everyone stay together.

Experimenting

1. Ok, now that we have the basic simulation up and running (whew!), lets start playing around a bit.
   (a) Start with half cooperators in the population and a cost of 0.1 and a benefit of 0.2. What happens? When you increase the benefit, does the behavior change? If so, at what value of the benefit does this change occur?
   (b) Do the same thing now by varying the cost of cooperation holding the benefit at 0.2.
   (c) You'll probably notice by now that the population can drift around a bit. If you could automate running the simulation over and over again, what would do it so that you could tell on average what happens for a specific set of parameter values? You don't have to implement your idea in code; just brainstorm and talk to me if you're curious. Hint: there is something you can do right away that's pretty **quick** and dirty and doesn't involve new code!
2. If we've had a chance to get this far, then we can now add an important feature, which uses those "Ignore" functions in the code. Right now, we're assuming that the neighborhood of each individual is the von Neumann neighborhood. We can make this much more general where the neighborhood size can be set at the beginning of the run and we can use the Moore neighborhood instead too. Let's give this crack; it shouldn't be too hard if we designed the simulation nicely in the beginning (I'll try and make sure we do so).
3. Now, we can check how the neighborhood size affects our previous results. Using the same initial parameter values, now increase the neighborhood size (either by increasing the radius or change first to a Moore neighborhood). What happens? Can you get cooperation to evolve stil?

4. Think back to our discussion of Hamilton's rule: $benefit \times relatedness - cost > 0$. What does increasing the neighborhood size do to relatedness?

5. Try changing the initial frequency of cooperators to a different value. Do you get different results from before given the same benefit and cost values? What does this say about how cooperation might have evolve initially from noncooperation (selfishness)?