In *The Evolution of Cooperation*, Axelrod built a model of cooperation that required each agent to remember its past interactions with every other agent. In *The Evolution of Ethnocentric Behavior*, Axelrod and Hammond build a model of cooperation that requires much less knowledge and thought on behalf of the agent. It relies on spatial locality—both in groups and interactions—to create clusters of agents with similar strategies. In this model, there are two ways agents might end up next to each other: either by both being the product of reproduction within a single population, or by two disparate populations growing until they brush up against each other. In the former case, it is highly likely if mutation rate is small that two nearby agents will share the same strategy. In the latter case, no such assumptions can be made. Since agents cannot directly observe each other’s strategies, they need another way to determine their degree of relatedness, and so the observable tag of color serves as a proxy for the likelihood of cooperation—nearby agents with the same color are far more likely to be close relatives that share the same strategy than agents with different colors.

Since this serves as a proxy for how closely related agents are, one logical extension to this would be to extend this from a single tag to multiple tags. Abstractly, we can see the tag as an *n*-dimensional vector that for each mutation moves some small distance in a random direction on each mutation. This is a random walk which we would expect to cause two initially closely related agents to gradually move further apart over time.

One simple way to implement this is to create a tag string consisting of a variable number *n* of bits, each of which is initially set randomly and which, when an agent reproduces, is copied to each offspring with each bit having some small random chance to be flipped. We can then measure the degree of similarity between two agents as the Hamming distance between their two tag strings. If we are considering this as a sociological model, we might think of each bit as representing some observable characteristic of an individual. One bit might represent skin color, another nose shape, religion, etc. One assumption we are making is that the value of each bit is something the agent is incapable of changing (we may revisit this later) and that they are stuck with what they are born with. It is possible that in deciding whether or not to cooperate, an agent may place more weight on differences in some bits than others, or that some bits may have higher rates of mutation than others, however as a first assessment, we are going to assume that each bit is weighted equally and has an equal chance of mutation. We may revisit implications of other models later.

*The Evolution of Ethnocentric Behavior* implements four strategies: CC (cooperator), CD (ethnocentric), DC (cosmopolitan) and DD (cheater). To allow us to extend the model from its original behavior, we implement these strategies in terms of two variables: different-threshold, the value of Hamming distance with another agent which, if met or exceeded, will alter an agent’s decision to cooperate or not, and is-cosmopolitan?. If set to false, an agent will cooperate with agents whose Hamming distance is < different-threshold and defect with agents whose Hamming distance is ≥ different-threshold. If set to true, these behaviors are inverted so an agent defects with agents Hamming distance is < different-threshold and cooperates otherwise. To allow for “color-blind” cases, we allow cooperation-threshold to go a little way outside of the range of possible values for Hamming distance, so for example in the 1-tag case, we set the range of possible values for cooperation-threshold to [0,2].

We can reproduce the strategies of the two-color Axelrod and Hammond model by setting *n* = 1 and setting thresholds for each strategy as follows:

* CC: different-threshold = 2, is-cosmopolitan? = false, or different-threshold = 0, is-cosmopolitan? = true
* CD: different-threshold = 1, is-cosmopolitan? = false
* DC: different-threshold = 1, is-cosmopolitan? = true
* DD: different-threshold = 2, is-cosmopolitan? = true, or different-threshold = 0, is-cosmopolitan? = false

We can also reproduce the single-color case given by Axelrod and Hammond simply by setting *n* = 1. Reproducing their mutation system with these parameters is slightly more complicated.

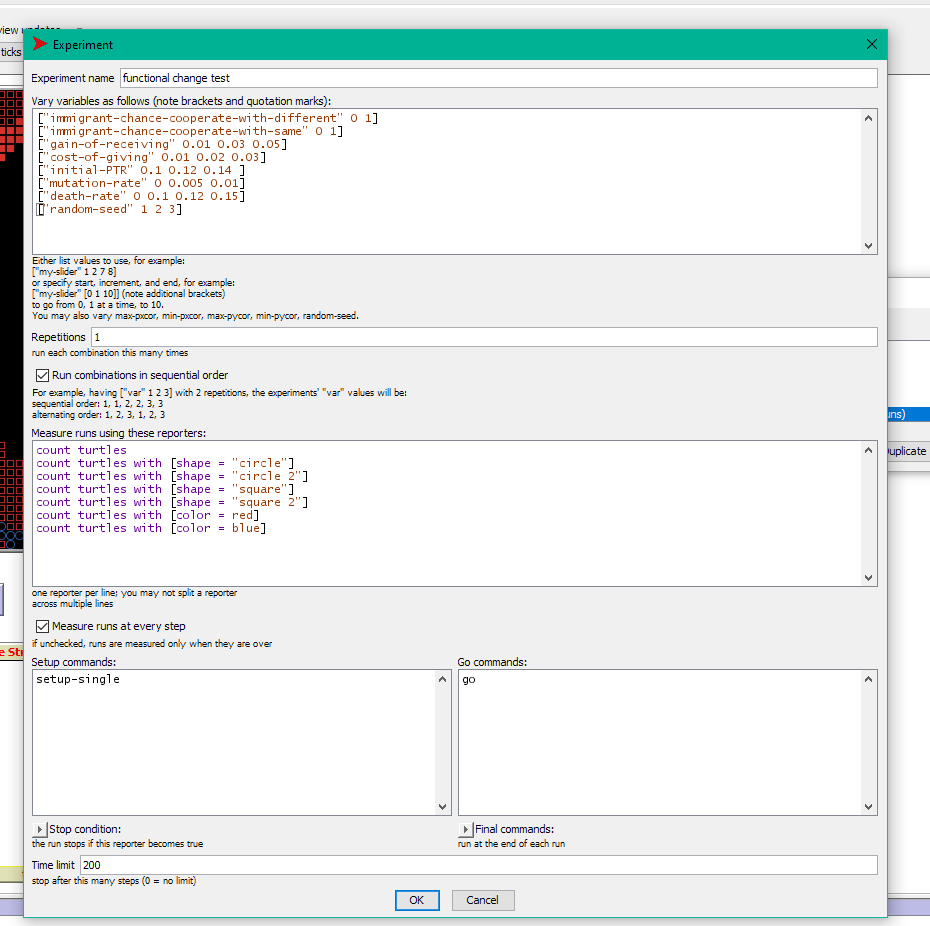
Before I do any of this, I want to do a non-functional re-write of Wilensky’s rather poorly organized code to make it more readable and easier to test, debug, and expand on. There are also a great many statistics collected on individual interactions that I will not have use for, which I will eliminate. To verify that this really is a completely non-functional rewrite, I will do a series of BehaviorSpace runs, varying different parameters, with a fixed random seed. I will then do the same runs with the same random seeds after the rewrite. If the rewrite has not had a functional effect, the output data should be identical. I am testing only a few different possible permutations of parameters, which I believe should be sufficient to establish functional identity. (I have already eliminated the immigrate and immigration-rate functions. Since I had already set immigration-rate to 0, it is self-evident that this should not change the run results.)

Dynamics of an initial test run look a little odd:

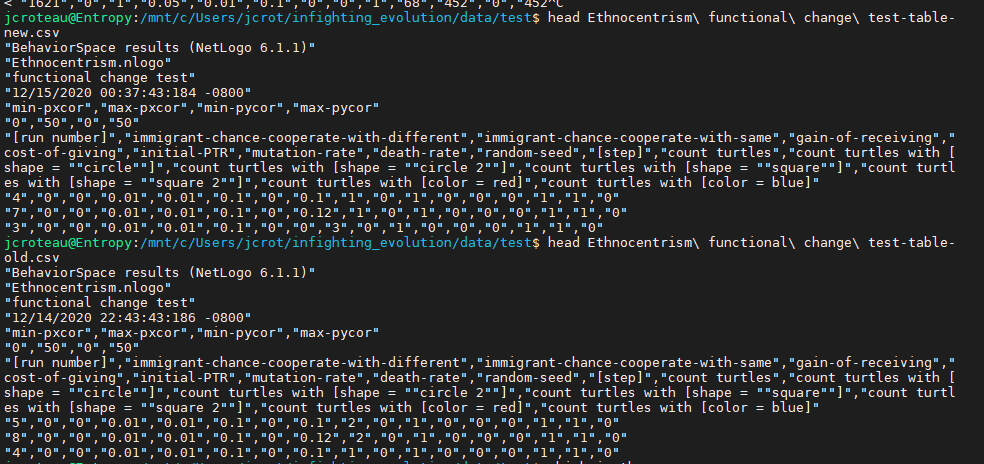
Graphical user interface

Description automatically generated

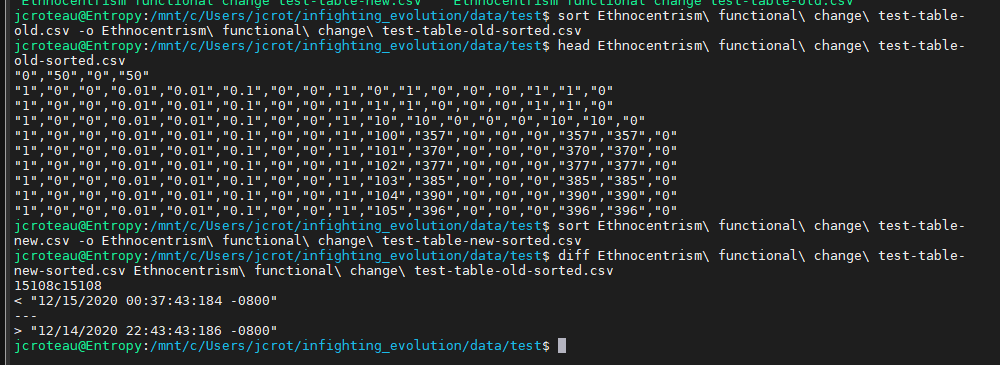
So we probably made a mistake somewhere, as the usually dominant “CD” strategy has yet to appear at all after over 2,000 ticks. The first agents were “DD” and appeared first and remained dominant for a long time. Then “DC” appeared and gradually supplanted it. When “CC” appeared, it quickly grew strongly dominant, which is unusual. It could be an error in our logic, or in the way we report strategies.



We run the BehaviorSpace test above first before making any changes, and again afterwards. After an initial non-functional change, we compare the two files:



While the headers match, we see an obvious problem: for computational efficiency, runs were made in parallel, which means that individual datapoints are output in random order. This means a simple file comparison will not work. We can however resolve this and verify that the underlying data is identical simply by sorting the files first and then comparing:



This verifies that the non-functional change is in fact a non-functional change, with the only difference in the two files being the timestamps, and underlying data generated being identical. Both files output a total of 290,075 lines (including 7 header lines) from 3,888 runs, which is about what we would expect given that runs were capped at 200 ticks, data was output for each tick, and some runs ended early when the population went extinct.

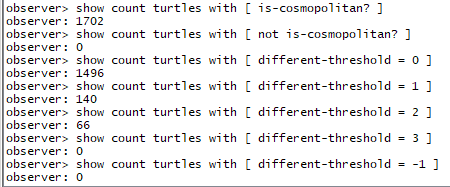
For ease of visualization, I moved the origin to the center (it had originally been in the lower left corner). This is very clearly a non-functional change as the overall topology is still toroidal. I did this so that the initial agent created would appear in the center of the map, making it more obvious what’s going on. I have also changed the colorization to HSB, which will make visualization with a larger number of tags easier.

Added integrated test code with run-tests method, including code to test test methods. This is not run as part of the normal setup routine because it includes a large number of calls to runresult with a string argument, which requires compilation and would slow the setup process. It has a button to run it at any time.

Since hamming distance is an integer, it makes sense to have difference-threshold be an integer too. If is-cosmopolitan? = false, then an agent will cooperate when hamming-distance < difference-threshold. This means the “always cooperate” strategy is equivalent to difference-threshold = num-tags + 1, and “always defect” is equivalent to difference-threshold = 0 so we bound difference-threshold in the range [0, num-tags + 1]. When mutating, difference-threshold may change by at most 1. If is-cosmopolitan? = false, the agent will defect when hamming-distance < difference-threshold

On initial setup with 1-tag, we selected from the four possible strategies with equal probability. Here, because of the nature of our variables, “CC” and “DD” are twice as likely as the others. I am ignoring this as more detailed examinations will likely not use random initial strategies. If we mutated is-cosmopolitan? at the same rate as the other variables, this would make jumps from “CC” to “DD” or “CD” to “DC” significantly more likely than in the original model, so instead we use a probability of mutation-rate^2. This slightly alters the probability distribution from the original model, though I claim that this change will not significantly alter the dynamics of the model.

An examination of the distribution of values in our turtles quickly shows us a problem:



While we have successfully constrained different-threshold in our desired range, we see that the vast majority of turtles have different-threshold = 0 and all of them have is-cosmopolitan? set to true, suggesting an error in our mutation logic. We may perhaps have been overly conservative in our mutation rate for is-cosmopolitan?. It is also somewhat odd from a validation perspective that unconditional cooperation is so strongly selected for over defectors, but not unheard given that this particular run has cost-of-giving set so low. Even in Axelrod and Hammond’s model, they had to set cost-of-giving higher to show a strong advantage for ethnocentrism over pure cooperation.

To address this issue, I set the mutation rate for is-cosmopolitan? back to being the same as other factors. This does make direct jumps from “CC” to “DD” or from “CD” to “DC” somewhat more likely than originally, however I do not believe this is core to the model. Doing a test run now, we see results that are much more familiar:

Graphical user interface

Description automatically generated

This much more closely resembles the dynamics we have previously observed, with “CC” gaining initial prominence before being crowded out by “CD”. The dynamics over the long term also match what we have previously seen:

Graphical user interface

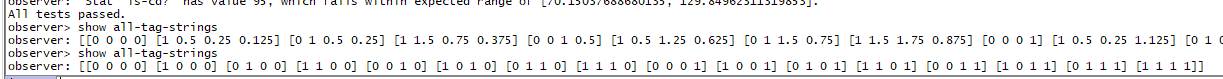
Description automatically generated

(This is the same run as above, run over a longer period of time)

Though we have verified the model at the macro level for the 1-tag case, we should also do so on the micro level before we proceed to more complex cases. We will do this by adding more unit tests. First, we re-write the code further into smaller sub-modules and then do another quick qualitative check to verify that the results still look correct, which they do. Then we write a suite of tests, all called by the run-tests command. The test suite starts with the most fundamental tests, including testing our test helper methods, and then moves into more advanced integration tests, eventually reaching a full end-to-end test. At each step, since we have already verified that the sub-components work correctly, we don’t need to re-verify every permutation in the larger-scale test. This also means that if something fails, we know the failure is likely in the first failure we see in the run sequence, as smaller modules have already been verified.

After testing deterministic behavior, we also run statistical tests based on expected probability distributions. We consider a test to have passed if the actual observed mean falls within 3 standard deviations of the expected mean (we could in principle set this range to be narrower, but this causes the tests to be flakier, and when a mean is off, it tends to be off by a very large amount). When testing, we set probability values both much smaller and much larger than we actually use in production. The larger values help make sure the probabilities we expect are accurate.

To test mutation under the more complex multi-tag guess we create many turtles, mutate many times, and then look at the probability distribution of each possible tag string, which we expect to follow an approximate binomial distribution. We write a short method to auto-generate every possible tag string, which we can easily test in the command center:



Doing this, we note an oddity in our probability distribution for different-threshold:

Text

Description automatically generated

The different-thresholds on the edges of the possible range are both half as likely as those in the middle. This does make sense with a moment’s thought—there are two possible ways to get to every other different-threshold, and only one for the edges. We would like our probability distribution to be even though, so we modify the mutate function to make mutation from one of these edges half as likely. In the long term, this will have the same effect.

Setting this we now end up with a much more even distribution (we also had initially set the expectation value for each different-threshold incorrectly, so fixed that as well):

A picture containing table

Description automatically generated