Watts & Strogatz model

duw

10/22/2019

The goal of this assignment is evaluate the Watts-Strogatz model (WS) by running Watts & Strogatz' (1998) key simulation showing the impact of p on clustering and shortest path lengths.

Overview

This assignment contains of 4 steps.

1. Place your model inside a function.
2. Run the simulation.
3. Control for the results of a regular lattice.
4. Plot the results.

Step I - Create function

1. The first step of running the simulation involves putting the Watts-Strogatz model into its a function. A function in R can be created using the code below.

```
function_name <- function(ARGUMENT1, ARGUMENT2, ARGUMENT3){

  CODE_HERE

}
```

2. In order to get your simulation into a function, you need to do three things. First, place your Watts & Strogatz code in-between the curly brackets (inplace of `CODE_HERE`). If you don't have a running version of the model, use this one:

```
# get edges
  edges = matrix(ncol = 3, nrow = size * max_dist)
  ind = 0
  for(i in 1:(size-1)){
    for(j in (i+1):size){
      dist = min(j - i, abs((j-size)-i))
      if(dist <= max_dist){
        ind = ind + 1
        edges[ind, ] = c(i, j, dist)
        }
      }
    }

  # sort edges by distance
  edges = edges[order(edges[,3],edges[,1],edges[,2]),]

  # fill matrix
  network = matrix(0, ncol = size, nrow = size)
  network[edges[,1:2]] = network[edges[,2:1]]  = 1

  # rewire
  for(i in 1:nrow(edges)){
```

```
    if(runif(1,0,1) < p){
      new_end = sample(1:size,1)
      if(new_end != edges[i, 1] & network[edges[i,1], new_end] == 0){
        network[edges[i,1], edges[i,2]] = network[edges[i,2], edges[i,1]] = 0
        network[edges[i,1], new_end]    = network[new_end, edges[i,1]]    = 1
        }
      }
    }
```

3. Now, change the names of `ARGUMENT1`, `ARGUMENT2`, and `ARGUMENT3` to the names of the three objects that control the size of the network, the neighborhood size (i.e., `max_dist`), and the probability of rewiring (in that order).
4. Finally, change the name of the function to `watts_strogatz`. That's it. You now can run your function like any other.
5. Try running `watts_strogatz(10, 2, .2)`. Does it work? Does the resulting matrix appear correct? If not, try to find the error before you proceed.

Step II - Simulate

1. In their simulation, Watts and Strogatz varied the probability of rewiring edges to evaluate its impact on clustering and average shortest path length. To do the same, we need to find a way to run the same model for different values of the rewiring probability. Solution: A for-loop. We begin by defining values for the rewiring probability using `ps <- seq(start, end, step)` where `start` is the smallest value, `end` the largest, and `step` the step-size of the sequence. Choose `start` and `end` to be `.00001` and `1`, and step size to be some number that produces, for now, no more than say 10 or 20 values.
2. Now create a loop that iterates over those `ps` and another one inside that loop that repeats the simulation for each value in `ps` exactly `n` times. Just like the code below. Chose `n` to be a small number, say 10, for now. Later you can increase it to increase the precision of your simulation. Alright, you got the backbone of your simulation.

```
n = 10
for(i in 1:length(ps)){
  for(j in 1:n){

    CODE_HERE

    return(network)
  }
}
```

3. Let's figure out how to calculate clustering and shortest path lengths. To do this, we want to use the `igraph` package. Once you installed the package (`install.packages('igraph')`) and loaded it (`library(igraph)`), you can use `graph_from_adjacency_matrix()` to load the matrix produced by your `watts_strogatz`-function into the `igraph` format. That is, assuming your `network` is an object created from `watts_strogatz`, use `graph_from_adjacency_matrix(network, mode = 'undirected')` and assign the result to a new object called `igraph_network`.
4. Now you can use the igraph functions `transitivity()` and `average.path.length()` on the new `igraph_network`-object to compute clustering and average shortest path length. In the`transitivity()` function, make sure to set `type` to `"localaverage"` to compute the type of clustering that Watts & Strogatz used.
5. Ok, now you pretty much got all the ingredients. What's left is storing the results. Create a 2-column matrix with the number of rows set to `n * length(ps)` and call it `result`. Also create an `index`

object and set it to `1`. Note both need to be defined before the loops. Then inside the inner loop assign the result of `transitivity()` to `result[index, 1]` and the result of `average.path.length()` to `result[index, 2]` and, finally, set `index` to `index + 1`.

6. Now, you are essentially ready to run the simulation. Before starting it, however, I recommend placing an `print(i)` between the inner and outer loop. This will help you keep track of the progress of the loop. Now, run the simulation once with network `size` equal to 10 and `max_dist` equal to 2. If it takes to long, reduce the length of `ps` and/or the value of `n`. If it runs fine, set the values of `size` and `max_dist` to what Watts & Strogatz used in their simualtion.

Step III - Normalization

1. Take a look at the Watts and Strogatz' paper again. You will notice that they do not show the raw clustering coefficients and average shortest path length, but those normalized by the results of the regular lattices. Obtain the necessary normalization constants, by creating networks using `watts_strogatz(size, max_dist, p = 0)` - remember `p = 0` means regular lattice - and evaluating its clustering and shortest path length using the same functions as before. Store the results as `C0` and `L0`, respectively.

2. Now normalize the simulation results using `result[, 1] <- result[, 1] / C0` and `result[, 2] <- result[, 2] / L0`.

Step IV - Aggregate & Plot

1. Ok, you ran the simulation, got results, normalized them, and now its finally time to inspect the results. First, aggregate the results of the repeated simulations. Do this using `agg_C <- tapply(res[,1], rep(ps, rep(n, length(ps))), mean)` and, respectively, the same for the average shortest path lengths. Now, plot the relationship between `ps` and `agg_C` using `plot(ps, agg_C, type = 'l', ylim = c(0, 1), log = 'x')`. Then run `lines(ps, agg_L)`, which will add a dashed line for the average shortest path lengths. What do you see? Does it look the same as in Watts & Strogatz (1998)?

```
require(igraph)

watts_strogatz = function(size = 10, max_dist = 2, p = 0){

  # get edges
  edges = matrix(ncol = 3, nrow = size * max_dist)
  ind = 0
  for(i in 1:(size-1)){
    for(j in (i+1):size){
      dist = min(c(abs(j - i),abs((j - size)-i)))
      if(i != j & dist <= max_dist){
        ind = ind + 1
        edges[ind, ] = c(i, j, dist)
        }
      }
    }

  # sort edges by distance
  edges = edges[order(edges[,3],edges[,1],edges[,2]),]

  # fill matrix
  network = matrix(0, ncol = size, nrow = size)
  network[edges[,1:2]] = network[edges[,2:1]]  = 1

  # rewire
```

```
  for(i in 1:nrow(edges)){
    if(runif(1,0,1) < p){
      new_end = sample(1:size,1)
      if(new_end != edges[i, 1] & network[edges[i,1], new_end] == 0){
        network[edges[i,1], edges[i,2]] = network[edges[i,2], edges[i,1]] = 0
        network[edges[i,1], new_end]    = network[new_end, edges[i,1]]    = 1
        }
      }
    }
  network
  }


# simulate
ps <- seq(.00001, 1, length = 20)
n = 10
res = matrix(ncol = 2, nrow = length(ps) * n)
ind = 0
for(i in 1:length(ps)){
  print(i)
  for(j in 1:n){
    ind = ind + 1
    network <- watts_strogatz(1000, 10, ps[i])
    igraph_network = graph_from_adjacency_matrix(network, mode = "undirected")
    res[ind, 1] = transitivity(igraph_network, type = 'localaverage')
    res[ind, 2] = average.path.length(igraph_network)
  }
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 16
## [1] 17
## [1] 18
## [1] 19
## [1] 20
```

```
# normalize
network_0 = graph_from_adjacency_matrix(watts_strogatz(1000, 10, 0), mode = "undirected")
res[,1] = res[,1] / transitivity(network_0, type = 'localaverage')
res[,2] = res[,2] / average.path.length(network_0)
```

```r
# aggregate & plot
agg_C = tapply(res[,1], rep(ps, rep(n, length(ps))), mean)
agg_L = tapply(res[,2], rep(ps, rep(n, length(ps))), mean)

# plot
plot(ps, agg_C, type = 'l', ylim = c(0, 1), log = 'x')
lines(ps, agg_L, lty = 2)
```