Exercise 5 – Preferential Attachment model

Open NetLogo

From the menu select File>Models Library.

Then select Sample Models>Networks>Preferential Attachment.

Click on the Info tab and read associated information. Click back on the Interface tab and try playing with the model to see what it does. This is an implementation of the Barabási & Albert model for scale-free networks from the late1990's[3]. A network is created by repeatedly adding new nodes and connecting them to an existing node selected with probability proportional to the number of links they already have.

We will now examine the program code in detail.

Select the Code tab. You should see something similar to:

```
;;;;;;;;;;;;;;;;;;;;;;;;
;;; Setup Procedures ;;;
;;;;;;;;;;;;;;;;;;;;;;;;

to setup
  clear-all
  set-default-shape turtles "circle"
  ;; make the initial network of two turtles and an edge
  make-node nobody        ;; first node, unattached
  make-node turtle 0      ;; second node, attached to first node
  reset-ticks
end

;;;;;;;;;;;;;;;;;;;;;;;;
;;; Main Procedures ;;;
;;;;;;;;;;;;;;;;;;;;;;;;

to go
  ;; new edge is green, old edges are gray
  ask links [ set color gray ]
  make-node find-partner         ;; find partner & use it as attachment
                                 ;; point for new node
  tick
  if layout? [ layout ]
end

;; used for creating a new node
to make-node [old-node]
  crt 1
  [
    set color red
    if old-node != nobody
      [ create-link-with old-node [ set color green ]
        ;; position the new node near its partner
        move-to old-node
        fd 8
      ]
  ]
end

;; This code is the heart of the "preferential attachment" mechanism, and acts like
;; a lottery where each node gets a ticket for every connection it already has.
;; While the basic idea is the same as in the Lottery Example (in the Code Examples
;; section of the Models Library), things are made simpler here by the fact that we
```

[3] Albert-László Barabási & Reka Albert. Emergence of Scaling in Random Networks, Science, Vol 286, Issue 5439, 15 October 1999, pages 509-512.

```
;; can just use the links as if they were the "tickets": we first pick a random link,
;; and than we pick one of the two ends of that link.
to-report find-partner
  report [one-of both-ends] of one-of links
end

;;;;;;;;;;;;;;
;;; Layout ;;;
;;;;;;;;;;;;;;

;; resize-nodes, change back and forth from size based on degree to a size of 1
to resize-nodes
  ifelse all? turtles [size <= 1]
  [
    ;; a node is a circle with diameter determined by
    ;; the SIZE variable; using SQRT makes the circle's
    ;; area proportional to its degree
    ask turtles [ set size sqrt count link-neighbors ]
  ]
  [
    ask turtles [ set size 1 ]
  ]
end

to layout
  ;; the number 3 here is arbitrary; more repetitions slows down the
  ;; model, but too few gives poor layouts
  repeat 3 [
    ;; the more turtles we have to fit into the same amount of space,
    ;; the smaller the inputs to layout-spring we'll need to use
    let factor sqrt count turtles
    ;; numbers here are arbitrarily chosen for pleasing appearance
    layout-spring turtles links (1 / factor) (7 / factor) (1 / factor)
    display  ;; for smooth animation
  ]
  ;; don't bump the edges of the world
  let x-offset max [xcor] of turtles + min [xcor] of turtles
  let y-offset max [ycor] of turtles + min [ycor] of turtles
  ;; big jumps look funny, so only adjust a little each time
  set x-offset limit-magnitude x-offset 0.1
  set y-offset limit-magnitude y-offset 0.1
  ask turtles [ setxy (xcor - x-offset / 2) (ycor - y-offset / 2) ]
end

to-report limit-magnitude [number limit]
  if number > limit [ report limit ]
  if number < (- limit) [ report (- limit) ]
  report number
end

; Copyright 2005 Uri Wilensky.
; See Info tab for full copyright and license.
```

Discussion of code:

The setup procedure:

All turtles (representing nodes) are set to the circle shape then two new nodes are created by calling the make-node procedure (defined below in the code). The make-node procedure takes one argument which is an existing node or nobody. Nobody is the null value used by NetLogo. It creates a new node (turtle) and then links it to the existing node (if it is not set to nobody). The new link is set to the colour green. Then the new node is moved to the same position as the old node and then moved forward 8 patch lengths. Since

new turtles are assigned a random heading this means the node will be positioned 8 patch lengths away in some random direction.

The go procedure:

Firstly all links are set to gray. The a new node is added using the make-node procedure. The existing node is determined by the find-partner reporter procedure (defined below in the code). The layout procedure is then called if the “layout?” switch is set to on in the Interface screen. The layout procedure (defined below in the code) maintains a nice looking visualisation of the network.

The find-partner procedure performs the preferential attachment function by returning (reporting) an existing node probabilistically proportional to its number of links. This is accomplished in a single line:

```
report [one-of both-ends] of one-of links
```

Working from right to left. Firstly “one-of links” returns a randomly chosen link from the agentset “links” which contains all links. Secondly “[one-of both-ends]” returns a randomly chosen turtle (node) from the agentset “both-ends” for the chosen link. For a given link both-ends returns an agentset containing the two turtles at each end of the link.

Hence the line equates to selecting a link at random and then selecting one end of that link at random.

The layout procedure:

The layout procedure is only concerned with the visual layout of the network on the screen.

It uses the layout-spring command to space out the nodes and minimise the link lengths. By repeating this a number of times, adapting the parameters passed to the layout-spring command and calling the display command during the repeat it creates a smooth visualisation of the network growth.

Additionally it calculates an x and y offset based on the most extreme nodes in the network (based on x and y coordinates). It uses this offset to reposition each node such that the network remains centred within the environment. Note: to make sense of the offset process remember that the coordinates have the origin (0,0) in the centre of the environment. When you run the program you may notice that as the network grows all the nodes occasionally shift together to keep the network centred in the environment.

The degree distribution plots:

On the Interface screen are two degree distribution plots. One is a histogram and one is a log / log plot. Let us see how these plots are defined.

Right click on the uppermost “Degree Distribution” histogram plot and select “Edit” from the context menu.

The Plot dialogue should appear. Notice some code in the “Pen update commands” section.

Click on the pen icon to the right of the code.

This opens the “Advanced pen options” dialogue. Notice that the “Mode” dropdown is set to “Bar” and that the Interval field is set to “1.0”. This means the histogram will use bars and the width of a bar will be one unit.

Resize the window so you can see all the code in the “Pen update commands” field. You should see something similar to:

```
if not plot? [ stop ]
let max-degree max [count link-neighbors] of turtles
plot-pen-reset  ;; erase what we plotted before
set-plot-x-range 1 (max-degree + 1)  ;;+1 to make room for width of the last bar
histogram [count link-neighbors] of turtles
```

This code dynamically resizes the x-axis range of the histogram based on the maximum degree in the population of nodes. This is necessary because the histogram command on its own does not automatically rescale the x-axis.

Firstly the plot terminates if the “plot?” switch is set to off in the Interface. Then a local variable “max-degree” is defined with “let” which stores the maximum degree in the network. The link-neighbours reporter returns an agentset of all turtles linked to a given turtle. Hence “[count link-neighbors] of turtles” returns a list of the number of links (degree) of each turtle (node). The max reporter returns the maxmum value in that list. This value is then used to rescale the x-axis using the set-plot-x-range command. Finally the historgram is plotted for degrees of each turtle (node).

Close the dialogues and return to the Interface screen.

Right click on the lower “Degree Distribution” (log-log) plot and select “Edit” from the context menu. Again examine the code by clicking the pen icon.

You should see something similar to:

```
if not plot? [ stop ]
let max-degree max [count link-neighbors] of turtles
;; for this plot, the axes are logarithmic, so we can't
;; use "histogram-from"; we have to plot the points
;; ourselves one at a time
plot-pen-reset  ;; erase what we plotted before
;; the way we create the network there is never a zero degree node,
;; so start plotting at degree one
let degree 1
while [degree <= max-degree] [
  let matches turtles with [count link-neighbors = degree]
  if any? matches
    [ plotxy log degree 10
             log (count matches) 10 ]
  set degree degree + 1
]
```

Netlogo does not provide built-in functions for plotting log scale histograms hence this code uses the plotxy function to plot points directly onto the chart area. Again max-degree is set to the maximum degree in the network then a whole loop is used to iterate from 1 up to max-degree. In each iteration the number of nodes with the given degree are assigned to an agentset variable called “matches”. The count of this agentset (the size) is the plotted onto the y-axis with the degree plotted on the x-axis (using log 10).

Task: Create a new Chooser dropdown that allows the user to select from “random” or “pref”. Modify the program so that when “pref” is selected it functions unchanged using preferential attachment but when “random” is selected new nodes are connected to a randomly selected existing node irrespective of the number of links it has. Try running the program and compare the difference in the produce degree distributions between random and preferential attachment.

Task: Create two new buttons called “attack-nodes” and “attack-links”. When these buttons are pressed they should immediately attack the network. Attack-nodes should remove half

of the nodes from the network and attack-links should remove half of the links from the network. Trying using these two attacks on both random and preferential attachment forms. Do you notice any difference in their effect?

Task: Create a procedure in the program called calc-comps and two new global variables called NCC and LCC. Calc-comps should calculate the number of components (and assign this to NCC) and the size of the largest component (assigned to LCC). Note: a component is a connected set of nodes such that there is at least one path between any two nodes. Hence any graph that has not been attacked in our model will have NCC = 1 and LCC = total number of nodes. Create a button called “calc-comps” that calls the procedure and create two monitors that display NCC and LCC. Experiment by running the model for some period. Stop it, perform an attack and then click the calc-comps button to see the effect on NCC and LCC. Compare the effect on random and preferential networks.