Agent-Based Modelling in NetLogo

Networks

David Hales

www.davidhales.com/abm-netlogo

Resources for learning networks

- See Links in NetLogo programming guide (Help > User Manual > Programming Guide > Links): https://ccl.northwestern.edu/netlogo/docs/programming.html#links
- See Links section in NetLogo dictionary (Help > Dictionary > Links): https://ccl.northwestern.edu/netlogo/docs/dictionary.html

Networks – undirected links

- NetLogo implements networks using Links
- Links is an agentset like turtles and patches
- A turtle can create a link with another turtle:

```
ask turtle 0 [create-link-with turtle 1]
ask turtles [create-link-with one-of other turtles]
```

- Many links can be created in one go:

```
ask turtles [create-links-with other turtles]
```

Networks

- Some link primitives that turtles can use:

  my-links ; returns set of my links

  link-neighbors ; returns turtles I am linked to

  link-neighbor? turtle ; true if turtle is my neighbour

- Examples of use:

  ask my-links [die] ; kill all my links

  ask link-neighbors with [color = red] [die]

  mean [count my-links] of link-neighbors

Networks – directed links

- Using “with” creates undirected links
- Directed links created with “from” and “to”:

```
ask turtle 0 [create-link-to turtle 1]
ask turtle 0 [create-link-from turtle 1]
```

- Primitives use “in” and “out”:

```
my-in-links ; set of links directed in to me
my-out-links ; set of links directed out from me
in-link-neighbors ; return my in link neighbors
out-link neighbor? turtle ; out link to turtle exists?
```

Networks – link properties

- A link can be identified by end turtle numbers:

  ask link 3 4 [die] ; kill link between turtle 3 & 4

- Links have variables for color, thickness etc.
- Include two variables end1 and end2 containing the turtles at each end of the link
  - undirected links: end1 = lowest who valued turtle
  - directed: it is direction of the link: end1 -> end2
- If a turtle dies then all its links die

Networks – turtle positions

- Links are drawn as lines between turtles (directed links as arrows)
- Turtle positions are not affected unless explicit commands are used
- Layout primitives move the turtles around to better visualise the network – e.g.:

  layout-radial turtles links root-turtle

  layout-spring turtles links tautness length repulsion

- tie and untie primitives tell a link to become "fixed" such that movement in one turtle is appropriately copied by the other turtle.

Networks – link breeds

- In the same way that turtle breeds can be created so can link breeds:

  undirected-link-breed [edges edge]

  directed-link-breed [arrows arrow]

- Then the link primitives use the breed name:

  create-edge-with turtle 0

  count my-edges

  ask edge-neighbors [set color red]

Task 1 – wire a random graph

- Write a program with:
  - Two sliders (input values):
    - N – number of nodes [1..100]
    - P – probability that any two nodes are connected with an undirected edge [0..1]
  - Two buttons:
    - Setup – calls procedure “setup” that creates the nodes
    - Rand – calls procedure “rand” that wires nodes with the Probability P
  - Two monitors (output values):
    - Edges – displays total number of edges in the network
    - Max degree – number of links of the most connected node

Hint: You *may* need to use a “foreach” loop or a “while” loop (which you need to lookup in the NetLogo dictionary).

Task 1: three ways of doing it

```
to setup
  clear-all
  create-turtles N [set shape "circle"]
  layout-circle turtles 10
end

; cheap but inefficient
to random-v1
  ask turtles [
    create-links-with other turtles ]
  ask links [ if prob (1 - p) [die] ]

; systematic
to random-v2
    foreach sort turtles [
    ask turtles with [who > [who] of ?]
    [
      if prob p [ create-link-with ? ]
    ]
  ]
end
```

```
; using nested while loops
to random-v3
  let node-count count turtles
  let i 0   ; start i at node 0
  while [i < node-count] [
    let j i + 1 ; start j at node i + 1
    while [j < node-count] [
      if prob p [
        ask turtle i [
          create-link-with turtle j
        ]
      ]
      set j j + 1   ; next j
    ]
    set i i + 1     ; next i
  ]
end

to-report prob [x]
  report (random-float 1 < x)
end
```

Graph topologies and measures

- Common graph topologies:
  - Random: all edges equally likely
  - Scale-free: degree distribution = power law
  - Small-world: clustered neighbours + long links
  - Lattice: neighbours connected in a space
- Common graph measures:
  - Clustering Coefficient (C): proportion of neighbours who are also neighbours
  - Average path length (L): average shortest distance between all pairs of nodes

Network examples program

- Download network examples program from the labs webpage
- It gives examples of various topologies, measures and layouts.
- We will briefly look at two topology creation algorithms:
  - Pref. attachment for scale-free networks
  - Lattice rewiring for small-world networks

Note: many of the functions implemented by the program (and many other useful ones) can be performed with the networks extension more efficiently. See “networks” under “extensions” in the NetLogo user manual.

Preferential attachment

- Add new nodes one-by-one
- Each new node makes one link to one other existing node:
  - select existing node probabilistically proportionately to it's existing number of links
  - e.g. a node with k links has half the probability of being selected as a node with 2k links
- Hence the "rich get richer" => scale-free degree distribution

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

Create a pref. attach. network with N nodes:

```
; create a preferential attachment network of N nodes
to create-pref-attach-network [N]
  ; create two nodes and link them together
  create-turtles 2
  ask turtle 0 [create-link-with turtle 1]
  ; create the rest of the nodes
  create-turtles N - 2 [
    ; new node makes link to an old node by
    ; selecting a random link and then selecting
    ; a random end of that link - giving a node
    create-link-with [one-of both-ends] of one-of links
  ]
  ; radial layout with first turtle at centre
  layout-radial turtles links turtle 0
end
```

also see: wire-small-world procedure in the network examples program

Preferential attachment model

- Model from netlogo models library: sample models > networks > preferential attachment
- This model animates the process of preferential attachment to produce a scale-free network
- It displays the degree distribution of the network as it forms in two plots
- Note: detailed explanation of the code and some tasks are given in exercise sheet on the labs page

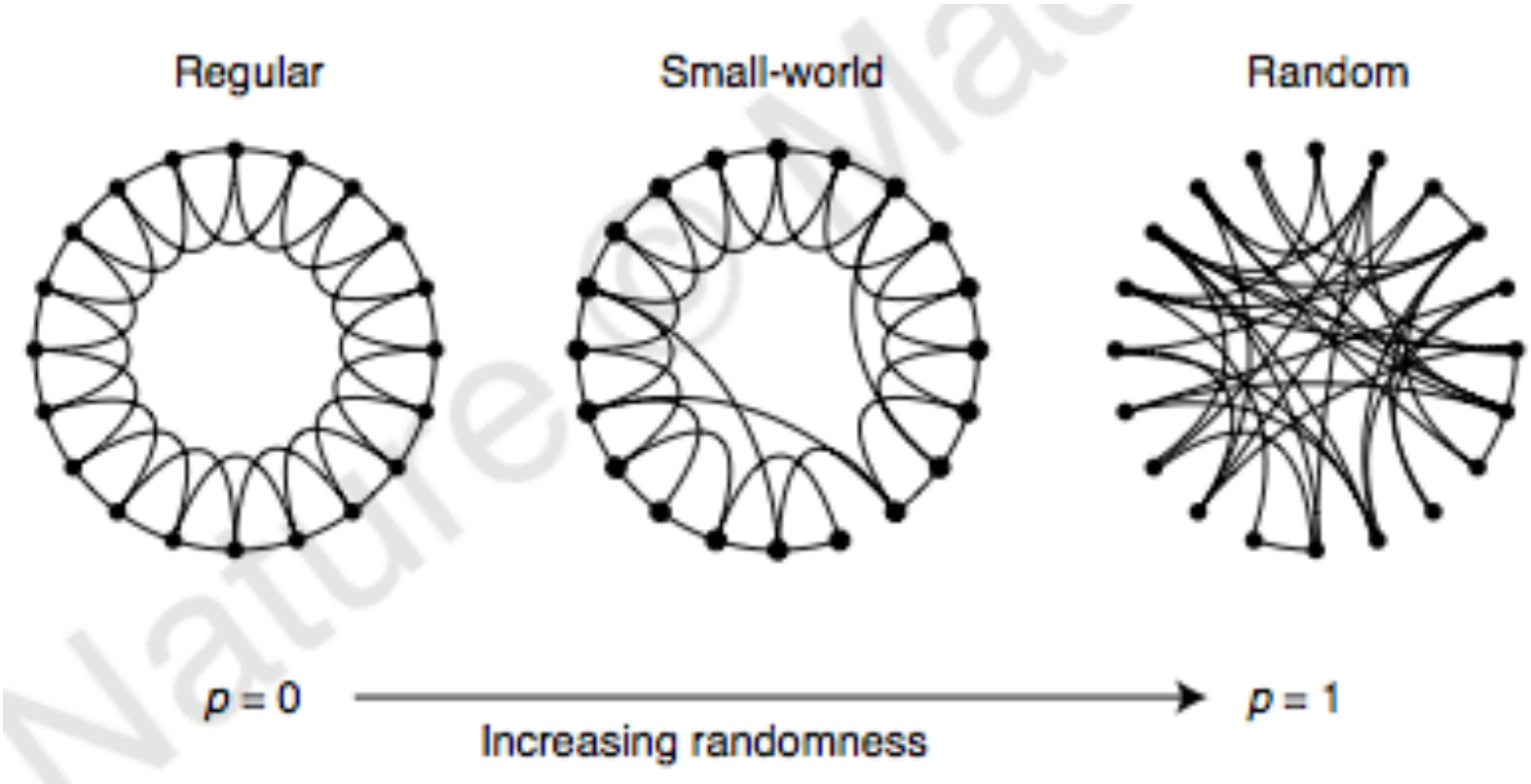
Small-world network

- Create a 1D lattice of N nodes in which each node has K neighbours
- Then with probability P rewire each edge:
  - one end of edge stays connected
  - other end rewired to random other node
  - Total number of edges stays constant
  - restricting rewire end to “forward edges” makes sure each node keeps at least K/2 edges
- Produces high C and low L for certain params.

Paper: DJ Watts and SH Strogatz. Collective dynamics of ‘small-world’ networks, Nature, 393:440-442 (1998)

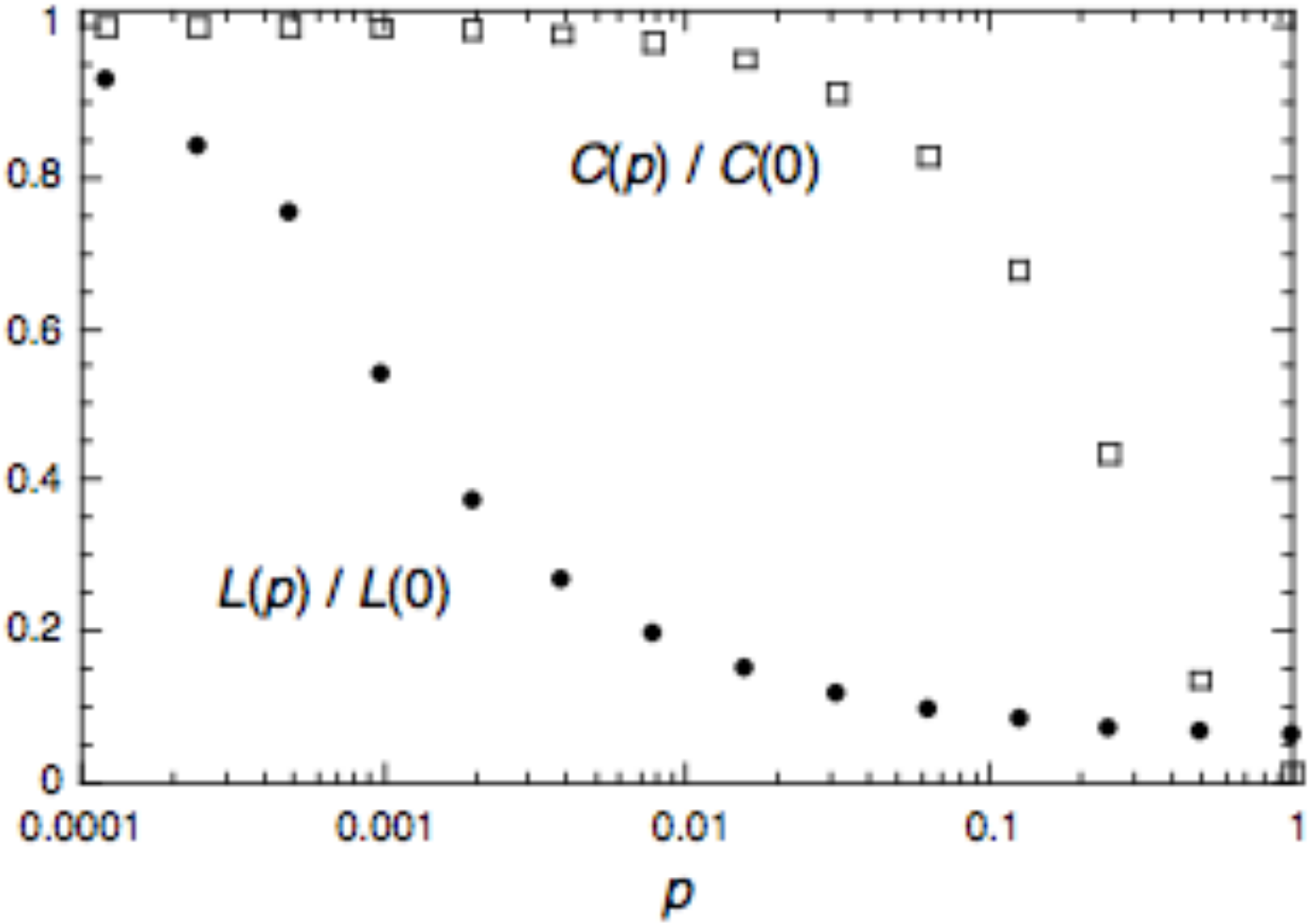
Small-world networks

Example: N=20, K=4, varying P:



From: DJ Watts and SH Strogatz. Collective dynamics of 'small-world' networks, Nature, 393:440-442 (1998)

Small-world networks



From: DJ Watts and SH Strogatz. Collective dynamics of 'small-world' networks, Nature, 393:440-442 (1998)

create small-world network

```
; create a small-world network of N nodes,
; with K degree and rewire prob P
to create-small-world-network [N K P]
  ; first create a 1D lattice
  create-turtles N      ; create nodes
  ask turtles [         ; make forward links for each turtle
    let links-done 0
    while [links-done < K / 2] [
      set links-done links-done + 1
      ; use mod N to wrap around the ring N+1 becomes 0
      create-link-with turtle ((who + links-done) mod N)
    ]
  ]
```

```
  ; then rewire links
  ask links [                    ; ask all existing (old) edges
    if P > random-float 1 [      ; with probability p (rewire)
     ; new edge from node end1 to random other node
     ask end1 [
       create-link-with one-of other turtles with
         [ not link-neighbor? myself]
     ]
     die ; remove the old edge
    ]
  ]
  ; layout turtles in a sorted circle
  layout-circle (sort turtles) max-pxcor - 2
end
```

This rewire links section *is not entirely correct* (can you work out why?).
See the *wire-small-world* procedure in the network examples program.

Small-world network model

- Load model from netlogo models library:
  sample models > networks > small worlds
- This model assumes nodes have k = 4 neighbours
- Lets you rewire one-by-one (or all at once) and displays C and L on a plot
- It also lets you highlight a node and see individual C and L statistics for the node

Virus on a network model

- Load model from netlogo models library: sample models > networks > virus on a network
- Read Info tab and play with model
- It wires a spatial network in which each node links to some number of other nodes that are closest to it in the 2D space
- **Task 2: Modify model to wire *some other* topology (e.g. random, scale-free, small-world)**
- How does it make a difference to the dynamics?

Task 2: pref. attach.

Add the following procedure and call it instead of the spatial-setup procedure:

```
to setup-pref-attach-network
  ask turtle 0 [create-link-with turtle 1]
  ask turtles with [count my-links = 0] [
   create-link-with [one-of both-ends] of one-of links
  ]
  layout-radial turtles links (turtle 0)
end
```

Here we can assume a population of turtles have already been created and they don't have any links.

Note: code example on lab page includes this and small-world networks

Task 3: Turtles moving on a network

- Write a program to wire some network topology with 30 nodes
- Create a set of 3 turtles that randomly walk on the network
  - Initially place the turtles on random nodes
  - Each time step they select a edge from their current node at random and move to the node linked to by the edge
- Hints: It helps to create two breeds of turtles one for the nodes and one for the moving turtles. The move-to command moves a turtle to the location of another turtle

Task 3: See example models

- Two sample models show one way of implementing this:
  - Sample models > code examples > link-walking turtles example
  - Sample models > code examples > lattice-walking turtles example

Dynamic networks

- Sample models > Networks > Team Assembly
- Implementation of a dynamic network ABM that relates to a theory of creative team formation
- Related to empirical data (plays, movies, scientific papers)

Paper: R Guimera, B Uzzi, J Spiro, L Amaral (2005) Team Assembly Mechanisms Determine Collaboration Network Structure and Team Performance. Science, 308(5722), p697-702