Network Software Modelling

MIS40550

James McDermott

UCD

April 2017

Outline

Note

These slides are a sneak preview of a topic we'll cover in Week 10. I'm making them available now, with accompanying code `epidemic.py`, to help you get started on Assignment 2, and so that you know the basic model which we will cover in class. In Assignment 2 some marks are awarded for "originality": to get marks for this, you'll want to avoid doing exactly the same model as we cover in class. So people need to know what that model is. Note that "originality" refers here to just avoiding what we do in class. You don't have to do something completely different: a variant of the model is enough for originality (and bonus marks for something completely original). You're not required to produce academic novelty, or carry out a literature review. (You still need to cite any sources you do use, of course.)

Section 1

Epidemic models

Epidemic models

An **epidemic** is a disease which is picked up by a large proportion of the population. Network simulations can be used to study the dynamics and behaviour of epidemics in different conditions (e.g. varying probability of infection) and in different types of networks (e.g. a fully connected graph versus a sparse one).

Despite the name, epidemic models also give insight into other scenarios:

- Computer viruses
- Information flow
- Viral videos
- Chain letters and pyramid schemes
- Zombie apocalypses

SI and SIR models

There are two main epidemic models:

- Susceptible-Infectious (SI)
- Susceptible-Infectious-Recovered (SIR)

S, I, D and R refer to **states** that an individual can be in at any time.

SI

In SI, an individual in state S is susceptible to picking up the infection (changing to I) via contact with an individual in state I.

"Contact" means via an edge in the network.

An individual who moves to state I will survive for some time, then die and drop out of the simulation.

SIR

In SIR, an individual in state I after some time may either die or recover (move to state R), with a certain probability. In state R, the individual is not susceptible to getting the disease again.

Solving and simulating

- For these simple models, if the graph is **fully connected**, the model can be written down as differential equations and solved to fully understand the epidemic dynamics.
- For more interesting graphs, it's easier to just **simulate**: create a graph, write code to express how nodes move from one state to another (probabilistically), and run it!

Time versus time-steps

In real life, time is continuous.

In a typical simulation, time is divided into **time-steps**: $t = 0, 1, 2, \ldots$. Changes to state happen synchronously ("all at once") at each time-step. The difference is a bit like the difference between an arcade game and chess.

How small should time-steps be? For realistic weather and physics, as small as possible! For epidemic models, see next slide.

Simulation parameters

- G: the graph whose edges represent contacts (e.g. being in same room, for common cold);
- p: probability of infection from an individual in state I, along each edge, **per time-step**;
- t_d: number of **time-step**s between infection and death
- i: number of nodes initially infected: $i = 1$ means there is a single "patient zero".

p and t_d must be appropriate to the time-step. E.g. if the time-step is milliseconds (probably far too small for our purposes), then p needs to be very small and t_d needs to be very large.

Update function

```python
def infection_update(s1, ss):
    # Return new state of node s1, given states of neighbours ss.

    if s1 < 0:
        return s1 # s1 < 0 means node has died, so no change
    if s1 > td:
        return -1 # t time steps after infection, node dies
    if s1 > 0:
        return s1 + 1 # one time-step has elapsed

    # if not yet infected, each infected neighbour is a new risk!
    for s in ss:
        if s > 0: # neighbour s is infected but still alive
            if random.random() < p:
                # with probability p, become infected
                return 1
    return 0
```

Statistics

Proportion of individuals still alive:

```
palive = sum(G.node[i]["state"] >= 0 for i in G.nodes()) / n
```

Proportion of individuals infected and alive:

```
pinf = sum(G.node[i]["state"] > 0 for i in G.nodes()) / n
```