Unholy Epidemiology

Sayed Abdulmohsen Alhashemi

Introduction

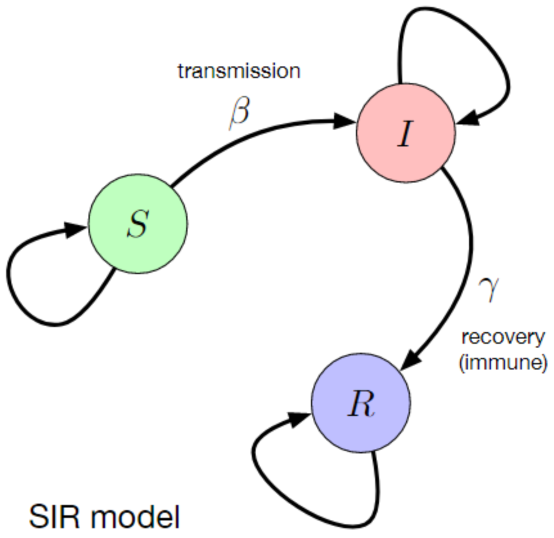
For my final project I wanted to do something related to one of my favorite genres of fiction: zombies. Luckily enough, there is a wide range of intersections between mathematical modeling and zombie epidemiology and so I found a research paper from which I could recreate some epidemiological findings.

Methods

I sought to recreate an epidemiological model of zombie infections as it was defined in Alemi et al's *You Can Run, You Can Hide: The Epidemiological and Statistical Mechanics of Zombies.* This paper defined a modified version of the SIR model to fit with a zombie virus, namely, the World War Z zombie virus.

To give some background on why this is necessary we need to enter a discussion on the mechanics of traditional viruses/infections and zombie viruses/infections. Traditional viruses and infections have various methods of transmission – airborne, waterborne, sexual transmission, etc. – and generally will also have a chance to recover from infection. In epidemiology this can be modeled to varying degrees of accuracy using the SIR model of infections. In the SIR model, nodes can be either Susceptible (S), Infected (I), or Recovered (R). Susceptible individuals are vulnerable to infection by Infected individuals. Infected individuals have a chance to transmit their infection to Susceptibles with a probability β. Once infected, a person has the chance to recover with a probability γ. With these states in mind we can construct a tree for each, representing nodes as states and edges as transitions between states. (Fig 1.)

Figure 1: Graph showing transition states of SIR model.[1]



The SIR model is great at modeling traditional viruses largely as a result of the Recovered state and the way individuals can transition into it. However, zombie infections are fundamentally different and therefore require a new set of dynamics through which they can be modeled.

Enter the SZR model. The SZR model is an epidemiological model created to model the dynamics of a zombie infection, namely, the World War Z zombie plague. World War Z zombie variants are the most simple variant of zombies to model epidemiologically but have the potential to result in some unique behaviors. These types of zombies infect others through traditional methods that can be modeled with the SIR model but lack the ability to recover from the infection. In an SZR model, Susceptibles can be infected if they are in contact with a Zombie with a probability β – same as the SIR model. The difference is in recovery. The SZR model has no parameter γ which determines the probability that an I node (in this case a Z node) transitions to an R node. Instead, the SZR model

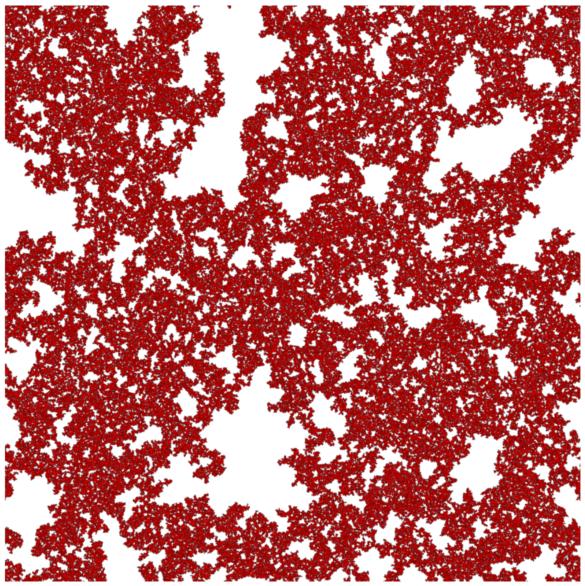
[1] Clauset lecture notes lecture 7

employs another parameter κ which defines the probability that a human kills a zombie.

This change to the state graph introduces new dynamics to the model. In the paper I'm recreating, they ran a stochastic simulation on a lattice structure to observe some critical behaviors in the equations they used for their model.

The Lattice Simulation

Figure 2.



[2]

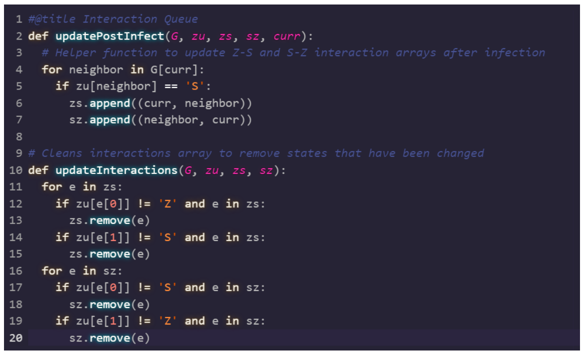
The lattice pictured above is the result of running the stochastic simulation of an SZR model. The way this was done was by maintaining a queue of Z-S interactions and simulating the infection probability with the equation: $\frac{\beta}{\beta+\kappa}$ and a probability to kill with the equation:$\frac{\kappa}{\beta+\kappa}$. In addition to this the study maintained a variable α that was defined as the ratio of bites (β) to kills (κ). This variable gives us a good way of determining the extinction probability of an infection given the two variables β and κ: $\alpha = \frac{\kappa}{\beta}$.

According to the paper, the relevance of a lattice simulation is tied to the fact that zombie viruses spread similarly to agricultural infections. Infections that spread through farms spread in a lattice due to the organization of different plots relative to each other. This pattern of infection is similar to how zombie infections would spread and – according to the WWZ book – did spread. The paper states that zombies can't board planes and will instead spread within their local populations. Thus, the lattice simulation.

In order to model their lattice simulation I had to translate their equations as best I could into python code. My starting point was the epidemiology in-class labs we'd done this semester as well as the seventh problem-set on epidemiology which had some good simulation code. The difficulty here came in rewriting the simulation code to accommodate for an interaction queue. The code looped through all the edges within a graph and then checked whether or not the set of nodes making up that edge constituted an S-I interaction pair. This was great in terms of clarifying the process to students learning these processes but in terms of running large-scale simulations, it significantly slowed down the code. And so, I created the interaction queue data structure and the functions associated with updating it. Instead of looping through the list of edges in a graph, I maintained an array of S-I interactions (S-Z) which I would loop through and run the bite and kill simulations. This is the same process[3] that was stated to have been used in the creation of figure 2.

[2] You Can Run, You Can Hide: The Epidemiology and Statistical Mechanics of Zombies Alexander A. Alemi,1, ∗ Matthew Bierbaum,1, † Christopher R. Myers,1, 2, ‡ and James P. Sethna1, § 1Laboratory of Atomic and Solid State Physics, Cornell University, Ithaca, NY 14853 2 Institute of Biotechnology, Cornell University, Ithaca, New York (Dated: June 5, 2015) p.6

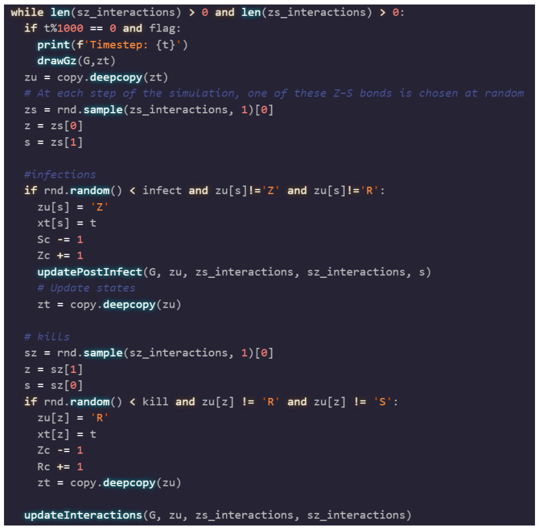
[3] You Can Run, You Can Hide: The Epidemiology and Statistical Mechanics of Zombies), p.6



```python
#@title Interaction Queue
def updatePostInfect(G, zu, zs, sz, curr):
  # Helper function to update Z-S and S-Z interaction arrays after infection
  for neighbor in G[curr]:
    if zu[neighbor] == 'S':
      zs.append((curr, neighbor))
      sz.append((neighbor, curr))

# Cleans interactions array to remove states that have been changed
def updateInteractions(G, zu, zs, sz):
  for e in zs:
    if zu[e[0]] != 'Z' and e in zs:
      zs.remove(e)
    if zu[e[1]] != 'S' and e in zs:
      zs.remove(e)
  for e in sz:
    if zu[e[0]] != 'S' and e in sz:
      sz.remove(e)
    if zu[e[1]] != 'Z' and e in sz:
      sz.remove(e)
```

Figure 3.

The above code is an example of the operations that needed to be done in order to update the interaction queues after simulating infections and kills. The updatePostInfect function updated the interactions in the Z-S and S-Z arrays since I only deal with undirected edges in the graphs used throughout this project.

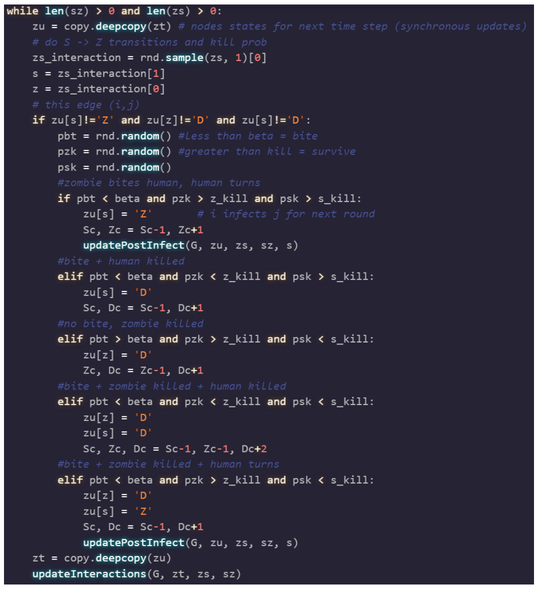


```python
while len(sz_interactions) > 0 and len(zs_interactions) > 0:
  if t%1000 == 0 and flag:
    print(f'Timestep: {t}')
    drawGz(G,zt)
  zu = copy.deepcopy(zt)
  # At each step of the simulation, one of these Z-S bonds is chosen at random
  zs = rnd.sample(zs_interactions, 1)[0]
  z = zs[0]
  s = zs[1]

  #infections
  if rnd.random() < infect and zu[s]!='Z' and zu[s]!='R':
    zu[s] = 'Z'
    xt[s] = t
    Sc -= 1
    Zc += 1
    updatePostInfect(G, zu, zs_interactions, sz_interactions, s)
    # Update states
    zt = copy.deepcopy(zu)

  # kills
  sz = rnd.sample(sz_interactions, 1)[0]
  z = sz[1]
  s = sz[0]
  if rnd.random() < kill and zu[z] != 'R' and zu[z] != 'S':
    zu[z] = 'R'
    xt[z] = t
    Zc -= 1
    Rc += 1
    zt = copy.deepcopy(zu)

  updateInteractions(G, zu, zs_interactions, sz_interactions)
```

Figure 4.

Figure 4 is a code snippet from my SZR simulation code. This was my attempt at recreating the paper's results programmatically. I'm no stranger to the hellscape that is another person's code so I will proceed with a brief breakdown of what's going on here. Similar to our in-class and problem-sets I have the variables Sc, Zc, and Rc which count the number of S, Z, and R at each time step. The sz_interactions and zs_interactions are arrays containing interactions between S nodes and Z nodes from which I randomly sample and simulate kills and infections by generating a random variable and checking its value against the variables "infect" and "kill". Those two variables are defined according to the rules set out earlier for β and κ.

In addition to the SZR model I implemented my own SZD model in an attempt to simulate additional aspects of zombie virus dynamics that I believe wouldn't be covered by the SZR model. The SZR model considers two possibilities: a person is infected or a zombie is killed. There is no accounting for the myriad of permutations to these two outcomes. As is seen in shows like The Walking Dead or any zombie media really, there are scenarios where a zombie is killed but a person is infected regardless. The way I had programmatically implemented the paper's simulation code couldn't account for any combination of infections or kills as it simulated bites first and then kills. This was an arbitrary decision as I believed it wouldn't result in any tangible differences in the model. The SZD model is a modified SZR model that allows for the simulation of bites and kills simultaneously through the usage of additional variables.



```python
while len(sz) > 0 and len(zs) > 0:
    zu = copy.deepcopy(zt) # nodes states for next time step (synchronous updates)
    # do S -> Z transitions and kill prob
    zs_interaction = rnd.sample(zs, 1)[0]
    s = zs_interaction[1]
    z = zs_interaction[0]
    # this edge (i,j)
    if zu[s]!='Z' and zu[z]!='D' and zu[s]!='D':
        pbt = rnd.random() #less than beta = bite
        pzk = rnd.random() #greater than kill = survive
        psk = rnd.random()
        #zombie bites human, human turns
        if pbt < beta and pzk > z_kill and psk > s_kill:
            zu[s] = 'Z'       # i infects j for next round
            Sc, Zc = Sc-1, Zc+1
            updatePostInfect(G, zu, zs, sz, s)
        #bite + human killed
        elif pbt < beta and pzk < z_kill and psk > s_kill:
            zu[s] = 'D'
            Sc, Dc = Sc-1, Dc+1
        #no bite, zombie killed
        elif pbt > beta and pzk > z_kill and psk < s_kill:
            zu[z] = 'D'
            Zc, Dc = Zc-1, Dc+1
        #bite + zombie killed + human killed
        elif pbt < beta and pzk < z_kill and psk < s_kill:
            zu[z] = 'D'
            zu[s] = 'D'
            Sc, Zc, Dc = Sc-1, Zc-1, Dc+2
        #bite + zombie killed + human turns
        elif pbt < beta and pzk > z_kill and psk < s_kill:
            zu[z] = 'D'
            zu[s] = 'Z'
            Sc, Dc = Sc-1, Dc+1
            updatePostInfect(G, zu, zs, sz, s)
    zt = copy.deepcopy(zu)
    updateInteractions(G, zt, zs, sz)
```

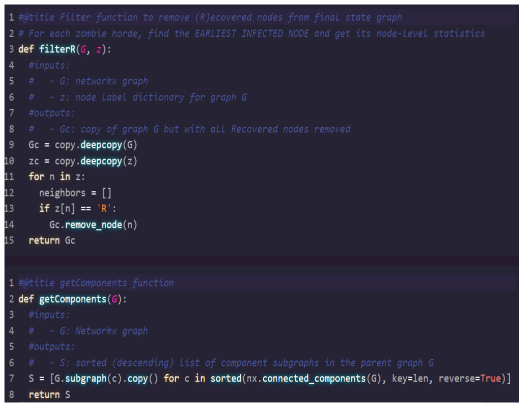
Figure 5.

Figure 5 shows a code snippet from my SZD implementation. Key things to note in the above code are the three random variables being generated: pbt, pzk, psk. Each of these represents the probability a zombie bites, zombie kills a human, or a human kills a zombie. By creating these three variables I've opened the door to simultaneously determining bites and kill outcomes. I must note however that this model is unique and not entirely related to the SZR model I'm using to recreate the results of the research paper. This is because the model in the SZR paper is based on rigorous mathematical definitions I am thoroughly unqualified to recreate as an undergraduate computer science student. The SZD model serves as my playground for the concepts explored more deeply by the SZR model.

By employing various networkx functions I managed to recreate the lattice structure albeit at a smaller scale and run my SZR code on it in the hopes of recreating the fractal structure shown in figure 2. In order to do this, I needed to first derive the critical value $\alpha_c = 0.437344654$. I elected to brute force this process and fell upon β and κ values of 0.3 and 0.13119 respectively. After deriving these values I got to work writing the driver code.

Though initially I wanted to explore the dynamics of the SZR model on various types of graphs outside the lattice structure, I found that the dynamics of the model within the structure warranted further investigation. I found linear trends in infectivity within the simulations and, additionally, noticed that the fractal patterns that resulted from running the simulation were the result of a sinister trend: borders of corpses. Borders of corpses formed as a result of humans killing zombies which prevented zombies from infecting human populations and thus created these enclaves of human society as well as these massive hordes of zombies.

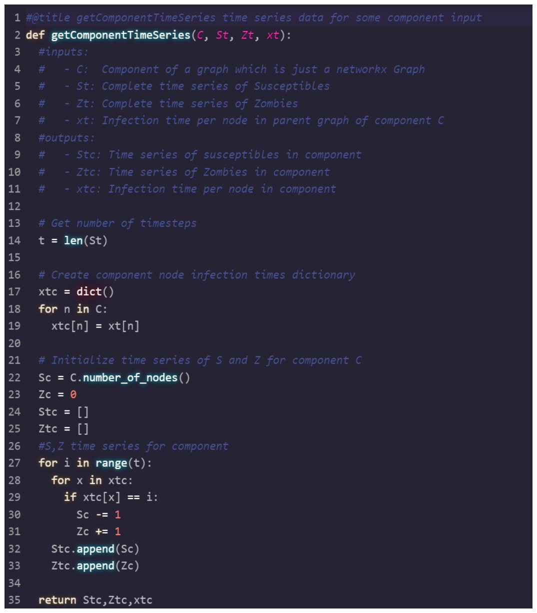
This phenomenon was not only interesting but convenient as well. The corpse borders that formed naturally separated the graph into subgraphs and I realized that removing Recovered nodes from my parent graph would yield me a set of subgraphs: human communities and zombie hordes. The human communities weren't at all interesting beyond the fantasy elements but the zombie hordes, now those were bound to have some interesting data. With this, I created some functions to partition the graph into its subgraphs by removing Recovered nodes and filtered out the human communities, leaving only the zombie hordes. By doing this I was now able to rebuild the time series of infections that happened locally as well as look at the node-level properties of patient zero within a zombie horde to hopefully glean some details as to the development of that horde structure.



```python
#@title Filter function to remove (R)ecovered nodes from final state graph
# For each zombie horde, find the EARLIEST INFECTED NODE and get its node-level statistics
def filterR(G, z):
  #inputs:
  #   - G: networkx graph
  #   - z: node label dictionary for graph G
  #outputs:
  #   - Gc: copy of graph G but with all Recovered nodes removed
  Gc = copy.deepcopy(G)
  zc = copy.deepcopy(z)
  for n in z:
    neighbors = []
    if z[n] == 'R':
      Gc.remove_node(n)
  return Gc
```

```python
#@title getComponents function
def getComponents(G):
  #inputs:
  #   - G: Networkx graph
  #outputs:
  #   - S: sorted (descending) list of component subgraphs in the parent graph G
  S = [G.subgraph(c).copy() for c in sorted(nx.connected_components(G), key=len, reverse=True)]
  return S
```

Figure 6. Component retrieval code



```python
#@title getComponentTimeSeries time series data for some component input
def getComponentTimeSeries(C, St, Zt, xt):
  #inputs:
  #   - C:  Component of a graph which is just a networkx Graph
  #   - St: Complete time series of Susceptibles
  #   - Zt: Complete time series of Zombies
  #   - xt: Infection time per node in parent graph of component C
  #outputs:
  #   - Stc: Time series of susceptibles in component
  #   - Ztc: Time series of Zombies in component
  #   - xtc: Infection time per node in component

  # Get number of timesteps
  t = len(St)

  # Create component node infection times dictionary
  xtc = dict()
  for n in C:
    xtc[n] = xt[n]

  # Initialize time series of S and Z for component C
  Sc = C.number_of_nodes()
  Zc = 0
  Stc = []
  Ztc = []
  #S,Z time series for component
  for i in range(t):
    for x in xtc:
      if xtc[x] == i:
        Sc -= 1
        Zc += 1
    Stc.append(Sc)
    Ztc.append(Zc)

  return Stc,Ztc,xtc
```

Figure 7. Component time series code

Results

Thankfully, my code proved formidable to the task of recreating the stochastic simulation results of the paper. First up, the lattice. I used a networkx grid graph function to generate a

64x64 grid of nodes on which I would run my SZR simulation code. The following is the resulting final graph.

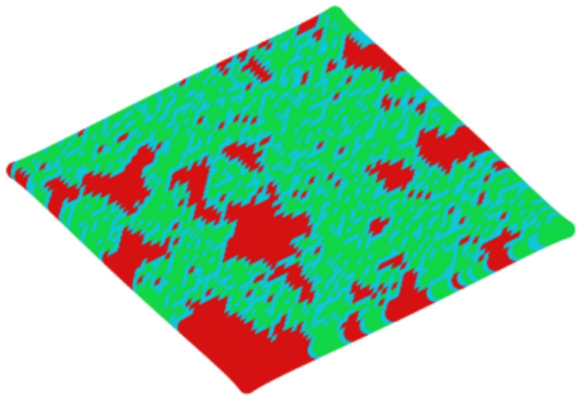
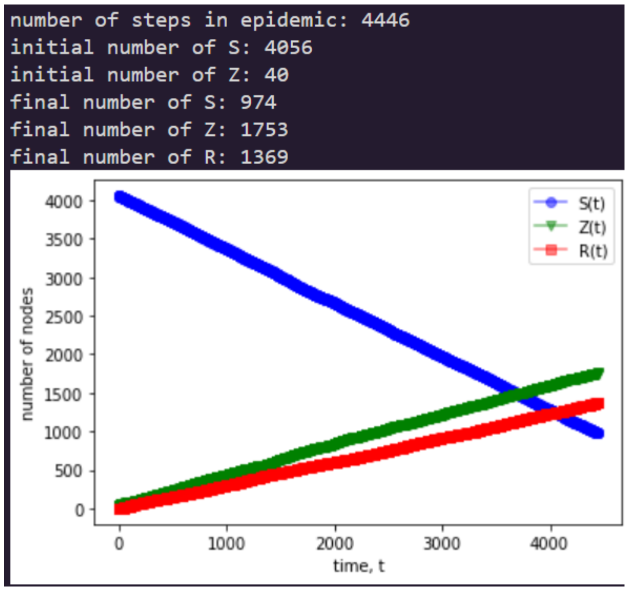


Figure 8.

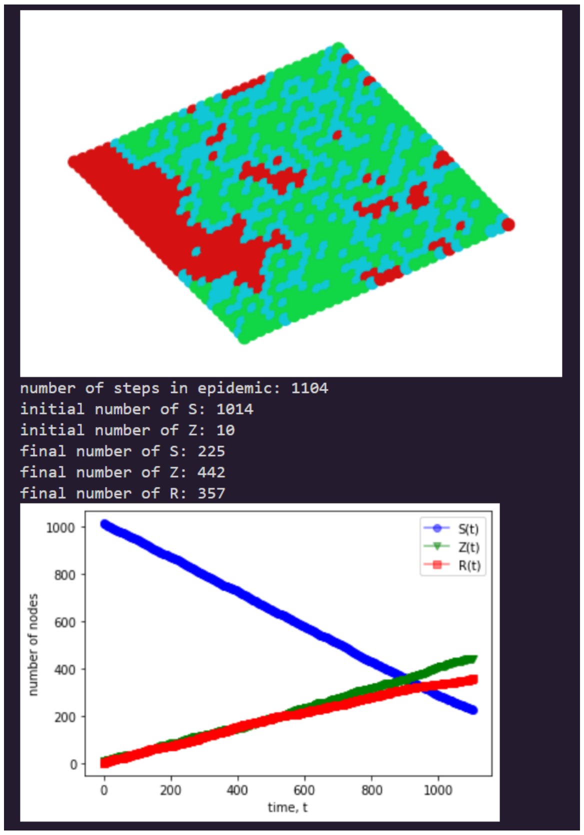
Red denotes susceptible nodes, green: zombies, and light blue nodes are recovered or in this case dead nodes. I couldn't run the full-scale simulation of a 2000x2000 node lattice as the draw calls would likely have crashed my jupyter notebook and taken a week or more to complete.



```
number of steps in epidemic: 4446
initial number of S: 4056
initial number of Z: 40
final number of S: 974
final number of Z: 1753
final number of R: 1369
```

Figure 9.

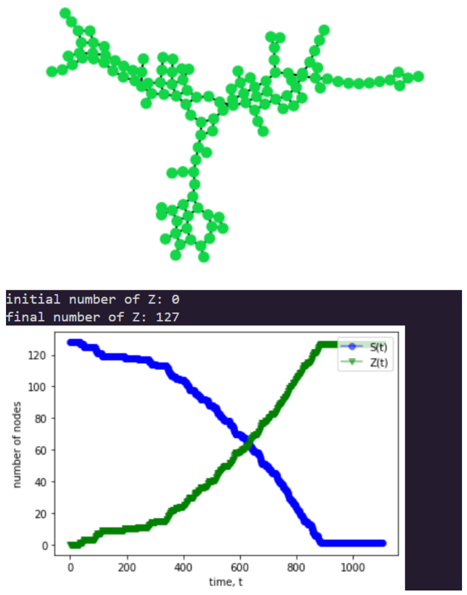
The graph above shows the number of nodes in each category per timestep of the epidemic. It was interesting to me that there was a linear trend overall and so I decided to run the simulation a few more times on smaller graphs to see if I got the same results.



```
number of steps in epidemic: 1104
initial number of S: 1014
initial number of Z: 10
final number of S: 225
final number of Z: 442
final number of R: 357
```

Figure 10.

I noticed in all my simulations that there was a linear trend when applying the SZR model to the lattice structure. In addition to this, the corpse borders mentioned earlier prompted me to do some component-level analysis of the zombie hordes to see if there were any interesting dynamics at play.



```
initial number of Z: 0
final number of Z: 127
```

Figure 11.

The visualizations above show the results of isolating and analyzing the progression of an epidemic through one zombie horde. I found that on average the degree of patient zero was around 1.9. Additionally, the trends within individual components matched the exponential trends in the SIR and projected SZR model in the original paper.

Discussion

What I found in exploring the dynamics of the SZR model was that on a large scale, it exhibited linear behavior within a lattice structure at its critical point α_c, but within its horde components it exhibited exponential behavior as would be expected of a virus. What this shows is that lattice structures when used to simulate zombie outbreaks act as case studies for the dynamics of horde formation, where a horde is defined as being a large population of zombies clustered together.

The linearity of the overall simulation is likely due to the added limitation of nodes in the Z state note ever transitioning to the R state if they aren't in contact with a human that can kill them. This limitation doesn't affect the exponential growth of hordes due to the fact that they tend to form early on during the epidemic and – due to my method of analyzing them – are already known to be sites where no zombie has died and all humans have been infected. This fact allows horde structures to behave very similarly to the early stages of a traditional SIR model exhibiting exponential growth of the infected population.

Given more time I would have liked to implement some more network-based concepts such as inference in order to simulate the partitioning of a graph into different communities or even using my existing model to infer communities onto unlabeled portion of a graph. Additionally large parts of this paper can be improved with a more rigorous understanding of the mathematics underlying SIR models.

Bibliography

You Can Run, You Can Hide: The Epidemiology and Statistical Mechanics of Zombies Alexander A. Alemi,1, ∗ Matthew Bierbaum,1, † Christopher R. Myers,1, 2, ‡ and James P. Sethna1, § 1Laboratory of Atomic and Solid State Physics, Cornell University, Ithaca, NY 14853 2 Institute of Biotechnology, Cornell University, Ithaca, New York (Dated: June 5, 2015)

Aaron Clauset Lecture Notes: Network Epidemiology (L7)

Appendix

Code for this project:
https://github.com/Bridge4/Unholy-Epidemiology